Towards Autonomous Networks: Creating and Orchestrating Intelligence for Next-Generation Network Management

Paola Andrea Soto Arenas

Tesis doctoral presentada para optar al título de Doctora en Ingeniería Electrónica y de Computación

Directores

Juan Felipe Botero Vega, Doctor (PhD)

Steven Latré, Doctor (PhD)

Universidad de Antioquia

Facultad de Ingeniería

Doctorado en Ingeniería Electrónica y de Computación

Medellín, Antioquia, Colombia

2025

Cita	Soto Arenas, 2025 [1]
Referencia Estilo IEEE (2020)	[1] Soto Arenas, P. A "Towards Autonomous Networks: Creating and Orchestrating Intelligence for Next-Generation Network Management ", Tesis Doctoral, Doctorado en Ingeniería Electrónica y de Computación, Universidad de Antioquia, Medellín, Antioquia, Colombia, 2025.

Doctorado en Ingeniería Electrónica y de Computación, Cohorte XXIV.

Grupo de Investigación en Telecomunicaciones Aplicadas (GITA)

Centro de Investigación Ambientales y de Ingeniería (CIA).

Centro de Documentación Ingeniería (CENDOI)

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.

Decano: Julio César Saldarriaga Molina.

Jefe departamento: Eduard Emiro Rodríguez Ramírez.

Jury

Chairman
prof. dr. Siegfried Mercelis, University of Antwerp, Belgium

Supervisors
prof. dr. Steven Latré, University of Antwerp, Belgium
prof. dr. Juan Felipe Botero Vega, University of Antioquia, Colombia

Mentorship
dr. Miguel Hernando Camelo Botero, University of Antwerpen, Belgium

Members
prof. dr. Jeroen Famaey, University of Antwerp, Belgium
prof. dr. Kevin Mets, University of Antwerp, Belgium
prof. dr. Paul Harvey, University of Glasgow, United Kingdom
dr. Mihai Fagadar-Cosma, Soil Capital, ex-CTO at Waylay, Belgium

Contact
Paola Soto

University of Antwerp
Faculty of Science
IDLab - Department of Computer Science
Sint-Pietersvliet 7, 2000 Antwerpen, Belgium
M: paola.soto-arenas@uantwerpen.be

Summary

Two intertwined trends have been exploited in designing and operating current Fifth Generation (5G) networks. On the one hand, the softwarization and virtualization trend brought to networks thanks to Software Defined Networking (SDN) and Network Function Virtualization (NFV) enablers. On the other hand, advancements in developing specialized hardware like Graphics Processing Units (GPUs) made computational power much more accessible and affordable, and the explosion of generated data in recent years, from social media to connected devices, have provided fuel for Artificial Intelligence (AI) algorithms to reborn and revolutionize our daily life.

SDN and NFV have introduced a paradigm shift, transforming the rigid network architecture into a flexible, service-based architecture that can be constructed ad hoc based on user demand. By decoupling hardware and software, traditional Network Functions (NFs) are evolving into software components (Virtual Network Functions (VNFs)) that can be deployed on general-purpose servers in a cloud-based manner. Softwarization and virtualization allow networks to be dynamically reconfigured and optimized for various tasks, ensuring efficient use of resources. Additionally, networks can be more easily scaled and adapted to changing demands, reducing underutilization and enhancing overall network performance. Thus, VNFs can be deployed, scaled, or removed as needed without significant hardware changes.

Besides the obvious benefits, these technologies reduce Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) by utilizing existing hardware more effectively and minimizing the need for specialized hardware for different functions. Moreover, virtualization enables quicker deployment of new services and network functions. This agility supports faster time-to-market for new features and services. Finally, network softwarization and virtualization facilitate greater automation in network management. Instead of manually managing NFs, VNFs can be managed by automated procedures, minimizing human intervention and lowering error rates.

Nevertheless, virtualizing physical resources comes with increased complexity and network management overhead. With NFV, network managers can create different network slices to serve multiple tenants, fulfilling specific requirements. Ensuring all components work seamlessly together requires sophisticated orchestration and management tools that human operators cannot provide. The traditional static Threshold (THD)-based approaches cannot cope with the variations in traffic demands and user mobility. Therefore, more sophisticated solutions are needed to manage NFV-based networks that are flexible enough to adapt to various working conditions in a fully autonomous manner.

From the perspective of AI, the networking domain has not escaped the hype, where we observe an increasing penetration of AI algorithms. Given their ability to learn from data and past experiences, AI and Machine Learning (ML) algorithms exploit the

highly vast amount of data that networks produce to replace or assist in specific NFs, composing what it is called Network Intelligence (NI). In this way, Network Intelligent Functions (NIFs) are not explicitly programmed to perform a given NF, but adapt, detect, or anticipate new requests, fluctuations, or changing conditions in the network events by learning from examples, using Supervised Learning (SL) techniques, learning from experiences, using Reinforcement Learning (RL) techniques, or by identifying patterns in the data, using Unsupervised Learning (UL) techniques.

Moreover, given the aforementioned capabilities of AI/ML functions, they are the perfect match for orchestrators and controllers, as they can react by instantiating, relocating, or re-configuring network functions fully automatedly. Thus, the evolution of network technologies towards next-generation architectures, such as 5G and beyond, necessitates a fusion between network programmability and AI, providing a paradigm shift in network management towards autonomy.

Prominent Standard-Defining Organizations (SDOs) are integrating NI into future network architectures, particularly emphasizing the closed-loop approach. However, existing methods for seamlessly integrating NI into network architectures are not yet fully effective. Current network architectures are not fully prepared to include and properly handle the promised NI or their required lifecycle. Moreover, current research focuses on enhancing model performance for specific network problems, but to achieve End-to-End (E2E) convergence, multiple NIFs must cooperate and interact, coordinating their decisions to ensure global optimality and overall stability in the autonomous network management process. Furthermore, NI solutions have been generically applied and adapted from computer science solutions rather than tailored for network management, highlighting the need for developing AI and ML algorithms specifically designed to address the unique complexities and requirements of network management functionalities.

This doctoral thesis explores the challenges mentioned above, where the creation and orchestration of tailor-made intelligence for next-generation network management is investigated. The research focuses on developing novel approaches and methodologies to enable autonomous behavior in network infrastructures, aiming to enhance efficiency, reliability, and scalability. The thesis introduces an in-depth architectural design for a Network Intelligence Stratum (NI Stratum). This stratum is supported by a novel E2E Network Intelligence Orchestrator (NIO) that supports closed-loop operations across various network domains. The primary goal of this design is to streamline the deployment and coordination of NI throughout the entire network infrastructure, tackling issues related to scalability, conflict resolution, and effective data management. We detail exhaustive workflows for managing the NI lifecycle, focusing on its compatibility and integration with current network systems and open-source platforms such as Kubernetes and European Telecommunications Standards Institute (ETSI) NFV Management and Orchestration (MANO).

Drawing upon concepts from AI and ML and their implementation in networks, this thesis proposes innovative algorithms and frameworks tailored to the unique requirements of network management. Two NI solutions are proposed to aid in managing wireless and NFV networks. Regarding wireless networks, we focused on IEEE 802.11 (Wi-Fi), one of the technologies that provides high performance with a high density of connected devices to support emerging demanding services, such as virtual and augmented reality. However, Wi-Fi performance is severely affected by interference, not

only by other Wi-Fi deployments but also by sharing the spectrum with other wireless technologies. For tackling intra-technology interference we propose a novel, data-driven approach to speed up the prediction of Wi-Fi performance after potential changes using a Graph Neural Network (GNN) model. This NI exploits the information carried in the deployment's topology and the intricate wireless interactions to predict Wi-Fi performance accurately. Inter-technology interference is handled by a distributed spectrum management framework that enhances the performance of Wi-Fi. The framework is based on a Convolutional Neural Network (CNN) that can identify different wireless technologies, provide spectrum usage statistics, detect harmful co-located wireless networks, and change the Wi-Fi's operating central frequency to avoid them.

Furthermore, this research explores the design of intelligent controllers capable of coordinating diverse NFs and adapting to changing environmental conditions in real time. An example is the NI for scaling VNFs, an automated and autonomous solution that self-adapts to the needs of network services. A Deep Reinforcement Learning (DRL) solution is proposed and compared with a classical Proportional–Integral–Derivative (PID) controller, and a THD-based algorithm for autonomously determining the amount of VNF instances to fulfill a service latency requirement without knowing or predicting the expected demand. Using a cloud-based scenario, we compare and highlight architectural differences between the different scaling methods, showing that only the data-driven approach is adaptable enough to achieve automation.

Finally, we explore the integration of NI into the network architecture, and we look at the different challenges in model development, application, and efficiently managing the lifecycle of intelligent functions. Building upon the DRL-based scaler, we propose a methodology for evaluating the training performance of these algorithms under varying Reward Functions (RFns), and secondly, validating their performance after being trained to discern the practical applicability in real-world settings. Despite significant progress, the development stage of RL techniques for networking applications, particularly in scaling scenarios, still leaves room for significant improvements.

This doctoral thesis advances autonomous network management by developing and orchestrating intelligence within networks to address next-generation network challenges. Integrating AI/ML methods with advanced network architectures, this research proposes sophisticated frameworks for continuous learning and adaptation, such as the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) closed-loop automation model. The adoption of Machine Learning Operations (MLOps) frameworks supports this transition by structuring the development, deployment, and improvement of ML models. This dissertation's findings contribute both theoretical and practical solutions to enhance the efficiency, reliability, and scalability of networks, driving innovation in network management and enabling operators to meet diverse user requirements and seize new business opportunities. Ultimately, this research highlights AI and ML's transformative potential in achieving fully autonomous networks, improving responsiveness, resource utilization, and system resilience, while providing a foundation for future intelligent network management research and shaping the future of telecommunications.

Samenvatting

Bij het ontwerpen en beheren van de huidige netwerken van de vijfde generatie (5G) wordt gebruik gemaakt van twee met elkaar verweven trends. Aan de ene kant de softwariserings- en virtualiseringstrend die netwerken hebben gekregen dankzij Software Defined Networking (SDN) en Network Function Virtualization (NFV) enablers. Aan de andere kant heeft de vooruitgang in de ontwikkeling van gespecialiseerde hardware zoals Graphics Processing Units (GPU's) rekenkracht veel toegankelijker en betaalbaarder gemaakt, en de explosie van gegenereerde data in de afgelopen jaren, van sociale media tot verbonden apparaten, heeft brandstof geleverd voor Artificial Intelligence (AI)-algoritmen om herboren te worden en een revolutie teweeg te brengen in ons dagelijks leven.

SDN en NFV hebben een paradigmaverschuiving geïntroduceerd, die de starre netwerkarchitectuur transformeert in een flexibele, servicegebaseerde architectuur die ad hoc kan worden opgebouwd op basis van de vraag van de gebruiker. Door hardware en software te ontkoppelen, evolueren traditionele Network Functions (NFs) naar softwarecomponenten (Virtual Network Functions (VNF's)) die op cloud-gebaseerde wijze kunnen worden ingezet op servers voor algemeen gebruik. Softwarisatie en virtualisatie maken het mogelijk om netwerken dynamisch te herconfigureren en te optimaliseren voor verschillende taken, waardoor een efficiënt gebruik van resources wordt gegarandeerd. Bovendien kunnen netwerken eenvoudiger worden geschaald en aangepast aan veranderende eisen, waardoor onderbezetting wordt verminderd en de algehele netwerkprestaties worden verbeterd. VNF's kunnen dus naar behoefte worden ingezet, geschaald of verwijderd zonder ingrijpende hardwareveranderingen.

Naast de voor de hand liggende voordelen verlagen deze technologieën de kapitaaluitgaven (CAPEX) en de operationele uitgaven (OPEX) door bestaande hardware effectiever te gebruiken en de behoefte aan gespecialiseerde hardware voor verschillende functies te minimaliseren. Bovendien maakt virtualisatie een snellere inzet van nieuwe diensten en netwerkfuncties mogelijk. Deze flexibiliteit ondersteunt een snellere time-to-market voor nieuwe functies en diensten. Tot slot zorgen netwerksoftwarisatie en -virtualisatie voor een grotere automatisering van het netwerkbeheer. In plaats van NF's handmatig te beheren, kunnen VNF's worden beheerd door geautomatiseerde procedures, waardoor menselijke tussenkomst wordt geminimaliseerd en het aantal fouten afneemt.

Het virtualiseren van fysieke middelen gaat echter gepaard met een verhoogde complexiteit en netwerkbeheeroverhead. Met NFV kunnen netwerkbeheerders verschillende netwerkslices creëren om meerdere huurders te bedienen, die aan specifieke eisen voldoen. Om ervoor te zorgen dat alle componenten naadloos samenwerken, zijn geavanceerde orkestratie- en beheertools nodig die menselijke operators niet kunnen leveren. De traditionele, op statische thresholds (THD) gebaseerde benaderingen kunnen niet omgaan met de variaties in verkeersbehoeften en gebruikersmobiliteit. Daarom

zijn er geavanceerdere oplossingen nodig om NFV-gebaseerde netwerken te beheren die flexibel genoeg zijn om zich volledig autonoom aan te passen aan verschillende werkomstandigheden.

Vanuit het perspectief van AI is het netwerkdomein niet ontsnapt aan de hype, waar we een toenemende penetratie van AI-algoritmen waarnemen. Gezien hun vermogen om te leren van gegevens en ervaringen uit het verleden, maken AI en Machine Learning (ML) algoritmen gebruik van de enorme hoeveelheid gegevens die netwerken produceren om specifieke NF's te vervangen of te assisteren, en vormen zo wat Network Intelligence (NI) wordt genoemd. Op deze manier zijn Network Intelligent Functions (NIFs) niet expliciet geprogrammeerd om een bepaalde NF uit te voeren, maar passen ze zich aan, detecteren ze of anticiperen ze op nieuwe verzoeken, fluctuaties of veranderende omstandigheden in de netwerkgebeurtenissen door te leren van voorbeelden, met behulp van Supervised Learning (SL)-technieken, te leren van ervaringen, met behulp van Reinforcement Learning (RL)-technieken, of door patronen te identificeren in de gegevens, met behulp van Unsupervised Learning (UL)-technieken.

Gezien de bovengenoemde mogelijkheden van AI/ML-functies passen ze perfect bij orchestrators en controllers, omdat ze volledig geautomatiseerd kunnen reageren door netwerkfuncties te instantiëren, verplaatsen of herconfigureren. De evolutie van netwerktechnologieën naar architecturen van de volgende generatie, zoals 5G en verder, vereist dus een fusie tussen netwerkprogrammeerbaarheid en AI, waardoor een paradigmaverschuiving in netwerkbeheer naar autonomie ontstaat.

Prominente standaard definiërende organisaties (SDO's) zijn NI aan het integreren in toekomstige netwerkarchitecturen, waarbij de nadruk ligt op de closed-loop benadering. De bestaande methoden voor het naadloos integreren van NI in netwerkarchitecturen zijn echter nog niet volledig effectief. De huidige netwerkarchitecturen zijn niet volledig voorbereid om de beloofde NI of hun vereiste levenscyclus op te nemen en goed te verwerken. Bovendien richt het huidige onderzoek zich op het verbeteren van modelprestaties voor specifieke netwerkproblemen, maar om End-to-End (E2E) convergentie te bereiken, moeten meerdere NIF's samenwerken en interageren en hun beslissingen coördineren om globale optimaliteit en algemene stabiliteit in het autonome netwerkbeheerproces te garanderen. Verder zijn NI oplossingen generiek toegepast en aangepast van computerwetenschappelijke oplossingen in plaats van op maat gemaakt voor netwerkbeheer, wat de noodzaak benadrukt voor het ontwikkelen van AI en ML algoritmen die specifiek zijn ontworpen om de unieke complexiteit en vereisten van netwerkbeheerfuncties aan te pakken.

Deze doctoraatsthesis onderzoekt de bovengenoemde uitdagingen, waarbij de creatie en orkestratie van op maat gemaakte intelligentie voor de volgende generatie netwerkbeheer wordt onderzocht. Het onderzoek richt zich op het ontwikkelen van nieuwe benaderingen en methodologieën om autonoom gedrag in netwerkinfrastructuren mogelijk te maken, met als doel de efficiëntie, betrouwbaarheid en schaalbaarheid te verbeteren. Het proefschrift introduceert een diepgaand architectonisch ontwerp voor een Network Intelligence Stratum (NI Stratum). Dit Stratum wordt ondersteund door een nieuwe E2E Network Intelligence Orchestrator (NIO) die closed-loop operaties over verschillende netwerkdomeinen ondersteunt. Het primaire doel van dit ontwerp is het stroomlijnen van de inzet en coördinatie van NI in de gehele netwerkinfrastructuur, waarbij problemen met betrekking tot schaalbaarheid, conflictoplossing en effectief gegevensbeheer

worden aangepakt. We beschrijven uitgebreide workflows voor het beheer van de NI-levenscyclus, waarbij we ons richten op de compatibiliteit en integratie met huidige netwerksystemen en open-source platforms zoals Kubernetes en het European Telecommunications Standards Institute (ETSI) NFV Management and Orchestration (MANO).

Op basis van concepten uit AI en ML en hun implementatie in netwerken, stelt deze dissertatie innovatieve algoritmen en frameworks voor die zijn afgestemd op de unieke vereisten van netwerkbeheer. Twee NI oplossingen worden voorgesteld om te helpen bij het beheren van draadloze en NFV netwerken. Met betrekking tot draadloze netwerken hebben we ons gericht op IEEE 802.11 (Wi-Fi), één van de technologieën die hoge prestaties levert met een hoge dichtheid van geconnecterde apparaten om opkomende veeleisende diensten te ondersteunen, zoals virtuele en augmented reality. De prestaties van Wi-Fi worden echter ernstig beïnvloed door interferentie, niet alleen door andere Wi-Fi implementaties maar ook door het delen van het spectrum met andere draadloze technologieën. Om interferentie tussen technologieën aan te pakken, stellen we een nieuwe, gegevensgestuurde aanpak voor om de voorspelling van Wi-Fi-prestaties na mogelijke veranderingen te versnellen met behulp van een Graph Neural Network (GNN) model. Deze NI maakt gebruik van de informatie in de topologie van de installatie en de ingewikkelde draadloze interacties om de Wi-Fi-prestaties nauwkeurig te voorspellen. Interferentie tussen technologieën wordt behandeld door een gedistribueerd kader voor spectrumbeheer dat de prestaties van Wi-Fi verbetert. Het raamwerk is gebaseerd op een Convolutional Neural Network (CNN) dat verschillende draadloze technologieën kan identificeren, statistieken over spectrumgebruik kan leveren, schadelijke draadloze netwerken op dezelfde locatie kan detecteren en de centrale frequentie van Wi-Fi kan wijzigen om deze te vermijden.

Verder onderzoekt dit onderzoek het ontwerp van intelligente controllers die in staat zijn om diverse NF's te coördineren en zich in realtime aan te passen aan veranderende omgevingsomstandigheden. Een voorbeeld is de NI voor het schalen van VNF's, een geautomatiseerde en autonome oplossing die zichzelf aanpast aan de behoeften van netwerkdiensten. Er wordt een Deep Reinforcement Learning (DRL) oplossing voorgesteld en vergeleken met een klassieke Proportional-Integral-Derivative (PID) controller en een THD-gebaseerd algoritme voor het autonoom bepalen van het aantal VNF-instanties om te voldoen aan een vereiste voor service latency zonder de verwachte vraag te kennen of te voorspellen. Aan de hand van een cloudscenario vergelijken we de architecturale verschillen tussen de verschillende schalingsmethoden en tonen we aan dat alleen de datagestuurde aanpak voldoende aanpasbaar is om automatisering te bereiken.

Tot slot onderzoeken we de integratie van NI in de netwerkarchitectuur en bekijken we de verschillende uitdagingen op het gebied van modelontwikkeling, toepassing en het efficiënt beheren van de levenscyclus van intelligente functies. Voortbouwend op de DRL-gebaseerde scaler, stellen we een methodologie voor om de trainingsprestaties van deze algoritmen te evalueren onder variërende Reward Functions (RFns), en ten tweede om hun prestaties te valideren nadat deze algoritmen getraind zijn om de praktische toepasbaarheid in real-world omgevingen te onderscheiden. Ondanks de aanzienlijke vooruitgang is er in het ontwikkelingsstadium van RL-technieken voor netwerktoepassingen, met name in schaalscenario's, nog steeds ruimte voor aanzienlijke verbeteringen.

Dit proefschrift bevordert autonoom netwerkbeheer door het ontwikkelen en orkestreren van intelligentie binnen netwerken om netwerkuitdagingen van de volgende generatie

aan te pakken. Door AI/ML-methoden te integreren met geavanceerde netwerkarchitecturen, stelt dit onderzoek geavanceerde frameworks voor voor continu leren en aanpassen, zoals het MAPE-K-automatiseringsmodel (Monitor-Analyze-Plan-Execute over a shared Knowledge). Het gebruik van Machine Learning Operations (MLOps) frameworks ondersteunt deze overgang door de ontwikkeling, inzet en verbetering van ML-modellen te structureren. De bevindingen van dit proefschrift dragen bij aan zowel theoretische als praktische oplossingen voor het verbeteren van de efficiëntie, betrouwbaarheid en schaalbaarheid van netwerken, waardoor innovatie in netwerkbeheer wordt gestimuleerd en operators in staat worden gesteld om te voldoen aan uiteenlopende gebruikerseisen en nieuwe zakelijke kansen te benutten. Uiteindelijk benadrukt dit onderzoek het transformatieve potentieel van AI en ML in het bereiken van volledig autonome netwerken, het verbeteren van de responsiviteit, het gebruik van middelen en de veerkracht van het systeem, terwijl het een basis biedt voor toekomstig onderzoek naar intelligent netwerkbeheer en de toekomst van telecommunicatie vormgeeft.

Resumen

En el diseño y la operacion de las actuales redes de quinta generación (5G) se han aprovechado dos tendencias entrelazadas. Por un lado, la tendencia a la softwarización y virtualización que han traído las redes definidas por software (SDN) y la virtualización de funciones de red (NFV). Por otro lado, los avances en el desarrollo de hardware especializado, como las unidades de procesamiento gráfico (GPU) haciendo que la potencia de computo sea mucho más accesible y asequible, y la explosión de datos generados en los últimos años, desde las redes sociales hasta los dispositivos conectados, han proporcionado combustible para que los algoritmos de inteligencia artificial (IA) renazcan y revolucionen nuestra vida cotidiana.

SDN y NFV han introducido un cambio de paradigma, transformando la rígida arquitectura de red en una arquitectura flexible, basada en servicios, que puede construirse ad hoc en función de la demanda de los usuarios. Al desacoplar hardware y software, las funciones de red tradicionales (NFs) están evolucionando hacia componentes de software (Virtual Network Functions [VNFs]) que pueden desplegarse en servidores de uso general de forma similar a como se hace en la nube. La softwarización y la virtualización permiten reconfigurar y optimizar dinámicamente las redes para diversas tareas, garantizando un uso eficiente de los recursos. Además, las redes pueden escalarse y adaptarse más fácilmente a las demandas cambiantes, reduciendo la infrautilización y mejorando el rendimiento general de la red. De este modo, las VNF pueden desplegarse, escalarse o eliminarse según sea necesario sin la necesidad de cambios significativos en hardware.

Además de las obvias ventajas, estas tecnologías reducen los gastos de capital (CAPEX) y los gastos operativos (OPEX) al utilizar el hardware existente de forma más eficaz y minimizar la necesidad de hardware especializado para diferentes funciones. Además, la virtualización permite un despliegue más rápido de nuevos servicios y funciones de red. Esta agilidad acelera la comercialización de nuevas funciones y servicios. Por último, la softwarización y virtualización de la red facilitan una mayor automatización en la gestión de la red. En lugar de gestionar manualmente las NF, las VNF pueden gestionarse mediante procedimientos automatizados, lo que minimiza la intervención humana y reduce las tasas de error.

No obstante, la virtualización de los recursos físicos conlleva una mayor complejidad y sobrecarga en la gestión de la red. Con la NFV, los gestores de red pueden crear diferentes redes virtuales para dar servicio a múltiples clientes, cumpliendo requisitos específicos. Garantizar que todos los componentes funcionen a la perfección requiere sofisticadas herramientas de orquestación y gestión que los operadores humanos no pueden proporcionar. Los enfoques tradicionales basados en umbrales estáticos (THD) no pueden hacer frente a las variaciones en la demanda de tráfico y la movilidad de los usuarios. Por lo tanto, se necesitan soluciones más sofisticadas para gestionar redes basadas en NFV que sean lo suficientemente flexibles como para adaptarse a diversas

condiciones de trabajo de forma totalmente autónoma.

Desde la perspectiva de la IA, el dominio de las redes no ha escapado a la moda, donde observamos una penetración cada vez mayor de los algoritmos de IA. Dada su capacidad para aprender de los datos y las experiencias pasadas, los algoritmos de IA y aprendizaje automático (ML) explotan la gran cantidad de datos que producen las redes para sustituir o ayudar en determinadas NF, componiendo lo que se denomina Inteligencia de Red (NI). De este modo, las Funciones Inteligentes de Red (NIF) no se programan explícitamente para realizar una determinada NF, sino que se adaptan, detectan o anticipan nuevas solicitudes, fluctuaciones o condiciones cambiantes en los eventos de red aprendiendo de ejemplos, mediante técnicas de Aprendizaje Supervisado (SL), aprendiendo de experiencias, mediante técnicas de Aprendizaje por Refuerzo (RL), o identificando patrones en los datos, mediante técnicas de Aprendizaje No Supervisado (UL).

Además, dadas las capacidades antes mencionadas de las funciones de IA/ML, son el complemento perfecto para los orquestadores y controladores, ya que pueden reaccionar instanciando, reubicando o reconfigurando funciones de red de forma totalmente automatizada. Así pues, la evolución de las tecnologías de red hacia arquitecturas de nueva generación, como 5G y posteriores, requiere una fusión entre la programabilidad de la red y la IA, que proporcione un cambio de paradigma en la gestión de la red hacia la autonomía.

Destacadas organizaciones de definición de estándares (SDO) están integrando las NI en las futuras arquitecturas de red, haciendo especial hincapié en el enfoque de bucle cerrado. Sin embargo, los métodos existentes para integrar perfectamente las NI en las arquitecturas de red aún no son plenamente eficaces. Las arquitecturas de red actuales no están totalmente preparadas para incluir y manejar adecuadamente las NI ni su ciclo de vida. Además, la investigación actual se centra en mejorar el rendimiento del modelo para problemas de red específicos, pero para lograr la convergencia de extremo a extremo (E2E), múltiples NI deben cooperar e interactuar, coordinando sus decisiones para garantizar la optimalidad global y la estabilidad general en el proceso de gestión autónoma de la red. Además, las soluciones de NI se han aplicado y adaptado de forma genérica a partir de soluciones informáticas en lugar de adaptarse a la gestión de redes, lo que pone de manifiesto la necesidad de desarrollar algoritmos de IA y ML diseñados específicamente para abordar las complejidades y requisitos únicos de las funcionalidades de gestión de redes.

Esta tesis doctoral explora los retos mencionados, donde se investiga la creación y orquestación de inteligencia a medida para la gestión de redes de nueva generación. La investigación se centra en el desarrollo de nuevos enfoques y metodologías que permitan un comportamiento autónomo en las infraestructuras de red, con el objetivo de mejorar la eficiencia, la fiabilidad y la escalabilidad. La tesis introduce un diseño arquitectónico en profundidad para un estrato de inteligencia de red (NI Stratum). Este estrato está soportado por un novedoso Orquestador Inteligente de Red E2E (NIO) que soporta operaciones de bucle cerrado a través de varios dominios de red. El objetivo principal de este diseño es agilizar el despliegue y la coordinación de NI en toda la infraestructura de red, abordando cuestiones relacionadas con la escalabilidad, la resolución de conflictos y la gestión eficaz de datos. Detallamos flujos de trabajo exhaustivos para gestionar el ciclo de vida de NI, centrándonos en su compatibilidad e integración con sistemas de red actuales y plataformas de código abierto como Kubernetes y NFV Management

and Orchestration (MANO) del Instituto Europeo de Estándares de Telecomunicaciones (ETSI).

Basándose en conceptos de IA y ML y su implementación en redes, esta tesis propone algoritmos y marcos de trabajo innovadores adaptados a los requisitos únicos de la gestión de redes. Se proponen dos soluciones de NI para ayudar en la gestión de redes inalámbricas y NFV. En cuanto a las redes inalámbricas, nos centramos en IEEE 802.11 (Wi-Fi), una de las tecnologías que proporciona un alto rendimiento con una alta densidad de dispositivos conectados para soportar servicios emergentes exigentes, como la realidad virtual y aumentada. Sin embargo, el rendimiento de Wi-Fi se ve gravemente afectado por las interferencias, no solo por otros despliegues Wi-Fi, sino también por compartir el espectro con otras tecnologías inalámbricas. Para hacer frente a las interferencias intratecnológicas proponemos un novedoso enfoque basado en datos para acelerar la predicción del rendimiento Wi-Fi tras posibles cambios utilizando un modelo de red neuronal basada en grafos (GNN). Esta GNN explota la información contenida en la topología del despliegue y las intrincadas interacciones inalámbricas para predecir con precisión el rendimiento Wi-Fi. Las interferencias entre tecnologías se gestionan mediante un marco de gestión del espectro distribuido que mejora el rendimiento de Wi-Fi. El marco se basa en una red neuronal convolucional (CNN) que puede identificar distintas tecnologías inalámbricas, proporcionar estadísticas de uso del espectro, detectar redes inalámbricas coubicadas perjudiciales y cambiar la frecuencia central de funcionamiento del Wi-Fi para evitarlas.

Además, esta investigación explora el diseño de controladores inteligentes capaces de coordinar diversas NFs y adaptarse a las condiciones cambiantes del entorno en tiempo real. Un ejemplo es el NI para escalar VNFs, una solución automatizada y autónoma que se adapta a las necesidades de los servicios de red. Se propone y compara un Aprendizaje Profundo por Refuerzo (DRL) con un controlador clásico Proporcional-Integral-Derivativo (PID), y un algoritmo basado en THD para determinar de forma autónoma la cantidad de VNFs necesarias para satisfacer un requisito de latencia de servicio sin conocer o predecir la demanda esperada. Utilizando un escenario basado en la nube, comparamos y destacamos las diferencias arquitectónicas entre los diferentes métodos de escalado, mostrando que sólo el enfoque basado en datos es lo suficientemente adaptable para lograr la automatización.

Por último, exploramos la integración de las NI en la arquitectura de red, y examinamos los diferentes retos en el desarrollo de modelos, la aplicación y la gestión eficiente del ciclo de vida de las funciones inteligentes. Basándonos en el escalador basado en DRL, proponemos una metodología para evaluar el rendimiento del entrenamiento de estos algoritmos bajo distintas Funciones de Recompensa (RFns) y, en segundo lugar, validar su rendimiento después de haber sido entrenados para discernir la aplicabilidad práctica en entornos del mundo real. A pesar de los importantes avances realizados, el desarrollo de las técnicas de RL para aplicaciones de red, especialmente en escenarios de escalado, aún deja margen para mejoras significativas.

Esta tesis doctoral avanza en la gestión autónoma de redes desarrollando y orquestando la inteligencia dentro de las redes para abordar los retos de las redes de próxima generación. Integrando métodos de IA/ML con arquitecturas de red avanzadas, esta investigación propone marcos sofisticados para el continuo aprendizaje y la adaptación, como el modelo de automatización de bucle cerrado Monitor-Analyze-Plan-Execute over

a shared Knowledge (MAPE-K). La adopción de marcos de Operaciones de Aprendizaje Automático (MLOps) apoya esta transición estructurando el desarrollo, despliegue y mejora de los modelos de ML. Las conclusiones de esta tesis aportan soluciones teóricas y prácticas para mejorar la eficiencia, fiabilidad y escalabilidad de las redes, impulsando la innovación en la gestión de redes y permitiendo a los operadores satisfacer las diversas necesidades de los usuarios y aprovechar nuevas oportunidades de negocio. En última instancia, esta investigación pone de relieve el potencial transformador de la IA y el ML para lograr redes totalmente autónomas, mejorar la capacidad de respuesta, la utilización de recursos y la resiliencia del sistema, al tiempo que proporciona una base para futuras investigaciones sobre gestión inteligente de redes y da forma al futuro de las telecomunicaciones.

Acknowledgements

When you reach the end of a long road, there is nothing left to do but look back, feel proud of the path you have walked, and say thank you. Completing this PhD has been a long (too long!) and transformative journey, and I am deeply grateful to those who have supported me along the way.

First and foremost, I would like to express my sincere gratitude to my advisors, Steven Latré and Juan Felipe Botero Vega. Juan Felipe, you were my advisor during my master's in Colombia, and I had so much fun doing research that I knew I wanted to pursue a PhD. Unfortunately, funding in Colombia is so scarce that I had to travel more than 8,000 kilometers to find the opportunity. That's when I met Steven. I still remember our first interview—it was during an Open Minds event, and you somehow managed to squeeze me into your always busy and chaotic schedule. Thank you both for providing the right conditions for me to grow as a researcher and for believing in me. Even with limited face-to-face interactions over the years, you trusted me, gave me the freedom to explore my ideas, and supported me in ways I will always appreciate.

I would also like to extend my gratitude to my dissertation committee members, Siegfried Mercelis, Jeroen Famaey, Kevin Mets, Paul Harvey, and Mihai Fagadar-Cosma for their insightful feedback and thoughtful discussions, which have helped shape this dissertation. Their time, effort, and expertise have been invaluable throughout this process.

I joined IDLab in 2018 as a research assistant, back when we were still at the Middelheim campus. I arrived in Belgium on February 14th, together with my colleague Nelson, filled with excitement and eagerness to continue my research journey. I remember the date so well because, coincidentally, it was Valentine's Day. IDLab was already growing rapidly, and soon, we had to move. I have seen IDLab at the university campus, at The Beacon, packed with researchers, then virtually during the pandemic, and finally, with fewer people than before. I wish I could personally thank every single one of you.

A heartfelt thank you goes to my colleagues Nelson, Pedro, Henrique, Nina, David, Julian, Esra, Daniel, Johan, Yorick, Lynn, Michelle, Ensar, Carlos, Jakob, Matthias, Johann, Andreas, Xhulio, Arno, and Raul. More than the stimulating discussions and collaborative spirit, what I truly will remember that made my time special were the beers, the coffee breaks, the jokes, and the off-topic conversations that kept us sane and made this journey enjoyable.

I would also like to express my gratitude to the mentors who have guided me along the way. Profesora Natalia Gaviria, thank you for being the first person to believe in me and open the doors that led me here. My journey into research began the day you recommended me for the DAAD scholarship. Profesor Neil Guerrero, thanks to you, I learned what research truly is. To my co-authors—Francesc, Danny, and Chia-Yu—thank you for the collaboration and fruitful discussions, and especially for the many papers

we wrote together. A special thanks to DAEMON, the project that showed me how collaborative research is done in Europe.

A particularly special acknowledgment goes to Miguel Camelo. This dissertation would not exist without your guidance. Miguelo, thank you for your patience, your time, and your dedication—invested in me without any expectation of recognition beyond the satisfaction of published papers. Thank you for the beers, the celebrations, and for lending an (attentive or not-so-attentive) ear to my endless complaints. Thank you for returning to IDLab to lead a team that once consisted of just us, but which later grew into a group that, as you liked to say, was diverse in gender but not in nationality. I always joke that half of my PhD is yours—you can now claim two and a half degrees! Truly, I have so much to thank you for.

On a personal level, to my wonderful girlfriend, Sanne—like I always say, I don't think I would have reached this stage if we hadn't met. Thank you for your endless love, for the comfort that feels so soothing after the sting of a rejection, and for supporting me when things at work don't go my way. When we started dating, you were so afraid that after finishing my PhD, I would return to Colombia and continue my life there. And look at us now—building the life of our dreams together. I can't believe how far we've come, despite all the challenges, cultural differences, and obstacles along the way. Thank you for being you, for believing in me every single day, even when I struggle to believe in myself.

To my in-laws, thank you for accepting me, listening to me, and seeing who I am despite the language barrier. Your warmth and support mean the world to me.

A mi familia, especialmente a mi madre, María Esther, a quien le debo todo—lo bueno y lo malo, pero, sobre todo, lo que me hace única. No hay palabras ni espacio suficiente para agradecerte, madre mía. A mis abuelos, Rosa y Santiago, por inculcarme siempre que el estudio viene primero. A mis tías y tíos, por verme siempre como un ejemplo, incluso cuando yo solo estaba siendo yo misma. A mis amigos, Cata, Patri, María, Sergio, Arley, Carlos, Yudy, y al maestro Carrillo, por su apoyo incondicional y por ser parte de este camino.

Finally, I want to acknowledge everyone who, in one way or another, contributed to this dissertation. Whether through academic discussions, moral support, or simply by being there, I appreciate each and every one of you. This dissertation is dedicated to all those who have believed in me and supported me through this journey.

Thank you.

Paola Soto
Antwerpen, April 2025

Table of Contents

List of Figures

List of Tables

List of Acronyms

Symbols

0MQ ZeroMQ.

3GPP 3rd Generation Partnership Project.

5G Fifth Generation.

5GC 5G Core.

5GPPP 5G Infrastructure Public Private Partnership.

6G Sixth Generation.

A

AE Autoencoder.

AF Application Function.

AI Artificial Intelligence.

AIaaS Artificial Intelligence as a Service.

ANN Artificial Neural Network.

AoI Age of Information.

AP Access Point.

API Application Programming Interface.

B

BSS Basic Service Set.

C

CAPEX Capital Expenditures.

CB Channel Bonding.

CCP Connect-Compute Platform.

CI/CD Continuous Integration, Delivery, and Deployment.

CNN Convolutional Neural Network.

COTS Commercial Off-The-Shelf.

CPU Central Processing Unit.

CSA Channel Switch Announcement.

CSMA/CA Carrier-sense Multiple Access with Collision Avoidance.

CSOI Creation Selection Optimization and Instantiation.

CSP Communication Service Provider.

CTS Clear-to-Send.

D

DataOps Data Operations.

DCB Dynamic Channel Bonding.

DCF Distributed Coordination Function.

DevOps Development Operations.

DIKW Data, Information, Knowledge, and Wisdom.

DL Deep Learning.

DNN Deep Neural Network.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

DS Direct-Sequence.

E

E2E End-to-End.

eNB Evolved Node B.

ENI Experiential Networked Intelligence.

EPC Evolved Packet Core.

ESS Extended Service Set.

ETSI European Telecommunications Standards Institute.

F

FA Frequency Analysis.

FCAPS Fault, Configuration, Accounting, Performance, and Security Management.

FH Frequency-Hopping.

FNN Feed Forward Neural Network.

FPGA Field Programmable Gate Arrays.

G

GB Gradient Boost.

GCL Graph Convolutional Layer.

GCN Graph Convolutional Network.

GNN Graph Neural Network.

GPU Graphics Processing Unit.

H

HPA Horizontal Pod Autoscaler.

HR.DS High Rate, Direct-Sequence.

I

i.i.d Independently and Identically Distributed.

IEEE Institute of Electrical and Electronics Engineers.

IMT International Mobile Telecommunications.

IoT Internet of Things.

IQ In-phase and Quadrature.

IR Infrared Light.

ISG Industry Specification Group.

ISM Industrial, Scientific, and Medical.

ITU International Telecommunication Union.

J

JSON JavaScript Object Notation.

K

K8s Kubernetes.

KPI Key Performance Indicator.

L

LLM Large Language Model.

LSTM Long Short-Term Memory.

LTE Long-Term Evolution.

LTE-LAA Long-Term Evolution Licensed Assisted Access.

LTE-U Long-Term Evolution Unlicensed.

M

M2M Machine-to-Machine.

MAC Medium Access Control.

MANO Management and Orchestration.

MAPE-K Monitor-Analyze-Plan-Execute over a shared Knowledge.

MCS Modulation and Coding Scheme.

MDAF Management Data Analytics Function.

MDAS Management Data Analytics Services.

MDP Markov Decision Process.

MEC Multi-access Edge Computing.

MILP Mixed-Integer Linear Programming.

ML Machine Learning.

MLOps Machine Learning Operations.

MRP Markov Reward Process.

N

N-MAPE-K Network Monitor-Analyze-Plan-Execute over a shared Knowledge.

NDT Network Digital Twin.

NEF Network Exposure Function.

NF Network Function.

NFV Network Function Virtualization.

NFVI Network Function Virtualization Infrastructure.

NFVO Network Function Virtualization Orchestrator.

NI Network Intelligence.

NI Stratum Network Intelligence Stratum.

NIF Network Intelligent Function.

NIF-C Network Intelligent Function Component.

NIFD Network Intelligent Function Descriptor.

NIO Network Intelligence Orchestrator.

NIS Network Intelligent Service.

NISD Network Intelligent Service Descriptor.

NN Neural Network.

NO Network Operator.

NP Non-deterministic Polynomial time.

NRF Network Repository Function.

NS Network Service.

NSAP Network and Service Automation Platform.

NWDAF Network Data Analytics Function.

O

O-CU Open Central Unit.

O-DU Open Distributed Unit.

O-RAN Open Radio Access Network.

OAM Operations, Administration, and Maintenance.

OFDM Orthogonal Frequency Division Multiplexing.

ONAP Open Network Automation Platform.

OODA Observe - Orient - Decide - Act.

OPEX Operational Expenditures.

OSM Open Source Management and Orchestration.

P

PCA Principal Component Analysis.

PHY Physical.

PI Proportional–Integral.

PID Proportional–Integral–Derivative.

PNF Physical Network Function.

PolicyIC Policy Interpreter and Configuration.

PPO Proximal Policy Optimization.

Protobuf Google Protocol Buffers.

Q

QoE Quality of Experience.

QoS Quality of Service.

R

RAN Radio Access Network.

ReLU REctified Linear Unit.

RF Random Forest.

RFn Reward Function.

RIC Radio Access Network Intelligent Controller.

RL Reinforcement Learning.

RLOps Reinforcement Learning Operations.

RMSE Root Mean-Squared Error.

RQ Research Question.

RSSI Received Signal Strength Indicator.

RT Real-Time.

RTS Request-to-Send.

S

SB3 Stable-Baselines3.

SCB Static Channel Bonding.

SD-WLAN Software Defined Wireless Local Area Network.

SDN Software Defined Networking.

SDO Standard-Defining Organization.

SDR Software Defined Radio.

SFC Service Function Chaining.

Sim-Diasca Simulation of Discrete Systems of All Scales.

SINR Signal-to-Interference plus Noise Ratio.

SL Supervised Learning.

SLA Service Level Agreement.

SMO Service Management and Orchestration.

SR Spatial Reuse.

SS Spread-Spectrum.

STA Station.

SVD Singular Value Decomposition.

SVM Support Vector Machine.

T

THD Threshold.

TLS Transport Layer Security.

TPU Tensor Processing Unit.

TR Technology Recognition.

TRPO Trust Region Policy Optimization.

Tx-Opp Transmission Opportunity.

U

UDP User Datagram Protocol.

UE User Equipment.

UL Unsupervised Learning.

USRP Universal Software Radio Peripheral.

V

vCPU virtualized Central Processing Unit.

VIM Virtual Infrastructure Manager.

VNF Virtual Network Function.

VNFM Virtual Network Function Manager.

VPN Virtual Private Network.

vRAN Radio Access Network Virtualization.

W

WG Working Group.

Wi-Fi IEEE 802.11.

WLAN Wireless Local Area Network.

X

XAI eXplainable Artificial Intelligence.

XR eXtended Reality.

Y

YAML Yet Another Markup Language.

Z

ZSM Zero-touch network and Service Management.

Chapter 1

Introduction

1.1 Context

Recently, the 3rd Generation Partnership Project (3GPP) has established standards for Fifth Generation (5G) mobile network operations [5]. These standards encompass a wide array of use cases within the same network, imposing stringent requirements regarding Quality of Service (QoS), throughput, reliability, and latency. The Sixth Generation (6G) services and applications are expected to be more demanding in terms of capacity, reliability and densification [6]. Network slicing is employed to accommodate heterogeneous services. Network slicing leverages concepts of network softwarization and virtualization, such as Software Defined Networking (SDN) and Network Function Virtualization (NFV), with the primary principle being resource sharing. This approach transforms the network into a large pool of software, storage, and computation resources shared by various actors, programmable to optimize diverse objectives. This paradigm allows network operators to meet user requirements while reducing Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) by mitigating resource underutilization. Nevertheless, challenges arise from the high heterogeneity and dynamic behavior of network actors, services, objectives, and domains.

The advent of 5G has necessitated a fundamental transformation in the management and orchestration of networks and services. Key challenges include handling the increased complexity from the shift to programmable, software-driven, service-based architectures, which is only being increased in future standards such as 6G, and achieving the operational agility required to support new business opportunities enabled by technologies like network slicing. These deployments demand a broad spectrum of capabilities, such as massive capacity, imperceptible latency, ultra-high reliability, personalized services, global reach, and extensive machine-to-machine communication.

End-to-end network and service management automation is essential for delivering services promptly and ensuring the economic viability of the diverse services offered by Communication Service Provider (CSP). The objective is to develop autonomous networks, where a large part of its operations is self-managed. An autonomous network operates independently according to business goals without human intervention, driven by high-level policies, intents, goals, and configuration data. Unlike automatic approaches, where the operational parameters are determined by an external designer

(e.g., user), autonomous networks can self-determine such parameters by executing management operations, including self-configuration, self-diagnosis, self-repair, self-healing, self-optimization, and self-protection of its resources, functions, applications, and services. These capabilities are facilitated by the network's ability to auto-discover operational information and act upon it [7].

Traditional methods for network management, based on heuristics [8] or optimization models [9], are insufficient to address the massive, dynamic, and heterogeneous nature of modern networks. This inadequacy necessitates the development of more intelligent solutions. For instance, in NFV networks, Network Services (NSs) are composed by multiple Virtual Network Functions (VNFs) in a process that is known as Service Function Chaining (SFC). A known challenge in SFC determine in which computing nodes, each of the VNFs is going to run. This problem in known as embedding. A traditional solution for this problem is by using Mixed-Integer Linear Programming (MILP). However, the problem complexity grows with the size of the substrate network. Fu et al. [10] demonstrated that Deep Reinforcement Learning (DRL) with experience replay and target network can efficiently handle complex and dynamic SFC embedding in Internet of Things (IoT) scenarios. Given the diverse range of network management tasks and their numerous functions, some Network Functions (NFs) are being replaced or supplemented by Artificial Intelligence (AI) and, more specifically, Machine Learning (ML) approaches. These AI/ML methods leverage learning from data and past experiences, thereby enhancing efficiency, reliability, and security. Consequently, many AI/ML models are being developed to automate decision-making, perform predictive analyses, manage networks proactively, enhance security, and optimize network performance [11, 12, 13]. These models are foundational in shaping the future of networks, collectively forming what is known as Network Intelligence (NI).

Network Intelligent Functions (NIFs) are not explicitly programmed to perform a given NF. Instead, they adapt, detect, or anticipate new requests, fluctuations, or changing conditions in network activities by learning from examples using Supervised Learning (SL), learning from experiences through Reinforcement Learning (RL), or identifying patterns in data with Unsupervised Learning (UL). Given the capabilities of AI/ML algorithms, they are ideally suited for integration with newly developed orchestrators and controllers, enabling fully automated instantiation, relocation, or reconfiguration of network functions [14].

In essence, AI/ML techniques are designed to assist networks by enhancing current NFs' capabilities and eliminating the need for human intervention. However, this requires a new comprehensive architecture framework designed for closed-loop automation and optimized for data-driven AI and ML algorithms. Leading Standard-Defining Organizations (SDOs) are integrating NI into future network architectures, particularly emphasizing the closed-loop approach [15, 16, 17]. For instance, the European Telecommunications Standards Institute (ETSI) has created two Industry Specification Group (ISG) for this purpose: ETSI Experiential Networked Intelligence (ENI) [18] and ETSI Zero-touch network and Service Management (ZSM) [19], which emphasize the notion of closed-loops, similar to the International Telecommunication Union (ITU) [16], as depicted in Figure 1.1.

Closed-loop automation draws inspiration from prominent reference control models for autonomic and self-adaptive systems, such as the MAPE-K [1], the Observe - Orient -

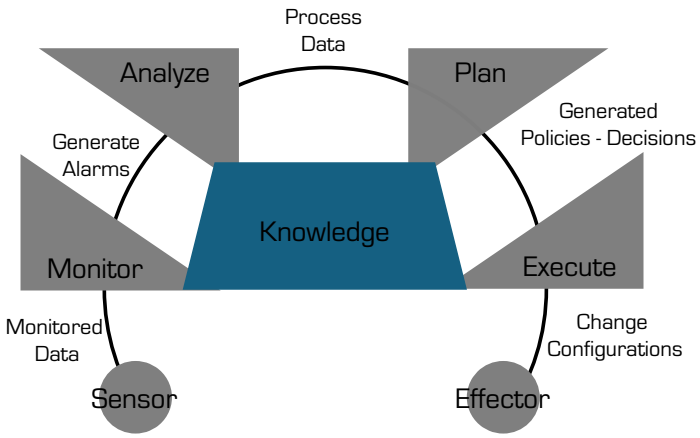


Figure 1.1: Closed-loop control proposed by Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K). Adapted from [1].

Decide - Act (OODA) model [20], and FOCALE [21]. Despite their differences, they are all composed of four main phases. We focused on the MAPE-K model since it allows us to granularly define NI (cf. Section 3.3.1). The MAPE-K model establishes a feedback loop that facilitates the dynamic adaptation and autonomous operation of complex systems, outlining the interaction of different modules as follows. *Sensors* and *Effectors* specify all the probes and Application Programming Interfaces (APIs) to gather input data or to update specific configuration parameters.

The *Monitor* block collects data from the system and its environment, observing system performance and capturing crucial metrics and events for subsequent phases. The *Analyze* block involves pre-processing, summarizing, or preparing the data for consumption by subsequent stages. Additionally, it interprets data to detect trends, anomalies, or significant patterns using techniques such as statistical analysis, clustering, and ML. Based on this analysis, the *Plan* block devises strategies for system adaptation, formulating a sequence of actions or policies to address identified issues or improve system performance. This phase often involves decision-making algorithms and optimization techniques.

The *Execute* block implements the planned actions by interacting with the system to apply the necessary changes, such as reconfiguring components, adjusting resource allocations, or modifying operational parameters. Finally, the *Knowledge* base is a repository that stores information and data used by all other components. It includes historical data, policies, system models, and rules that guide the monitoring, analysis, planning, and execution processes, ensuring coherence and consistency across the feedback loop.

In general, automating network processes without human intervention will become a must for next-generation networks due to the increased complexity, scale, and dynamic nature of these networks [22]. This also includes properly managing the lifecycle of such intelligent components. Designing, developing, and validating ML algorithms follows the workflow proposed by current Machine Learning Operations (MLOps) frameworks [23].

Based on the principles of Development Operations (DevOps), MLOps frameworks em-

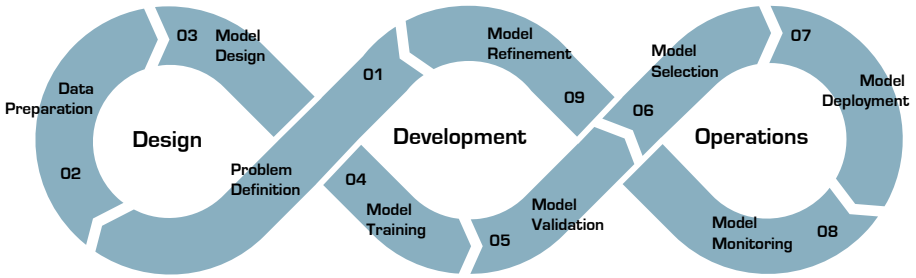


Figure 1.2: Generic MLOps workflow. Adapted from [2].

body a methodology promoting software automation via Continuous Integration, Delivery, and Deployment (CI/CD), enabling fast, frequent, and dependable releases. Both DevOps and MLOps adhere to the infinity model, prioritizing ongoing product enhancement, thus proving beneficial in software development contexts focused on integrating updates and features rather than overhauling products entirely. Figure 1.2 portrays a standard ML workflow.

The initial phase of this workflow is the design stage, where problem definition, data preparation, and model design are executed. In the problem definition stage, a mapping between the networking problem to be solve and the type of ML problem is realized to facilitate the subsequent ML model selection. For instance, the scaling of virtualized resources can be solved through SL techniques, which implies the collection, curation, and maintenance of a historical database with scaling decisions and values of the Key Performance Indicators (KPIs) collected from the managed entity. However, scaling can also be solved through RL techniques (as it will shown later in this dissertation), which implies the preparation of a sandbox [24] in which the RL agent can freely interact with a digital replica [25] of the managed system to train a scaling policy.

Additionally, relevant data sources are identified during this phase. Subsequently, data collection and preprocessing occur, ensuring the data is clean, relevant, and properly formatted. Suitable ML algorithms, techniques, and hyperparameters, such as loss/reward functions, are selected for the problem during the model design stage. The second phase involves model development, where model training and validation are performed. The ML model is trained using appropriate training data and evaluation criteria.

In the operations phase, the trained model is packaged with all necessary dependencies into a format suitable for deployment, such as Docker containers [26]. The packaged model is then deployed in a production environment, including cloud infrastructure or on-premises servers. Continuous monitoring and logging are employed to ensure the model's performance meets the desired standards by tracking relevant metrics, such as accuracy, latency, and resource usage, for performance evaluation and troubleshooting. If the model underperforms, feedback is provided to enhance the model through techniques like retraining with new data or fine-tuning hyperparameters.

Although the CI/CD and MLOps concepts are not new, the ML field is now mature enough to realize them; however its practical implementation in networks is still lacking. In conclusion, softwarization and virtualization have substantially improved network management and operations, marking a milestone in network evolution. The next step in this evolutionary process is enabled by the integration of AI and ML into modern network

architectures, enabling between levels two and three according to the autonomy levels scale [17, 15], having fully autonomous networks as the end goal in network evolution.

This integration is facilitated by sophisticated frameworks capable of continuous learning and adaptation, exemplified by the MAPE-K model and other closed-loop automation approaches. These frameworks ensure efficient and effective management of network processes with minimal human intervention, addressing contemporary networks' increasing complexity, scale, and dynamic nature. The adoption of MLOps frameworks further supports this transition by providing a structured methodology for developing, deploying, and continuously improving ML models. As the industry evolves, the principles of MLOps will be crucial in maintaining the agility and responsiveness of network services, thereby enhancing the overall performance and reliability of next-generation networks.

This dissertation is positioned at the confluence of three critical domains. On the one hand, Artificial Intelligence, specifically Machine Learning, where the primary focus of this research is to explore and develop intelligent algorithms tailored to meet the specific needs of modern networks, addressing key challenges such as interference management and the optimization of complex network operations. On the other hand, concrete network management challenges where this research aims to simplify the management processes, making networks more adaptive and resilient to changing conditions while enhancing network reliability and throughput. Finally, the control of intelligent network components, where the main focus of this dissertation is to achieve optimized network operation with minimal human intervention by developing frameworks that allow for continuous learning and adaptation, ensuring that network components can respond to emerging challenges and maximize performance.

The overarching goal of this dissertation is to advance the state of the art in network management through the integration of AI and ML, addressing practical challenges and paving the way for fully autonomous networks. This research contributes to the theoretical foundations of ML in networking and provides practical solutions to enhance future networks' efficiency, reliability, and scalability. The findings of this dissertation are expected to have significant implications for the design and operation of next-generation networks, driving innovation and enabling new business opportunities in the telecommunications industry.

1.2 Problem Statement

As outlined in the previous section, the vision of autonomous networks can be achieved by implementing AI and ML, given their potential to automate network operations and management. Recent advancements in network softwarization and virtualization, coupled with the cloud's provision of virtually unlimited storage and compute resources, have significantly contributed to this development. Additionally, the accessibility of high-performance computing platforms such as Graphics Processing Units (GPUs) and more specialized ones like Field Programmable Gate Arrayss (FPGAs) has facilitated the exponential growth in applying intelligent algorithms for enhanced network management. However, when applying ML to networking problems, several problems at the architectural level have been identified during the development of this dissertation.

[P_1] **AI and ML algorithms are often developed in isolation.** Current research primarily focuses on enhancing model performance for specific network problems, such as resource allocation. This approach results in NI functioning in isolation. However, the development of multiple NIFs necessitates cooperation and interaction among them to achieve End-to-End (E2E) convergence [27, 28]. Isolated decisions or predictions can affect other network operations areas. For example, maintaining a given energy consumption level might hinder the response time of a service by limiting the number of replicas. Therefore, it is essential for NIFs to coordinate their decisions to achieve global optimality in the zero-touch network management process, ensuring overall stability.

[P_2] **Most of the research conducted in the field of network management relies on the use of "vanilla" versions of the ML algorithms.** Historically, these algorithms have been applied in a generic manner, often adapted from solutions developed for computer science problems rather than tailored to the specific requirements of network management functionalities. For instance, the commonly used loss functions in ML may not accurately reflect errors within the network domain [29]. Furthermore, various data representations may be employed for the same input signal, illustrating the complexity of network data analysis. For example, traffic classification tasks can utilize different representations such as time series or flow sequences transformed into images [30, 31]. These examples underscore the necessity of developing AI and ML algorithms explicitly designed to address the intricacies of network management challenges.

[P_3] **AI and ML are not yet fully integrated into network architectures.** Intelligent algorithms have often been developed as an afterthought, and recent Management and Orchestration (MANO) frameworks do not adequately support these advanced algorithms. The integration of AI/ML solutions within current network architectures remains an open challenge, as evidenced by the recent establishment of two ETSI ISG [19, 18]. These initiatives introduce the concept of closed-loops [20, 1, 21] to facilitate the development of autonomous systems. However, the original MAPE-K does not distinguish between the various NI types, such as learning-based or just tuned/engineered models, or consider if the NI is being trained or it is being used for inference. In these modes, outcomes should be directed to either the actual network or its digital twin [25, 32]. Furthermore, there is a pressing need for the development of standardized frameworks to enable the seamless and interoperable deployment of AI/ML models across diverse network environments. Despite significant progress in creating AI/ML algorithms tailored to specific network tasks, there is still a lack of universally accepted standards and protocols to ensure these models' effective and efficient integration into heterogeneous network infrastructures.

[P_4] **MLOps frameworks are not fully optimized for networking.** Currently, there is a lack of comprehensive tools and methodologies for the lifecycle management of AI/ML models within network operations. Given the multiplicity of vendors, each developing distinct ML solutions to networking problems, there is a critical need for mechanisms to evaluate and select the most effective solution. This selection process must consider both operational and non-operational objectives, especially as next-generation networks demand exceptionally high performance in terms of latency and capacity while efficiently utilizing networking resources. For example, marginal gains in model accuracy often require increased computational resources,

translating into higher energy consumption. Therefore, AI/ML solutions must be designed to optimize performance and resource efficiency jointly. Furthermore, existing frameworks lack the capability to support the automated testing and validation of AI/ML models, ensuring their ongoing accuracy and reliability under changing network conditions. Addressing these gaps is essential for advancing the integration and effectiveness of AI/ML in modern network architectures.

Moreover, since this dissertation proposes two prominent use cases where ML is applied to solve critical networking problems, i.e., interference management and scaling in service-based network architectures, the following research problems were identified.

[P_5] **The densification of wireless networks, including IEEE 802.11 (Wi-Fi), worsens the fight for spectrum access.** To meet the increasing demand for higher data rates, better coverage, and enhanced user experiences, particularly in densely populated urban areas, wireless networks are being densified [33]. The unlicensed spectrum, which includes bands such as the 2.4 GHz and 5 GHz used by Wi-Fi, is limited in availability. As more small cells, other wireless technologies [34], and other wireless devices operate in these bands, the competition for spectrum increases, leading to potential congestion and interference. Consequently, effective interference management and robust regulatory frameworks are essential to ensure different devices can coexist without significant performance degradation. Advanced technologies like dynamic spectrum access, cognitive radio, and enhanced interference mitigation techniques are being developed to optimize spectrum use and improve network performance. Addressing these issues under dynamic conditions—like user mobility and coexistence with other wireless systems—requires autonomous networks.

[P_6] **The configuration of network control parameters in dense network deployments is not context-aware.** Current approaches for optimizing network configuration parameters in dense network deployments generally rely on static or semi-static models, which do not allow real-time adaptation to the fluctuating conditions of dense network environments, such as varying user density, mobility patterns, and interference levels. Traditional optimization methods may fail to account for the rapid changes and complex interactions typical in such settings, leading to suboptimal performance. On the other hand, network simulators can provide highly accurate results at the packet level. However, as the network size and parameter configurations (such as constant changes in network conditions) increase, these simulators' processing time and resource consumption grow exponentially. As a result, existing networks are difficult to analyze, monitor, and manage in near real-time. Constant assessment of the feasibility of potential system changes and their impact on user QoS and Quality of Experience (QoE) is required to ensure conflict-free configuration and optimization. Unfortunately, performance prediction models using analytical models or simulators are unsuitable due to low accuracy or high computational cost, requiring data-driven approaches to improve prediction times and accuracy.

[P_7] **Popular ML models like Convolutional Neural Networks (CNNs) do not exploit the topological information encrypted in networks.** While popular ML models such as CNNs have shown great success in various domains, they do not effectively

exploit the topological information inherent in network structures. This presents a significant research gap in applying ML to network analysis and optimization. Networks, by nature, possess rich topological features that are crucial for understanding their behavior and dynamics. CNNs, which are primarily designed for grid-like data such as images, fail to capture these complex relationships. Therefore, there is a need for developing novel ML models or adapting existing ones to integrate and leverage topological information, potentially through the use of graph-based methods. Such advancements could enhance the accuracy and efficiency of network-related tasks, including anomaly detection, traffic prediction, and network optimization. Addressing this gap would improve the performance of ML models in network contexts and open new avenues for research in network science and ML integration.

[P_8] **The need for benchmark datasets and standardized evaluation metrics for ML models in networking.** ML approaches have taken advantage of the abundance of data to solve complex problems where analytical models are intractable. They have been proven to be successful in learning from examples [35]. Unfortunately, obtaining sufficient data for ML in networks is challenging due to privacy concerns and data sparsity from real deployments [24]. Moreover, the networking community lacks benchmark datasets and standardized evaluation metrics that specifically cater to developing and assessing ML models [36]. Establishing these benchmarks would facilitate more rigorous and comparative studies in this domain, fostering the development of more robust and effective ML models that can better exploit network specificities. Synthetic data generated through network simulators or emulators [37, 38, 39] can produce datasets to effectively train ML models to adapt to real and dynamic network conditions. However, two primary challenges persist. The first challenge involves enhancing the accuracy of simulators and emulators to minimize the gap between simulated environments and real-world conditions. The second challenge pertains to the development of specialized simulators and emulators tailored to specific networking tasks [40], network domains [41], network technologies [42], or types of ML [43].

1.3 Research Questions

In the previous section, we have identified eight problems grouped into two complementary views: the architectural changes needed to adopt AI and ML natively in next-generation networks and the specific difficulties in implementing AI and ML for solving network-related problems. This chapter defines a set of five Research Questions (RQs) to target one (part) of the problem statements identified in the previous section.

[RQ_1] **How can NIFs be designed and decomposed to facilitate the coordination and reutilization of their components among multiple NIFs to achieve E2E convergence in network operations?** The concept of E2E convergence in network operations involves achieving global optimization and stability across all network layers and functions. To realize this vision, NIFs must be designed with interoperability and modularity in mind, allowing for the reutilization of components and harmonious integration with other NIFs. This approach requires the establishment of standardized interfaces and protocols that facilitate communication and

coordination among NIFs. By enabling NIFs to operate in a cooperative manner, networks can better manage complex interactions and dependencies, leading to improved performance, reduced latency, and enhanced QoS and QoE for end-users. Additionally, such a framework would support the dynamic adaptation of NFs in real-time, addressing challenges posed by user mobility, varying traffic patterns, and evolving network topologies. Ultimately, designing NIFs for coordination and reutilization is essential for advancing toward fully autonomous network management and achieving the full potential of next-generation network technologies.

[RQ_2] **What frameworks and protocols are necessary to support the seamless and interoperable deployment of AI/ML models across diverse network environments, ensuring effective integration and cooperation among various NIFs to achieve global optimality in zero-touch network management processes?** Realizing the vision of autonomous networks requires robust frameworks and protocols that can support AI/ML models' seamless and interoperable deployment across diverse network environments. Current network management frameworks such as NFV MANO provide a good starting point for achieving global optimality in network operations. However, MANO often lacks the flexibility needed to integrate AI/ML solutions effectively. For instance, MANO does not provide procedures for monitoring a NI whose performance is deteriorating due to a drift in its training data; alternatively, MANO does not provide integration with MLOps frameworks to enact ML model (re-)training. Establishing such frameworks and protocols will leverage their collective intelligence to enhance overall network performance.

[RQ_3] **How can ML algorithms be tailored to incorporate network-specifics (e.g., topological information) in their learning objectives or adapting already gained knowledge from the ML field to networking to enhance their effectiveness in network management tasks such as interference management or resource allocation?** The field of network management stands to gain significantly from the advanced capabilities of ML algorithms, yet the current applications often use their "vanilla" versions, lacking network-specific customization. Traditional ML models, originally designed for generic data tasks, frequently overlook critical aspects unique to networking, such as topological information or specific data representation. Moreover, adapting existing ML knowledge to the networking domain involves leveraging proven techniques while customizing them to address network-specific challenges. Incorporating network-specific details into ML algorithms could revolutionize tasks such as interference management and resource allocation, enabling more precise and effective solutions. For instance, integrating topological information could allow ML models to understand the network's relationships and dependencies better, leading to more accurate predictions and optimized configurations. In essence, by bridging the gap between generic ML applications and the specific needs of network management, we can unlock new levels of performance and reliability in network operations.

[RQ_4] **What methodologies can be developed to evaluate and select the most effective AI/ML models for networking problems, ensuring optimal performance and automated testing and validation under changing network conditions, enhancing current MLOps frameworks to provide comprehensive lifecycle management of such models?** The integration of AI/ML models into networking

presents a unique set of challenges, primarily due to the dynamic and complex nature of network environments. To ensure that these models deliver optimal performance, developing mechanisms that can rigorously evaluate and select the most effective models for specific networking problems is crucial. Current practices often involve manually selecting the most well-suited model according to expert knowledge, which is time-consuming and prone to errors and suboptimal choices. Automated selection and evaluation mechanisms can streamline this process, leveraging performance metrics tailored to networking tasks such as latency, throughput, and reliability. These mechanisms would enable continuous assessment of model efficiency, ensuring that the selected models can maintain high-performance levels. These evaluation systems can dynamically adjust model parameters and configurations by incorporating real-time feedback loops and adaptive learning strategies, leading to more resilient and efficient network management solutions.

[RQ_5] **How can data-driven approaches be developed for real-time, context-aware optimization of network control parameters in dense network deployments to effectively manage interference, optimize spectrum use, and ensure coexistence and high performance of various wireless devices in unlicensed bands?** The complexity of modern wireless environments, characterized by dense deployments and diverse wireless technologies, necessitates advanced mechanisms for real-time decision-making in spectrum management and interference mitigation. Data-driven approaches that leverage real-time analytics to predict and respond to changes swiftly offer a promising solution. For instance, ML models can be trained to recognize interference patterns and predict optimal spectrum allocation strategies under different conditions. These models can be continually refined with new data, improving their accuracy and effectiveness over time. Such approaches can dynamically adjust control parameters to manage interference and optimize spectrum use by continuously analyzing network conditions and context. This real-time, context-aware optimization is crucial for maintaining high performance and ensuring the coexistence of various wireless devices operating in unlicensed bands, which are prone to congestion and interference.

In general, conducting the research required to assess these RQs would yield knowledge that could be applied to broaden the interdisciplinary field of AI/ML-enhanced networking. The telco business model has changed due to softwarization in networks, giving rise to new roles like the CSP, whose goal is to create NSs that run on a shared infrastructure. Similar to this, it is anticipated that the incorporation of AI and ML into networking will give rise to new positions like ML service provider [44]. Nevertheless, networking ML is a relatively new field that requires multidisciplinary knowledge in computer/data science, communications, and ML.

While ML, for example, maximizes learning according to a loss/reward function particular to a learning problem (e.g., minimizing cross-entropy in a classification task), this optimization might not coincide with the best solution of a networking problem (e.g., choosing the optimal modulation scheme to minimize interference). Therefore, networking and ML competence are prerequisites for creating high-quality models. Furthermore, current ML development frameworks frequently prioritize applications such as chatbots, image recognition, and recommendation systems while providing insufficient assistance for networking applications [45, 46]. A slow paradigm shift from traditional non-ML

approaches to ML-based or hybrid solutions [47] is facilitated by this lack of qualified professionals.

1.4 Hypotheses

The rapid advancement of network technologies, particularly with the emergence of 5G and beyond, has increased the complexity of networks, necessitating the integration of sophisticated AI/ML models to enhance network management and operations. However, current AI/ML implementations in networking often fall short due to their generic design and their lack of integration with well-known management frameworks. This gap highlights the need for tailored AI/ML algorithms that can effectively address networking challenges like interference management and resource allocation. On the other hand, the existing MLOps frameworks, which are optimized for managing ML pipelines, are not fully optimized for the unique demands of network environments, lacking comprehensive lifecycle management tools that ensure optimal performance through automated testing and validation. This section formulates the following hypotheses based on the identified problems and the corresponding RQs to address these issues.

[H_1] **Autonomic computing frameworks based on closed-loop control could potentially decompose NI solutions, thereby enhancing their coordination and interaction among multiple NIFs. This approach will improve E2E convergence and global optimization in network management, addressing the current limitations in MANO frameworks and facilitating the development of autonomous systems.** To facilitate self-X capabilities for autonomous networks (where "X" can represent adaptation, configuration, management, healing, and optimization), decision-making engines should adhere to the closed-loop control suggested by frameworks for autonomic computing. Furthermore, intelligent algorithms might be essential to this closed loop at one or more stages. Perceptive algorithms enable sensors and monitoring systems to automatically detect disruptions, diagnose issues, and initiate corrective actions without interfering with network operations. Using AI-enhanced analytical techniques, it is possible to find hidden patterns in data, aggregate multiple data sources, and decrease the dimensionality of data flowing through the network. AI-based decision-making can use high-level rules to dynamically adapt to changes in the environment, adding or removing components and maintaining the adaptability and resilience of the system. NI solutions might be broken down into smaller atomic parts by precisely defining the scope in each step of this closed-loop control. This would allow the intelligence to be located in one or more of the blocks within a MAPE-K loop. Regardless of how simple or complicated the NI solution is, NI solutions can be composed in this manner, just as VNFs are composed to generate a NS [48]. In addition, after breaking down the NI solution, it is possible to identify functional similarities between several NI solutions. This might lead to the reuse of components, which could save computational and network resources and enhance coordination amongst NI solutions.

[H_2] **Developing context-aware, data-driven strategies would allow networks to enhance real-time adaptation to fluctuating conditions and improve network and computation resource utilization. As such, these models would boost overall**

network performance, reducing latency and minimizing interference without compromising users QoS and QoE. While there are numerous algorithms that can suit the functional characteristics of every block in the MAPE-K loop, data-driven methods stand out due to their capacity for continuous learning. Heuristic approaches usually have operating parameters that are fine-tuned to function in extremely particular circumstances. In the event that conditions alter, the heuristic parameters must be manually adjusted following a lengthy and costly computing process that mostly uses brute force techniques. However, during runtime, ML-based algorithms have the ability to modify or adjust their learned model, that is, the parameters of their model architecture. This is particularly powerful because it does not require the algorithm to be completely retrained in order to discover new patterns under different circumstances. Furthermore, intelligent algorithms that are tailored to use the network context—such as its topology—can anticipate outcomes with greater accuracy in extremely complex contexts than those of existing analytical or system models. These features can be applied to high-level applications where measuring the impact of potential modifications in real time and beforehand is necessary.

[H_3] **Enhancing current network management frameworks with integrated platforms and protocols for lifecycle management of NI solutions—including support for various ML types (e.g., SL, RL) and modes (e.g., training, testing)—will improve model effectiveness and network performance. Additionally, establishing benchmark datasets and standardized evaluation metrics for ML models in networking will facilitate rigorous research, leading to more robust and effective models that better exploit network specificities.** ML-based algorithms are not supported by the network management frameworks that are currently in use, such as NFV MANO. To not reinvent the wheel, these frameworks could be enhanced by offering a suitable interface for interacting with more specialized frameworks, such as MLOps, which handle the lifecycle of standard ML pipelines [49, 50]. However, in order to operate with networks, such MLOps frameworks should also be modified. For example, instead of utilizing widely used evaluation metrics like accuracy or F1 scores, NI solutions should be evaluated for the performance attained in the network-related metrics. Conversely, ML approaches rely on data, meaning that the models they use are fed with representative data of the environment in which the ML model will be used. The network generates a lot of data, but because privacy regulations usually protect this data, research must rely on the quality and availability of synthetic data. This will significantly hinder ML's adoption in networking. Consequently, there is a need for large, publicly available, high-quality datasets on which NI solutions can be trained, creating benchmarks for fair comparison. Similar actions were done in the RL community with its standard environments [51] and the computer vision community with its open datasets [52, 53, 54], fostering rigorous research and the development of more powerful algorithms.

In summary, the relationship between the problem statements as outlined in Section 1.2, the Research Question as developed in Section 1.3, and the hypothesis considered in this section is shown in Table 1.1. In order to address the problem statements P_1 and P_3, hypothesis H_1 proposes an architectural blueprint for autonomous networks, in which intelligent components might be readily created modularly, easing their orchestration and coordination. Answers to RQ_1 and RQ_2 would provide compelling evidence for this hypothesis. Moreover, hypothesis H_2 predicts more precise, specialized ML-based

Table 1.1: Relationship between Problem Statements, Research Questions and Hypotheses

Research Questions	Hypotheses		
	H1	**H2**	**H3**
RQ1	P1	-	-
RQ2	P3	-	-
RQ3	-	P2, P7	-
RQ4	-	-	P4, P8
RQ5	-	P5, P6	-

networking algorithms that take advantage of its contextual data, concentrating on problem statements P_2, P_5, P_6, and P_7. The responses to RQ_3 and RQ_5 will offer evidence supporting this hypothesis. Lastly, hypothesis H_3 addresses problem statements P_4 and P_8 by predicting suitable techniques for NI lifecycle management that take into account both the behavior of the ML models and their influence on network-related metrics. The investigation into RQ_5 will support this theory.

1.5 Dissertation Scope and Contributions

This dissertation focuses on the use of ML techniques in the network management domain. We begin the dissertation by defining how the Network Intelligence should be natively integrated in future network architectures. For this purpose, we present the complete architectural design and supporting procedures for a Network Intelligence Stratum (NI Stratum), an E2E orchestrator for next-generation networks. This includes the decomposition of NI solutions using an extension of the known MAPE-K framework, the definition of internal and external interfaces based on ETSI NFV MANO, and detailed workflows for inter- and intra-Network Intelligence Orchestrator (NIO) procedures, confirming that autonomic computing frameworks can effectively enhance coordination and optimization in network management, addressing limitations in current MANO frameworks.

Based on the NI Stratum, this dissertation provides two NI approaches to tackle two known network management challenges. On the one hand, we look at wireless networks in the sub-6GHz band. Since Wi-Fi is still one of the preferred technology for broadband access in unlicensed bands, we focus on two interference management use cases. Firstly, we investigate the interference caused by Wi-Fi deployments that belong to the same management domain (intra-technology interference). As such, different Wi-Fi deployments can potentially interfere with each other if their configurations, i.e., the initial channel allocation, are not properly optimized. Secondly, we assume legacy operation of two competing wireless technologies, i.e., Wi-Fi and Long-Term Evolution Unlicensed (LTE-U), leading to severe Wi-Fi throughput deterioration (inter-technology interference). This dissertation proposes two NI approaches for interference management: a Graph Neural Network (GNN) model that uses network topology to predict dense wireless deployment performance and a distributed spectrum management framework enhanced by AI to improve coexistence between Wi-Fi and Long-Term Evolution (LTE). These data-driven approaches provide real-time adaptation to fluctuating condi-

tions, demonstrating improved spectrum utilization, reduced throughput loss, and the ability to minimize interference without compromising user QoS and QoE.

On the other hand, we look at scaling, a fundamental MANO operation in service-based architectures that allows CSPs to automatically resize NSs at runtime to meet traffic surges while providing performance guarantees. NI is needed in scaling since current approaches for scaling are based on Central Processing Unit (CPU) usage. However, under this approach, the correlation between Service Level Agreement (SLA), QoS parameters, scaling decision triggers, and learning metrics is not known, or it is difficult to model. This dissertation introduces a scaling method using DRL, demonstrating reduced SLA violations while maintaining control over VNFs replicas. The chapter evaluates three auto-scaling methods (RL-, Proportional–Integral–Derivative (PID)-, and Threshold (THD)-based) and proposes a methodology to adapt well-known RL problems to auto-scaling, showing that data-driven algorithms enhance real-time adaptation and automation in network operations, improving overall network performance and minimizing interference without compromising QoS and QoE.

Finally, this dissertation provides a methodology for designing, training, validating, and selecting DRL algorithms with different Reward Functions (RFns) and its implementation in networking, aimed at enhancing MLOps frameworks by considering both learning and network-related metrics. Using scaling as a use case, this dissertation details the development of intelligent scalers based on multiple DRL algorithms. For each scaling agent, we design three RFns suitable to the scaling problem, providing a broader spectrum of potential solutions and leading to more robust and adaptive learning. Moreover, we incorporate practices for algorithm selection and comparison based on learning and networking metrics, extending the current implementation of MLOps practices, which are focused on single-model optimization. Additionally, we establish a benchmark for evaluating scaling agents based on RL algorithms, integrating with well-known frameworks in RL research.

1.6 Dissertation Outline

This dissertation is structured into seven chapters, as depicted in Figure 1.3. This chapter provides the introduction. Chapter 2 presents the foundational concepts essential for understanding this dissertation, including the basic principles of wireless communications, programmable networks, and Machine Learning (ML). Notice that a review of the state-of-the-art techniques is provided in each technical chapter (i.e., Chapters 3, 4, 5, and 6) rather than in Chapter 2. Chapter 3 introduces the architectural design of a NI Stratum, the internal and external interfaces and the procedures needed for the native integration of AI and ML solutions into network architectures and that allows the orchestration of intelligent solutions. Building on these foundations, Chapter 4 discusses two NI solutions for interference management in wireless networks. Similarly, Chapter 5 presents a NI solution for scaling in service-based architectures using DRL. From the perspective of orchestrating NI, Chapter 6 builds on scaling, considering that different NI algorithms can be developed by different companies for controlling the number of replicas. As contribution, Chapter 6, proposes a methodology for training, validating, and selecting NI algorithms. Finally, Chapter 7 concludes the dissertation by reviewing our contributions and their link with the proposed RQs and hypotheses, as well as

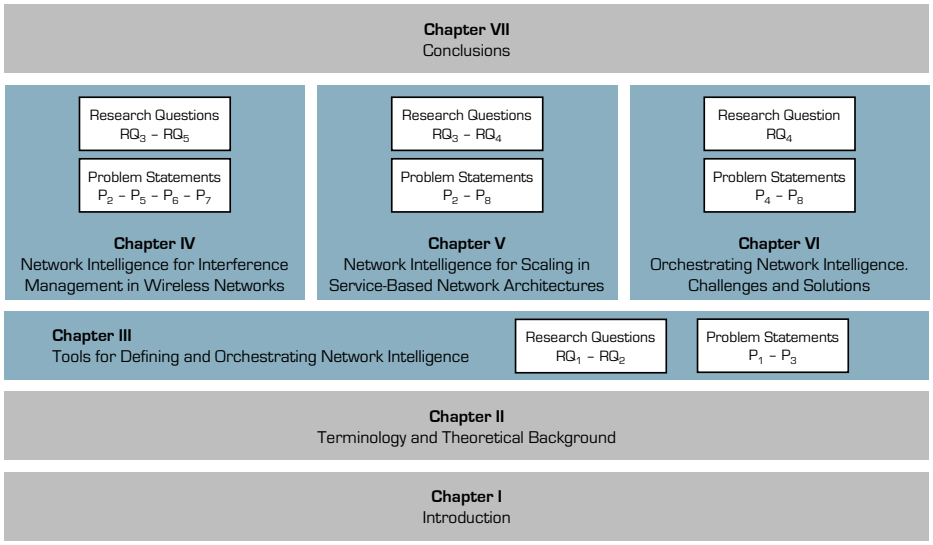


Figure 1.3: Organization of this dissertation. The chapters containing the main contributions are highlighted in blue. The relationship between problem statements, Research Questions and contributions is also indicated.

offering future research perspectives.

1.7 List of Publications

The contributions made during this research have been published and presented in several international, peer-reviewed scientific journals and conferences. The following list provides an overview of the publications during the development of this dissertation.

1.7.1 A1: Journal publications indexed by ISI Web of Science "Science Citation Index Expanded"

1. **P. Soto**, M. Camelo, K. Mets, F. Wilhelmi, D. Góez, L. A. Fletscher, N. Gaviria, P. Hellinckx, J. F. Botero, and S. Latré, "ATARI: A graph convolutional neural network approach for performance prediction in next-generation WLANs," Sensors, vol. 21, no. 13, p. 4321, 2021 [IF 3.847]

2. F. Wilhelmi, D. Góez, **P. Soto**, R. Vallés, M. Alfaifi, A. Algunayah, J. Martín-Pérez, L. Girletti, R. Mohan, K. V. Ramnan et al., "Machine learning for performance prediction of channel bonding in next-generation ieee 802.11 wlans," ITU Journal on Future and Evolving Technologies, vol. 2, no. 4, pp. 67–79, 2021.

3. M. Camelo, **P. Soto**, and S. Latré, "A general approach for traffic classification in wireless networks using deep learning," IEEE Transactions on Network and Service Management, vol. 19, no. 4, pp. 5044–5063, 2021. [IF 4.758]

4. D. Góez, **P. Soto**, S. Latré, N. Gaviria, and M. Camelo, "A methodology to design quantized deep neural networks for automatic modulation recognition," Algorithms, vol. 15, no. 12, p. 441, 2022. [IF 2.3]

5. **P. Soto**, M. Camelo, D. De Vleeschauwer, Y. De Bock, C.-Y. Chang, J. F. Botero, and S. Latré, "Network Intelligence for NFV Scaling in Closed-Loop Architectures," IEEE Communications Magazine, vol. 61, no. 6, pp. 66–72, 2023. [IF 8.3]

6. M. Farreras, **P. Soto**, M. Camelo, L. Fàbrega, and P. Vilà, "Improving network delay predictions using GNNs," Journal of Network and Systems Management, vol. 31, no. 4, p. 65, 2023. [IF 3.6]

7. N. Slamnik-Kriještorac, M. Camelo, C.-Y. Chang, **P. Soto**, L. Cominardi, D. De Vleeschauwer, S. Latré, and J. M. Marquez-Barja, "AI-empowered management and orchestration of vehicular systems in the beyond 5G era," IEEE Network, vol. 37, no. 4, pp. 305–313, 2023. [IF 9.3]

8. **P. Soto**, M. Camelo, G. Garcia-Aviles, E. Municio, M. Gramaglia, E. Kosmatos, N. Slamnik-Kriještorac, D. De Vleeschauwer et al., "Designing the Network Intelligence Stratum for 6G Networks," Computer Networks, vol. 254, p. 110780, 2024. [IF 5.6]

9. D. De Vleeschauwer, C.-Y. Chang, **P. Soto**, Y. De Bock, M. Camelo, and K. De Schepper, "A method to compare scaling algorithms for cloud-based services," IEEE Transactions on Cloud Computing, 2024. [IF 5.3]

1.7.2 A2: Non-peer-reviewed Journal Articles

1. **P. Soto**, M. Camelo, D. De Vleeschauwer, Y. De Bock, N. Slamnik-Kriještorac, C.-Y. Chang, N. Gaviria, E. Mannens, J. F. Botero, and S. Latré, "Designing, Developing, and Validating Network Intelligence for Scaling in Service-Based Architectures based on Deep Reinforcement Learning," arXiv preprint arXiv:2405.04441, 2024.

1.7.3 P1: Proceedings included in the ISI Web of Science "Conference Proceedings Citation Index - Sciences"

1. **P. Soto**, M. Camelo, J. Fontaine, M. Girmay, A. Shahid, V. Maglogiannis, E. De Poorter, I. Moerman, J. F. Botero, and S. Latré, "Augmented Wi-Fi: An AI-based Wi-Fi management framework for Wi-Fi/LTE coexistence," in IEEE International Conference on Network and Service Management (CNSM). IEEE, 2020, pp. 1–9.

2. M. Camelo, T. De Schepper, **P. Soto**, J. Marquez-Barja, J. Famaey, and S. Latré, "Detection of traffic patterns in the radio spectrum for cognitive wireless network management," in IEEE International Conference on Communications (ICC). IEEE, 2020, pp. 1–6.

3. **P. Soto**, D. De Vleeschauwer, M. Camelo, Y. De Bock, K. De Schepper, C.-Y. Chang, P. Hellinckx, J. F. Botero, and S. Latré, "Towards autonomous VNF auto-scaling using deep reinforcement learning," in Eighth International Conference on Software Defined Systems (SDS). IEEE, 2021, pp. 01–08.

4. N. Slamnik-Kriještorac, **P. Soto-Arenas**, M. C. Botero, L. Cominardi, S. Latré, and J. M. Marquez-Barja, "Realistic experimentation environments for intelligent and distributed management and orchestration (MANO) in 5G and beyond," in IEEE 19th Annual consumer communications & networking conference (CCNC). IEEE, 2022, pp. 943–944.

5. M. Camelo, L. Cominardi, M. Gramaglia, M. Fiore, A. Garcia-Saavedra, L. Fuentes, D. De Vleeschauwer, **P. Soto-Arenas**, N. Slamnik-Krijestorac, J. Ballesteros et al., "Requirements and Specifications for the Orchestration of Network Intelligence in 6G," in IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2022, pp. 1–9.

6. M. Farreras, **P. Soto**, M. Camelo, L. Fàbrega, and P. Vilà, "Predicting network performance using GNNs: generalization to larger unseen networks," in IEEE/IFIP Network Operations and Management Symposium. IEEE, 2022, pp. 1–6.

7. M. Camelo, M. Gramaglia, **P. Soto**, L. Fuentes, J. Ballesteros, A. Bazco-Nogueras, G. Garcia-Aviles, S. Latré, A. Garcia-Saavedra, and M. Fiore, "Daemon: A network intelligence plane for 6G networks," in IEEE Globecom Workshops (GC Wkshps). IEEE, 2022, pp. 1341–1346.

8. J. F. N. Pinheiro, C.-Y. Chang, T. Collins, E. Smekens, R. Berozashvili, A. Shahid, D. De Vleeschauwer, **P. Soto**, I. Moerman, J. Marquez-Barja et al., "5GECO: A cross-domain intelligent neutral host architecture for 5G and beyond," in IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2023, pp. 1–6.

9. L. E. Chatzieleftheriou, M. Gramaglia, M. Camelo, A. Garcia-Saavedra, E. Kosmatos, M. Gucciardo, **P. Soto**, G. Iosifidis, L. Fuentes, G. Garcia-Aviles et al., "Orchestration procedures for the network intelligence stratum in 6G networks," in European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit). IEEE, 2023, pp. 347–352.

10. J. Jimenez, **P. Soto**, D. De Vleeschauwer, C.-Y. Chang, Y. De Bock, S. Latre, and M. Camelo, "Resource Allocation of Multi-User Workloads in Cloud and Edge Data-Centers Using Reinforcement Learning," in 19th International Conference on Network and Service Management (CNSM). IEEE, 2023, pp. 1–5.

11. T. Avé, **P. Soto**, M. Camelo, T. De Schepper, and K. Mets, "Policy compression for low-power intelligent scaling in software-based network architectures," in NOMS 2024-2024 IEEE Network Operations and Management Symposium, IEEE, 2024, pp. 1–7.

12. L. E. Chatzieleftheriou, M. Gramaglia, M. Fiore, N. Slamnik-Krijestorac, M. Camelo, **P. Soto**, E. Kosmatos, A. Garcia-Saavedra, M. Gucciardo et al., "Network intelligence in action: the DAEMON perspective," in European Conference on Networks and Communications & 6G Summit, 2024, pp. 1–6.

Chapter 2

Background on Machine Learning and Wireless Communications

In the landscape of technology and industry, two significant trends have emerged over the past two decades, reshaping the fabric of networks and systems. The programmability and softwarization of networks, propelled by breakthroughs like Software Defined Networking (SDN) and Network Function Virtualization (NFV), have ushered in an era of enhanced configurability and openness. Simultaneously, the proliferation of Artificial Intelligence (AI) has catalyzed what is now referred to as the "summer of AI," revolutionizing various sectors. It is at the intersection of these trends that this dissertation finds its genesis.

However, alongside these advancements of SDN and NFV comes the challenge of managing increasingly complex networks characterized by a diverse array of vendors and devices. Thus, given the intricate nature of modern networks, AI, particularly Machine Learning (ML)-based solutions, harness vast operational data to inform management decisions across networks, data centers, and clouds. Moreover, these adaptive methods demonstrate agility in navigating the ever-evolving conditions of contemporary networks.

This chapter reviews the theoretical underpinnings essential for comprehending this dissertation's scope and objectives. We offer a concise overview of wireless communication principles, providing foundational knowledge crucial for our subsequent discussions. Subsequently, we explore the paradigm of network programmability facilitated by SDN and NFV advancements, examining their implications for network management. Finally, we embark on an introductory exploration of the captivating realm of ML, elucidating its potential to revolutionize network management practices.

Nonetheless, this chapter does not undertake a review of related literature nor a detailed explanation of the many wireless transmission protocols. Each subsequent chapter will include a section dedicated to addressing the relevant work. Numerous surveys have been conducted that comprehensively review the prominent applications of ML in networking. For instance, Boutaba et al. [11] present the application of diverse ML techniques in various of areas networking across different network technologies. Particularly, it reviews different learning paradigms and ML techniques applied to fundamental problems in networking, including traffic prediction, routing and classification, congestion control, resource and fault management, Quality of Service (QoS), and Quality of Experience

(QoE) management, and network security. Similarly, Ayoubi et al. [12] explore the applicability of ML in network management strategies. Furthermore, building upon IBM's well-known Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) [1], Ayoubi et al. propose the C-MAPE, a cognitive control loop integrating learning into each function of the MAPE-K. In a complementary vein, Coronado et al. [13] align with the Zero-touch network and Service Management (ZSM) perspective to review recent ML applications in autonomous network management.

2.1 Wireless Networking: A primer on Wi - Fi

The content of this section is partially based on information that can be found in the following references:

[55] M. S. Gast, 802.11 Wireless Networks: The Definitive Guide. O'Reilly Media, Inc., 2005. Second Edition.

[56] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher Speed Physical Layer (PHY) Extension in the 2.4 GHz band," IEEE, Standard, 2000.

[57] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6GHz," IEEE, Standard, 2013.

[58] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: Enhancements for High-Efficiency WLAN," IEEE, Draft, 2021.

Like all networks, wireless networks transmit data over a medium. The medium, in this case, is the open space, using a form of electromagnetic radiation. To be well-suited for use on wireless networks, the medium must be able to cover a wide area so clients can move throughout a coverage area. Radio waves are the most used form of wireless transmission since they can penetrate most obstructions and offer a wider coverage range. Wireless devices are constrained to operate in a certain frequency band. The use of a radio spectrum is rigorously controlled by regulatory authorities through licensing processes. To prevent overlapping uses of the radio waves, frequency is allocated in bands, which are simply ranges of frequencies available to specified applications. A broad division of the spectrum defined by the International Telecommunication Union (ITU) [59] is given in Table 2.1.

From the entire spectrum, there are some portions that are internationally reserved for Industrial, Scientific, and Medical (ISM) purposes. The use of such bands is not licensed, meaning that users do not need specific authorization to operate them. Some common frequencies within the ISM bands include 2.4 GHz and 5.8 GHz, which are widely used for technologies like IEEE 802.11 (Wi-Fi) and Bluetooth. Independently of the technology, wireless networks operate thanks to the principle of radio propagation, governed by the law of electromagnetism. A wireless link is a one-to-many connection between a transmitter and (several) receivers through the air. The transmitter converts

Table 2.1: Spectrum division defined by the ITU.

Electromagnetic waves	Acronym	Frequency	Wavelength
Extremely low frequency	ELF	30–300 Hz	10–1,000 km
Ultra low frequency	ULF	300–3,000 Hz	1–100 km
Very low frequency	VLF	3–30 kHz	100–10 km
Low frequency	LF	30–300 kHz	10–1 km
Medium frequency	MF	300–3,000 kHz	1,000–100m
High frequency	HF	3–30MHz	100–10m
Very high frequency	VHF	30–300MHz	10–1m
Ultra high frequency	UHF	300–3,000MHz	100–10 cm
Super high frequency	SHF	3–30 GHz	10–1 cm
Extreme high frequency	EHF	30–300 GHz	10–1mm
Tremendously high frequency (Terahertz radiation)	THF	300–3,000 GHz	1–0.1mm
Infrared rays	IR	43,000–416,000 GHz	7–0.7μm
Visible light		430,000–750,000 GHz	0.7-0.4 mum
Ultraviolet	UV	750,000–3,000,000 GHz	0.4–0.1 μm

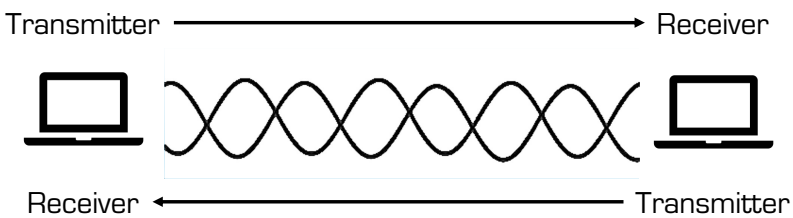


Figure 2.1: General wireless system. The communication can be dual.

the data into electromagnetic waves, which are emitted to the surrounding space thanks to the antennas. Then, the emitted radio waves travel through the air in straight lines until they encounter their destination or obstacles. From then on, the waves can be absorbed or reflected, losing part of the transmission power and reducing the coverage range [60].

To be able to communicate, both devices must use the same frequency. If a large number of wireless devices communicate simultaneously and use the same radio frequency, it can cause interference with each other. This is why wireless communication is standardized by the Institute of Electrical and Electronics Engineers (IEEE) in its 802 technical standards series. Specifically, the IEEE 802.11 standard refers to Wi-Fi, while the IEEE 802.15 refers to Bluetooth. The IEEE 802.11 standard is related to the Medium Access Control (MAC) layer and the Physical (PHY) layer. Figure 2.2 shows the MAC layer and the PHY layer standards. Several PHY layers have been defined across the years. The Infrared Light (IR) PHY uses infrared light instead of radio waves; however, this PHY does not offer any of the big drivers of the 802.11 family, i.e., flexibility and mobility, due to the properties of light transmission. Thus, its use was discontinued as a broadband transmission technology.

Frequency-Hopping (FH), Direct-Sequence (DS) and High Rate, Direct-Sequence (HR.DS) are based on Spread-Spectrum (SS). SS diffuses the signal power over a large range of frequencies. The receiver performs the inverse operation, allowing the reconstruction of

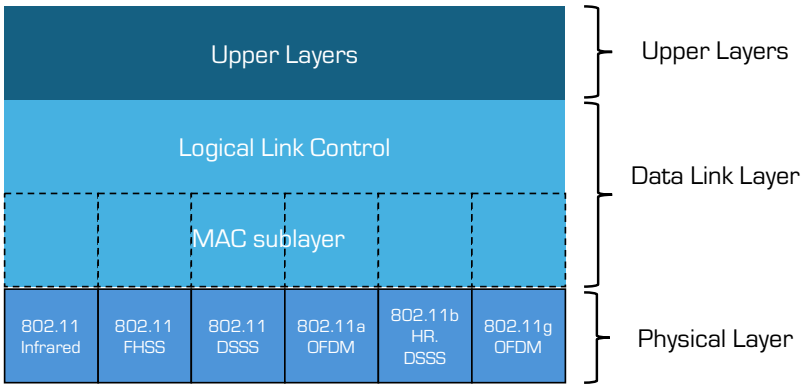
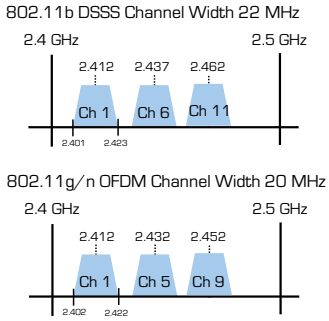
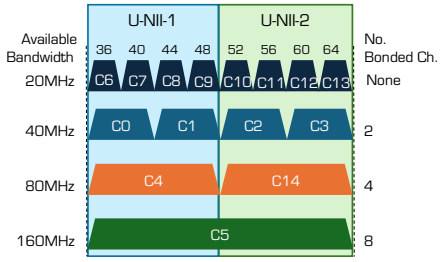


Figure 2.2: MAC sublayer and corresponding physical layer standards within the IEEE 802.11 protocol suite.



(a) Non-overlapping channels for 2.4 GHz with the most used PHY layers.



(b) 5 GHz band and its channel configurations.

Figure 2.3: Spectrum use in the 2.4 GHz and the 5 GHzbands.

the original signal without losing too much quality. In general, IEEE 802.11 divides the ISM band into a series of channels. The bandwidth of each channel is determined by the modulation scheme. Channels are defined by their center frequencies, which begin at 2.400 GHz for channel 0. Successive channels are derived by adding the bandwidth step. For instance, FH defines 1 MHz channels, which centers channel 1 at 2.401 GHz, and so on. DSSS [56] defines 22 MHz channels, having the 2.412 GHz as the center frequency of channel 1. Likewise, Orthogonal Frequency Division Multiplexing (OFDM) [57] defines 20 MHz channels.

Although configuring overlapping frequencies at a single location is feasible and often functional, it may lead to interference, potentially causing slowdowns, especially during periods of high usage. However, specific subsets of frequencies can be concurrently employed at a given location without encountering interference, as illustrated in Figure 2.3a. The need for careful frequency spacing arises from both the inherent bandwidth occupancy dictated by the protocol and the attenuation of interfering signals over distance.

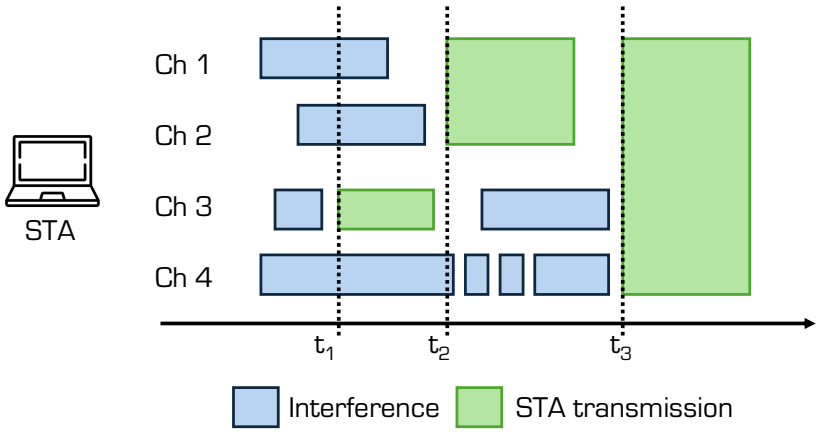


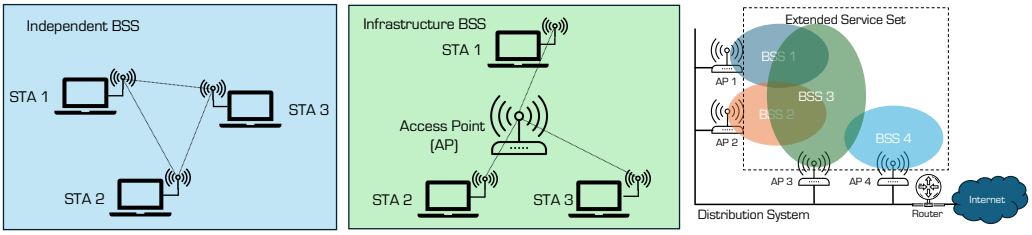
Figure 2.4: Channel selection applying DCB.

2.1.1 Channel Bonding

More advanced IEEE 802.11 standards, such as 802.11n/ac/ax [58], use OFDM and the lower part of the 5 GHz band (i.e., the U-NII-1 and U-NII-2 bands), where there is a total of eight non-overlapping channels of 20 MHz bandwidth as shown in Figure 2.3b. Additionally, aiming to improve the throughput of Wi-Fi transmissions, Channel Bonding (CB) is a technique introduced in 802.11n that enhances the channel capacity by bonding a maximum of two primary channels in a given transmission, augmenting the available bandwidth. In CB, multiple adjacent channels can be bonded to create a higher bandwidth channel to increase the throughput of a given transmission potentially. The total bandwidth of CB has grown from 40 MHz (802.11n) to 160 MHz (802.11ac/ax) and it is expected to support up to 320 MHz (802.11be, using the 6 GHz band).

Under the constraint to bond only adjacent channels, the number of available configurations in a small portion of the spectrum is 15 (8×20 MHz, 4×40 MHz, 2×80 MHz and 1×160 MHz). With the introduction of novel bonding techniques using OFDM, non-contiguous channels can be bonded, increasing the number of available configurations. Several wireless transmission policies can be applied, including static, dynamic, or probabilistic. In Static Channel Bonding (SCB), the same subset of channels is selected in a per-packet basis transmission. In Dynamic Channel Bonding (DCB), a variable subset is chosen according to the available spectrum, as shown in Figure 2.4. Finally, in probabilistic CB, a configuration is chosen with a given probability [61].

In Figure 2.4, we assume a transmitter in which DCB can be applied over channels 1–4, using configuration C4 or C14 from Figure 2.3b. At time t_1, the Station (STA) senses that only channel 3 is available for transmission, using the Carrier-sense Multiple Access with Collision Avoidance (CSMA/CA) procedure (explained in the following subsection). Similarly, at t_2, channels 1 and 2 are sensed to be free at the moment of transmission, and the two are bonded consequently. Finally, at t_3, the STA can transmit over the bonded channels 1–4. Note that if a SCB policy is applied, the STA can transmit only at t_3, while if a probabilistic policy is used, any combination of channels can be selected in t_2 and t_3.



(a) Independent BSSs. (b) Infrastructure BSSs. (c) Extended Service Set

Figure 2.5: Architectural blocks and topologies of IEEE 802.11 networks. Dashed lines represent the wireless transmissions.

2.1.2 Interference Management

At the architectural level, the IEEE 802.11 standard defines several basic components as follows. A STA is any transmitting device that contains an interface that is compliant with the IEEE 802.11 MAC and PHY layer. An Access Point (AP) is any device that provides access to distribution services, i.e., the Internet, over the air. Then, a Basic Service Set (BSS) comprises a cluster of STAs engaged in communication. Interactions occur within a defined area, shaped by the wireless medium's propagation traits. When within this area, a STA can engage in communication with other BSS' STAs. There are two variants of a BSS. In *independent* BSSs STAs communicate with each other directly without the need for an intermediary, as shown in Figure 2.5a. That means that the STAs in the *independent* BSSs must be within each other's range of transmission. Conversely, the *infrastructure* BSSs requires the use of an AP, as shown in Figure 2.5b.

BSSs can interconnect through a distribution system, forming an Extended Service Set (ESS), as shown in Figure 2.5c. ESSs are formed to offer continuous coverage to the STAs connected to the different APs, provided that all the APs are configured to belong to the same ESS and there is an overlap between the coverage area of the individual BSSs. However, since the medium is shared by all wireless devices, the transmissions should be coordinated. The standard defines this coordination mechanism as independent because there is no central entity to enforce this coordination and the STA are free to try to access the medium. This mechanism is known as the CSMA/CA.

With CSMA/CA, wireless devices sense the medium in search of concurrent transmissions. If the channel is idle, Wi-Fi nodes randomly choose a timer value (back-off) within a range (the contention window) and count down until the timer expires. If the channel is still idle, then the transmission takes place. If the channel is occupied or a collision is detected, Wi-Fi doubles the size of the contention window to minimize the probability of consecutive collisions in the next transmission. When the contention window reaches its maximum size, the transmission is dropped.

Collisions are expected as more STAs are simultaneously transmitting. Assuming all the nodes are within others' carrier-sensing range causes well-known problems, such as hidden or exposed nodes. The hidden node problem refers to a scenario where two or more STAs belong to the same BSS but cannot sense each other due to being outside each other's transmission range. Thus, both STAs are "hidden" from each other. A collision is generated if both STAs simultaneously transmit data to the AP. The Request-

to-Send (RTS) and the Clear-to-Send (CTS) scheme is introduced to avoid this. Besides the CSMA/CA, a transmitting STA should send a RTS frame to the destination STA. Upon receiving this frame, the receiving STA sends a CTS frame, in which case, the transmitting STA proceeds with the transmission. In the exposed node problem, a STA defers its transmission due to the sensing of ongoing transmissions. However, the STA' intended receiver is actually located beyond the range of the ongoing transmissions and is therefore not susceptible to interference. Consequently, the STA unnecessarily defers its transmission, leading to potential inefficiency in network utilization.

2.2 Programmable Networks

The content of this section is partially based on information that can be found in the following references:

[62] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, 2014

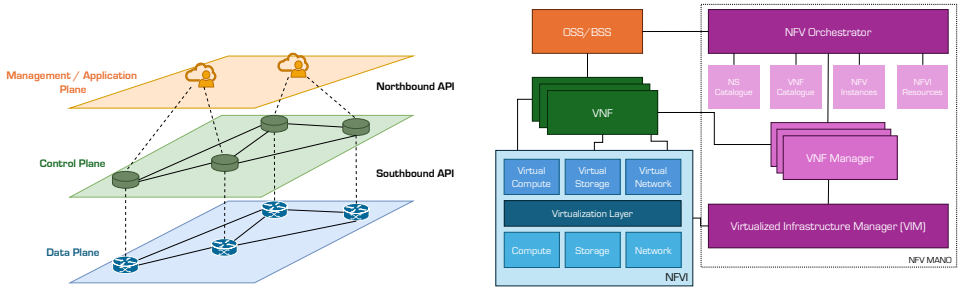
[63] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," IEEE Communications surveys & tutorials, vol. 18, no. 1, pp. 236–262, 2015.

[64] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration," ETSI, Specification, 2014. [Online]. Available: `https://www.etsi.org`

This section introduces the domain of programmable networks, a paradigm shift in network architecture that enables dynamic and flexible configuration of network functions and services to meet evolving application requirements. This subsection explores the fundamental principles, methodologies, and emerging technologies driving the development of programmable networks. As with the preceding section, our objective is not to conduct an exhaustive analysis of programmable networks but rather to offer an introductory overview of the themes that will be further explored in subsequent technical chapters.

With the growing complexity of traditional networks, their management also becomes increasingly challenging [65]. This increased difficulty arises from the diverse array of network devices and manufacturers available in the market. Consequently, network operators must manually configure each device separately using vendor-specific commands whenever a fault occurs, or a configuration change is necessary. SDN [66] emerges as a novel paradigm aiming to address these challenges. By decoupling the network's control logic from the underlying routers and switches, SDN facilitates network control's (logical) centralization and introduces programmability into network management.

Decoupling the control plane (that decides how to handle network traffic) and the data plane (that forwards traffic according to the decisions made by the control plane) increases flexibility and fosters innovation and evolution of the network infrastructure we are experiencing today. The separation of the control and data planes can be realized by employing a well-defined Application Programming Interface (API) between the



(a) Basic SDN components. APIs are represented as dashed lines.

(b) NFV-Management and Orchestration (MANO) architectural framework [64].

Figure 2.6: SDN and NFV, their similarities and differences.

switches and the SDN controller, called the southbound API. Therefore, the control functionality is removed from the network devices, which become simple packet-forwarding elements. The controller exercises direct control over the state in the data plane elements via this API, as depicted in Figure 2.6a. The most notable example of such an API is OpenFlow [67].

More importantly, SDN enables the control plane to be implemented as an external entity that can run on Commercial Off-The-Shelf (COTS) hardware and commodity servers. This setup provides the necessary resources and abstractions to simplify the programming of network devices. Additionally, this abstraction layer allows for the easier development of advanced networking functions and applications, as the control plane functions similarly to an operating system by providing an abstraction layer for the underlying hardware.

Following the trend of decoupling software (control logic) from hardware (forwarding elements), NFV [68] emerged from a telecom alliance to extend the softwarization trend beyond switches and routers. Traditionally, Network Functions (NFs) are tightly coupled with their underlying infrastructure. NFV decouples these software implementations from the computing, storage, and networking resources they use, insulating NFs through a virtualization layer. This approach allows for consolidating various network equipment types onto high-volume servers, switches, and storage devices, which can be located in data centers, distributed network nodes, or end-user premises.

This decoupling introduces Virtual Network Functions (VNFs) and redefines their relationships with the Network Function Virtualization Infrastructure (NFVI). Consequently, a specific service can be decomposed into VNFs implemented as software on industry-standard physical servers. These VNFs can be dynamically relocated and instantiated at different network locations, facilitating service introduction in targeted geographical areas without the need for new hardware. VNFs can also be chained with other VNFs and/or Physical Network Functions (PNFs) to create a comprehensive Network Service (NS).

2.2.1 Management and Orchestration Operations

Unfortunately, managing NSs within NFV architectures necessitates a new set of MANO functions to effectively handle the setup of virtual nodes and links, their placement, and resource orchestration. The deployment, execution, and operation of VNFs is driven by standardized procedures created by European Telecommunications Standards Institute (ETSI) [64], whose behavior is guided by high-level policies or intents [69] that describes the characteristics of the NSs and their constituent VNFs. The MANO system includes an Network Function Virtualization Orchestrator (NFVO) responsible for the lifecycle management of NSs, a set of Virtual Network Function Managers (VNFMs) overseeing the lifecycle of individual VNFs, and a Virtual Infrastructure Manager (VIM), as shown in Figure 2.6b. Beyond defining such procedures, ETSI-NFV [64] also describes the roles and responsibilities of each functional block of its architectural framework, the reference point between the NFV-MANO functional blocks and other functional blocks, e.g., between the VIM and the NFVI.

Concerning NSs and their composing VNFs, the NFV-MANO framework introduces several operations that extend beyond the traditional Fault, Configuration, Accounting, Performance, and Security Management (FCAPS) model [70]. The decoupling of NFs from the physical infrastructure requires new management functions centered on the creation and lifecycle management of the required virtualized resources for the VNF, collectively known as VNF management. In particular, VNF and NS's lifecycle management includes operations such as onboarding, instantiation, updating, and termination.

Onboarding NSs includes registering the NS in a catalog. The deployment and the operational behavior of each NS is encapsulated in a deployment template. This template includes the composing VNFs, the attributes, and requirements necessary for the NS to operate. Each VNF has a similar template. Then, the NFVO coordinates the lifecycle of the NSs, while the VNFMs perform the lifecycle of the VNFs according to such templates.

A second step includes the instantiation of the VNFs composing the NS. In this case, the NFVI reserves the resources (computation, storage, and networking) necessary for the operation of the NS, considering specific requirements, constraints, and policies that have been pre-provisioned or accompany the instantiation request. During the NS lifecycle, the NFVO may monitor performance Key Performance Indicators (KPIs) to support explicit requests from other functions, such as updating or upgrading a VNF whose behavior is affecting the network performance, or scaling.

Scaling operations are needed to change the configuration of the virtualized resources used by NSs or VNFs. For instance, scaling operations can be used to resize NSs due to a surge of network traffic. Scaling may include the addition of new virtualized resources, i.e., scale up by adding more virtualized Central Processing Unit (vCPU) power or scale out by adding a new instance of the VNF, or the release of virtualized resources, i.e., scale down by removing Central Processing Unit (CPU) power or scale in by removing an instance of the VNF.

The NFVO performs its tasks by utilizing the VNFM services and orchestrating the NFVI, which facilitates the interconnection between VNFs. Its functions are exposed as services to other functions in an open and well-known abstracted manner. To fulfill its responsibilities, the NFVO functions consume services provided by other functions, such

as the VIM. Finally, once the NS fulfills its purpose, the NFVO should ensure that the virtualized resources at use by the NS are released in the termination phase.

2.3 Machine Learning

The content of this section is partially based on information that can be found in the following references:

[71] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," Science, vol. 349, no. 6245, pp. 255–260, 2015.

[72] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT Press, 2018.

[73] C. M. Bishop, Pattern Recognition and Machine Learning. Springer, 2006.

[74] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," IEEE Transactions on Neural Networks, vol. 20, no. 1, pp. 61–80, 2008.

Machine Learning (ML) is a subset of Artificial Intelligence (AI) and computer science that enables software to perform tasks without explicit programming. By analyzing large quantities of data, ML uncovers hidden patterns and relationships using statistical and optimization methods. ML algorithms typically learn from examples of desired input-output behavior, mimicking human learning processes.

An ML algorithm consists of three key components. The first is a *decision process*, which involves a series of computations to identify hidden patterns in the input data. The second is an *error function*, which measures the deviation between the algorithm's estimations and the actual values, thereby assessing the model's performance. The third component is a *model optimization process*, which adjusts the decision process to minimize the error.

The learning process in ML involves an iterative evaluation and optimization cycle. Input data is fed into the decision process, the error is calculated, and the decision process is updated based on this error. This cycle is repeated with subsequent data batches until a predefined performance threshold or level is achieved.

ML methods can be subclassified based on the learning paradigm employed. Broadly, there are three primary types: Supervised Learning (SL), Unsupervised Learning (UL), and Reinforcement Learning (RL). Each paradigm corresponds to different types of tasks that the machine aims to learn.

In Supervised Learning (SL), the algorithm learns from labeled examples. The labels are used to indicate the degree of error based on a distance metric (e.g., mean squared error). Common applications of SL include classification, where outputs are discrete variables, and regression, where outputs are continuous variables.

In Unsupervised Learning (UL), labeled data is not available. This technique is primarily used to identify patterns within the data that are not immediately apparent to humans.

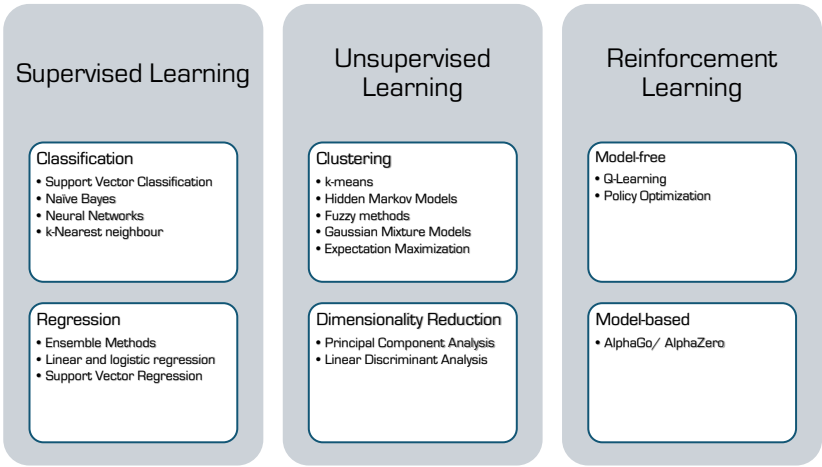


Figure 2.7: General classification of Machine Learning approaches.

A typical application of SL is clustering, where the algorithm groups data points with similar characteristics by performing density estimation. Clusters are defined by small distances among data points within a subgroup, whereas large distances between subgroups indicate distinct clusters.

Reinforcement Learning (RL) differs in that the machine learns through interaction with its environment. The machine performs actions that result in reactions from the environment, which serve as feedback to quantify the actions' effectiveness. RL involves feedback from the environment, distinguishing it from UL, and it does not rely on labeled examples, distinguishing it from SL.

Figure 2.7 presents a non-comprehensive classification of various ML techniques. SL techniques include classification and regression, with Neural Networks (NNs) being particularly prominent. UL techniques are used for clustering data and reducing dimensionality (e.g., representing a 1000-feature data point with a smaller subset of features). RL techniques are categorized into model-free and model-based. In model-free RL, the machine does not have a model of the environment, whereas in model-based RL, the environment is represented by a model (e.g., the rules in the game of Go [75]).

Building on the introductory overview, next subsections provide more details regarding these three types of ML algorithms, making especial emphasis on the methods used during the development of this dissertation. In particular, subsection 2.3.1 will examine algorithms, and applications of SL; subsection 2.3.2 will focus on UL, while subsection 2.3.3 will focus on RL. These subsections aim to provide comprehensive insights, to enhance understanding and facilitate further discussion.

2.3.1 Supervised Learning

Data forms the cornerstone of any ML method, serving as the basis for building ML models. Mathematically, data is represented as $X = \{x_1, x_2, \ldots, x_N\}$, where $x_i \in X$ for all $i \in [N] := \{1, \ldots, N\}$. Each x_i represents a data point or example, which may be

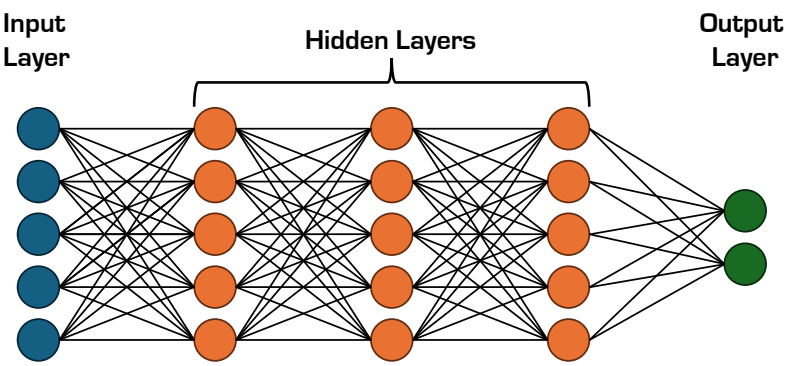


Figure 2.8: A DNN model

structured (e.g., databases, tables) or unstructured (e.g., sensor data streams, audio, text). A fundamental assumption of ML methods is that these data points are Independently and Identically Distributed (i.i.d) from a common distribution on X.

In SL, each data point x_i is paired with a corresponding label y_i. The objective is to find a mapping $f(x)$ from X to Y, given a training set of pairs (x_i, y_i). When $Y = \mathbb{R}$ or $Y = \mathbb{R}^d$, indicating that the labels are continuous, the task is referred to as *regression*. Conversely, when y takes values from a finite set of discrete labels, the task is known as *classification*.

The integrity of the learning process hinges on the quality and representativeness of the data. Ensuring that data points are i.i.d. is crucial for the generalization capabilities of the model. In real-world applications, data preprocessing steps such as normalization, handling missing values, and feature extraction are often necessary to prepare the data for effective model training.

Various forms of the mapping function f exist, including the algorithms illustrated in Figure 2.7. Among the most widely utilized are NNs, also known as Artificial Neural Networks (ANNs). NNs are ML algorithms designed to emulate the human brain's information processing to understand and intelligently classify data. They consist of multiple node layers, including an input layer, one or more hidden layers, and an output layer, as shown in Figure 2.8. Each node, or artificial neuron, is interconnected and has an associated weight and threshold. When the output of a node exceeds the specified threshold, the node is activated and transmits data to the subsequent layer. Conversely, if the output does not meet the threshold, the node does not forward any data.

Mathematically, ANNs can be regarded as function approximators, denoted as f^*, of the actual function $f(x)$. The term "network" in ANNs refers to the interconnected functions that can be represented as a directed acyclic graph. For example, a three-layer ANN can be described by three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$, which are sequentially connected to form $f^* = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Here, $f^{(1)}$ is known as the first layer of the network, and this sequential composition continues through the subsequent layers.

Depending on the number of layers, ANNs can be Deep Neural Networks (DNNs). The number of layers that are part of such composition gives the depth of the model, from which the term "deep" in DNNs comes. A NN with more than three layers, including the input and output layers, qualifies as a Deep Learning (DL) algorithm or DNN. In

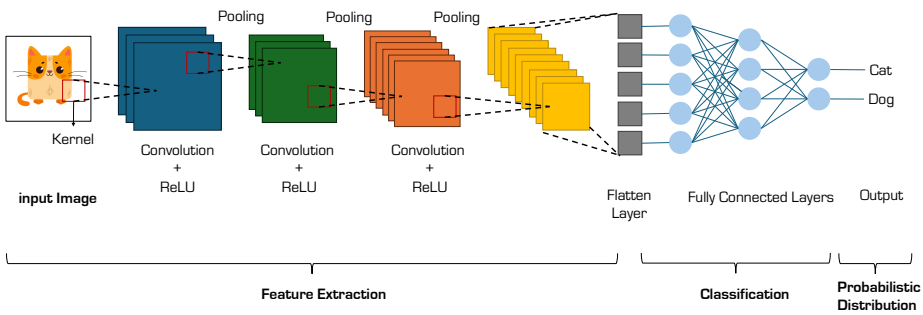


Figure 2.9: A CNN model

contrast, a NN with only three layers is considered a shallow NN. DNN and NNs have significantly advanced fields such as computer vision, natural language processing, and speech recognition. The subsequent sections will discuss two of the most prominent NNs used in networking.

2.3.1.1 Convolutional Neural Networks

A specialized type of DNN is the Convolutional Neural Network (CNN), which is particularly well-suited for computer vision tasks. CNNs are designed to analyze and process data with a grid-like topology, such as images. They excel in image recognition, object detection, and pattern recognition due to their ability to automatically and adaptively learn spatial hierarchies of features from input images, thus providing automatic feature extraction. The architecture of a CNN, illustrated in Figure 2.9, comprises an input layer that receives raw data, convolutional and activation layers that extract features, pooling layers that down-sample feature maps to reduce dimensionality, fully connected layers that utilize the extracted features to make predictions, and an output layer that produces the final output, such as class probabilities in classification tasks.

The primary component of CNNs is the convolutional layer, which applies a convolution operation to the input and passes the result to the next layer. These layers use filters, or kernels, that slide over the input data to capture spatial features such as edges, textures, and patterns. Each filter detects specific features within the image.

Pooling layers are another essential component of CNNs. These layers reduce the input volume's spatial dimensions (width and height), making the computation more efficient and decreasing the number of parameters. Common pooling operations include max pooling and average pooling. For instance, max pooling selects the maximum value from each sub-region of the feature map. Following the convolution operation, an activation function such as REctified Linear Unit (ReLU) is applied to introduce non-linearity into the model, allowing it to learn more complex patterns.

Before the classification stage, a flattening layer is typically used. This step converts the multi-dimensional output of the convolutional and pooling layers into a one-dimensional vector, which is then fed into the fully connected layers. The fully connected layers resemble traditional NNs, where each neuron connects to the previous layer's neurons. These

layers are generally positioned at the network's end to make predictions or classifications based on the features extracted by the convolutional and pooling layers. A softmax layer is often employed in classification tasks to convert the output into probability distributions over different classes.

2.3.1.2 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of NNs specifically designed to handle data that is structured as graphs. Unlike traditional NNs operating on fixed-size and grid-like structures such as images or sequences, GNNs are adept at processing graph-structured data where the number of nodes and their connectivity patterns can vary. Graphs are mathematical representations consisting of nodes (or vertices) and edges that define relationships between pairs of nodes. This data structure is prevalent in various domains, including social networks, biological networks, communication networks, and knowledge graphs.

Mathematically, a graph is defined as a 3-tuple $G = (V, E, U)$ with V representing a set of nodes with cardinality N, E representing the set of edges with cardinality L, and a set U representing the global attributes of the graph. Each node has an associated set of features, represented by a vector of $1 \times D$, where D is the number of input features per node. Similarly, each edge represents the interactions among nodes; their associated features are represented by a vector of the form $1 \times R$ where R is the number of input features per edge. The graph's topology is represented by a binary adjacency matrix A of $N \times N$, where a 1 represents an edge between nodes.

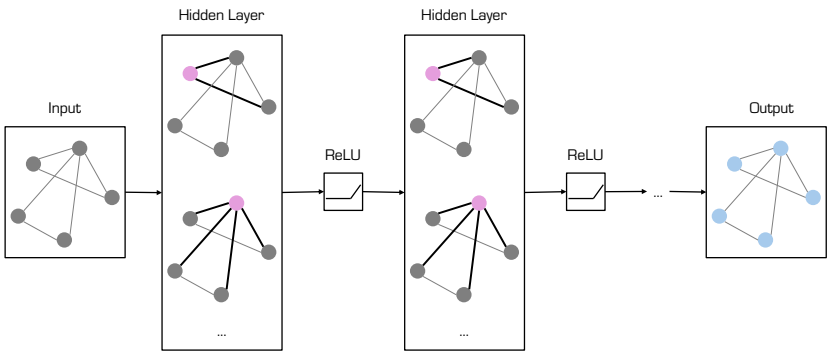
A GNN model, shown in Figure 2.10a employs a "graph in, graph out" architecture consisting of one or more graph layers. Each GNN layer, illustrated in Figure 2.10b, operates on the components of the graph—nodes, edges, or global attributes— utilizing NNs to learn new representations of these components, a process known as embedding. The output of a GNN layer maintains the same graph structure but updates the embeddings for nodes, edges, and global attributes, thereby enhancing their representations.

A GNN layer updates nodes' states by exchanging neighborhood information until a stable equilibrium is reached. The update for node v at layer k hidden state, h, can be expressed as:

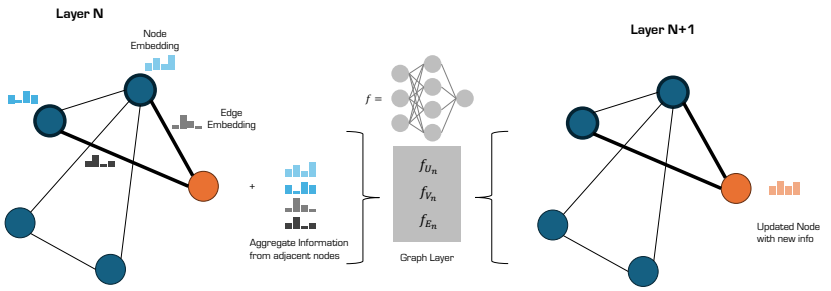
$$h_v^{(k)} = \text{UPDATE}\left(h_v^{(k-1)}, \text{AGGREGATE}\left(\{h_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\right)\right)$$

where $h_v^{(k)}$ is the feature vector of node v at layer k, $\mathcal{N}(v)$ represents the set of neighbors of the node v, and UPDATE and AGGREGATE are functions that define the update and aggregation mechanisms, respectively.

The key idea behind GNNs is to iteratively aggregate and transform node features by incorporating information from their neighbors. This process is known as message passing or neighborhood aggregation. For each node, messages are generated based on the node's features and the features of its neighbors. These messages are then propagated across the graph using different propagation modules. The most common propagation modules are the convolution and the recurrent operator. Then, the messages received by each node are aggregated using a permutation-invariant function such as sum, mean,



(a) A GNN Model.



(b) A GNN Layer.

Figure 2.10: A GNN model is composed of several GNN layers. Using a graph as an input, each component (V, E, U) is updated by a fully connected layer to produce a new graph. Each function subscript indicates a separate function for a different graph attribute at the n-th layer of a GNN model.

or max. This ensures that the node's representation is updated based on the aggregated information from its neighbors and does not alter the graph's topology. Thirdly, the aggregated message updates the node's features through a transformation, often parameterized by a NN layer. After several rounds of message passing and aggregation, the node features are combined to form a graph-level representation, such as graph classification, if the task requires it.

2.3.2 Unsupervised Learning

Similar to SL, UL operates under the assumption of i.i.d data points represented by $X = \{x_1, x_2, \ldots, x_N\}$, where $x_i \in X$ for all $i \in [N] := \{1, \ldots, N\}$. However, unlike SL, the primary objective of UL is to discern valuable properties of the underlying structure of X without utilizing labeled data. These algorithms aim to uncover latent patterns or groupings within the data.

The capability of UL algorithms to unveil similarities and disparities within information renders them particularly suited for various applications, including exploratory data analysis, cross-selling strategies, customer segmentation, and image and pattern recognition. Moreover, UL techniques are instrumental in reducing the dimensionality of data, facilitating more efficient model representation. Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) are commonly employed methodologies for dimensionality reduction. Additionally, UL encompasses a spectrum of algorithms beyond PCA and SVD, including k-means clustering, and probabilistic clustering methods, each tailored to address specific data analysis challenges.

2.3.3 Reinforcement Learning

Reinforcement Learning (RL) is a type of ML algorithm where an agent learns to interact with an environment by taking actions and receiving feedback in the form of rewards or penalties, as shown in Figure 2.11. The goal of RL is for the agent to learn a policy, which is a mapping from states to actions, that maximizes the cumulative reward over time.

RL has emerged as a powerful and versatile approach for solving Markov Decision Processs (MDPs), offering a robust framework to tackle a wide range of complex decision-making problems. RL's appeal lies in its ability to address MDPs by exploring and analyzing the effects of different actions in an environment, making it well-suited for dynamic and uncertain scenarios.

An MDP is a discrete-time stochastic framework that models decision-making problems. In general terms, an MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces; $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition kernel, with $p(s'|s, a)$ denoting the probability of transitioning to state s' from s after action a is taken; $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathrm{R}$ is the immediate reward obtained for performing action a, following a predetermined Reward Function (RFn). The policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, is a mapping function from the state space to the space of probability distributions over the actions, with $\pi(a|s)$ denoting the probability of selecting action a in state s. The solution to an MDP is an optimal policy that maximizes the expected long-term reward, which is a discounted sum of immediate

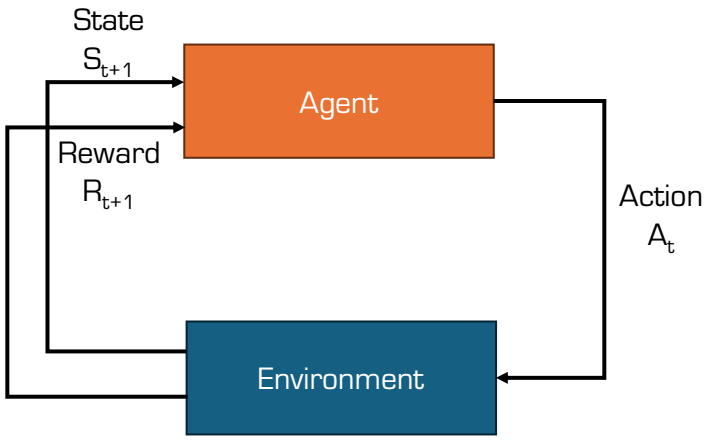


Figure 2.11: Basic composition of an RL problem.

rewards. This optimal policy mostly uses iterative solution methods, e.g., value or policy iteration.

RL's capacity to adapt to various environments and its capability to navigate the intricate trade-offs between exploration and exploitation makes it a compelling choice for finding optimal policies within the context of MDPs. This adaptability, coupled with its potential for achieving near-optimal solutions over time, underscores why RL stands as a prominent and effective tool for solving MDPs [72].

The most well-known RL algorithm is Q-Learning. In this algorithm, the agent tries to maximize the state-action value function, known as the Q-function, defined in Equation 2.1. The discount factor $\gamma \in [0, 1)$ is used to bound the sum, which otherwise might grow unbounded. The Q-function represents the reward accumulated by the agent in the long run for taking action a in state s and following the policy π. To obtain the optimal Q-function, $Q^*(s, a)$, Equation 2.1 can be written as a Bellman optimally equation as in Equation 2.2.

$$Q^{\pi}(s, a) = E\Big\{ \sum_{t=0}^{\infty} \gamma R_t \Big| s = s_0, a = a_0, \pi \Big\} \tag{2.1}$$

$$\begin{aligned} Q^*(s, a) = \sum_{s \in S} P(s, a, s') \big[R(s, a, s') + \\ \gamma \max_{a'} Q^*(s', a') \big] \end{aligned} \tag{2.2}$$

Intuitively, the Bellman optimality equation allows expressing the expected total reward for an agent by taking a in s in terms of the optimal value from the next state. After computing Q^* for every pair $(s, a) \in S \times A$, the optimal policy, also known as greedy, can be computed as $\pi^*(s) = arg\max_{a'} Q^*(s', a)$.

When the state and action spaces derived from the MDP are sufficiently small, Q-Learning can store the value functions as arrays or tables [72]. In this case, the approaches can find exact solutions; that is, they can find the optimal value function and the optimal policy if they visit each state-action pair an infinite number of times [76]. However, when the data is real-valued, tabular Q-Learning lacks scalability as the size of the tables is unpractically large.

On the contrary, Deep Reinforcement Learning (DRL) is preferred for complex, high-dimensional, or continuous state and action spaces. Under DRL, a DNN approximates the value function, policy, or both. These NNs provide the capacity to handle high-dimensional and uncountable infinite-state spaces. DRL algorithms can be classified as on-policy or off-policy. Off-policy algorithms update their policy using data generated by a different policy. This means that the agent can learn from past experiences and use data collected by any previous policy, making them more sample-efficient but potentially introducing higher variance in the learning process. On the other hand, on-policy algorithms update their policy while using the data generated by the same policy. This means that the data collected during the learning process is directly used to improve the current policy, leading to more stable but potentially less sample-efficient performance. As a result, the policy being learned is constantly changing as the agent explores the environment and gathers new experiences. Other classifications of DRL techniques are possible. We refer the reader to [77] for a detailed view of the state-of-the-art of this kind of algorithm.

Chapter 3

Designing the Network Intelligence Stratum for 6G Networks

The content of this chapter has been partially published in:

[78] **P. Soto**, M. Camelo, G. Garcia-Aviles, E. Municio, M. Gramaglia, E. Kosmatos, N. Slamnik-Kriještorac, D. De Vleeschauwer, A. Bazco-Nogueras, L. Fuentes et al., "Designing the Network Intelligence Stratum for 6G Networks," Computer Networks, vol. 254, p. 110780, 2024. [Online]. Available: https://doi.org/10.1016/j.comnet.2024.110780. [IF 5.6]

In response to the growing complexities of modern and future networks, a broad spectrum of Artificial Intelligence (AI)/Machine Learning (ML) models are being crafted. These models are crucial for enabling automated decision-making, predictive analytics, proactive network management, security enhancements, and network performance optimization, laying the groundwork for what is termed Network Intelligence (NI). Leading Standard-Defining Organizations (SDOs) are embedding NI into upcoming network architectural designs, with a focus on the closed-loop methodology. Yet, seamlessly incorporating NI into network frameworks remains a challenge. This chapter presents the tools for defining and orchestrating NI. Together, they unveil a comprehensive architectural blueprint and supporting procedures to realize a Network Intelligence Stratum (NI Stratum) for 6G networks. The main component of this *Stratum* is an innovative Network Intelligence Orchestrator (NIO) to facilitate closed-loop NI operations across diverse network domains. This approach aims to refine NI deployment and management, addressing scalability, conflict resolution, and data management challenges, and underscores the integration and validation challenges of NI in real-world settings. Specifically, this chapter aims to solve the following Research Questions (RQs):

[RQ_1] **How can Network Intelligent Functions (NIFs) be designed and decomposed to facilitate the coordination and reutilization of their components among multiple NIFs to achieve End-to-End (E2E) convergence in network operations?**

[RQ_2] **What frameworks and protocols are necessary to support the seamless and interoperable deployment of AI/ML models across diverse network environments, ensuring effective integration and cooperation among various NIFs to achieve global optimality in zero-touch network management processes?**

3.1 Context and Problem Statement

To meet the stringent demands imposed by upcoming services such as multisensory eXtended Reality (XR) applications and connected robotics [6], which will rely on performance metrics such as virtually unlimited capacity and perceived zero latency, 6G networks will necessitate advanced algorithms. These algorithms will be executed by diverse controllers and orchestrators, leveraging AI and ML techniques, managing different micro-domains within the network. They will oversee the intricate assembly of Network Functions (NFs) and associated resources, forming a mosaic to support various Network Services (NSs), such as network slices, utilized by different tenants. This orchestration contributes to the development of advanced networking algorithms, referred to as NI. More specifically, an instance of NI is described as a pipeline of supple yet effective algorithms that rapidly identify or predict new requests or fluctuations in network activities [79], mainly based on learning methods. Subsequently, these algorithms respond by automatically instantiating, relocating, or re-configuring Virtual Network Functions (VNFs).

Hence, the effectiveness and sustainability of 6G systems will heavily rely on the seamless integration of NI solutions into the network infrastructure. Within the network framework, each controller and orchestrator is expected to execute multiple instances of NI, aligning with various Key Performance Indicator (KPI) objectives. These objectives encompass guarantees related to Quality of Service (QoS) and Quality of Experience (QoE), infrastructure and resource utilization optimization across diverse tenants or network services, and the realization of end-to-end network automation for achieving zero-touch network and service management. Consequently, the architectural blueprint of mobile networks requires a comprehensive reconsideration, ensuring that the operations of multiple NI instances can be seamlessly accommodated across all micro-domains through fully automated processes [19].

Ongoing initiatives led by major SDOs to incorporate NI into next-generation network architectures are mostly based on the concept of "closed-loop AI" [80, 19, 14]. Under this paradigm, NI instances deployed at orchestrators and controllers function within closed control loops: they capture the context of management decisions, gather feedback on decision quality through continuous monitoring, and use it to enhance future decision-making. The closed-loop model enables NI to comprehend the significance of specific factors in a given situation and progressively automate decision-making in alignment with targeted KPIs.

However, existing frameworks for network management established by prominent SDOs such as 3rd Generation Partnership Project (3GPP) and European Telecommunications Standards Institute (ETSI), along with global industrial initiatives such as Open Radio Access Network (O-RAN), currently fall short in facilitating the seamless integration of closed-loop NI. An example of this is seen in the Fifth Generation (5G) Network Data Analytics Function (NWDAF) proposed by 3GPP [81], which collects data from any 5G core function to provide analytics information to other core functions or the network Operations, Administration, and Maintenance (OAM). Despite being a pioneer in its field, the NWDAF opens the door to standardized integration of AI/ML into modern networks but does not consider how these intelligent models are created or maintained. This hinders the practical adoption of NI within Sixth Generation (6G) networks and calls for original enhancements to the network architectural paradigm. In particular, it

is fundamental that network architectures become AI-native, facilitating the integration of the NI approaches by considering their full lifecycle, from creation to termination. This AI-native architecture will support automation capabilities in future generations of communication systems. Notice that an AI-native architecture is a requisite for an autonomous network since the AI-native architecture will enable the necessary processes to ensure the correct operation of the NI, guaranteeing stability and reliability in the network.

This chapter presents a complete architectural design of a NI Stratum, including the NIO and the tools to define and design NI in a unified manner. Compared to state-of-the-art works (see Table 3.1), the proposed NI Stratum tackles simultaneously multiple challenges of NI such as how to properly define NI, how to manage their lifecycle, and how to coordinate them via well-defined procedures. Additionally, we define a set of internal interfaces to facilitate the interaction among functionalities in the NI Stratum. Such interfaces represent an adaptation or an extension of current standardized interfaces proposed by major SDOs, e.g., ETSI Network Function Virtualization (NFV) Management and Orchestration (MANO) [64] Moreover, we analyze how the external interfaces may be realized when interacting with the Radio Access Network (RAN) and Core network segments. The external interfaces provide a means to increase integration and interoperability with current systems. Finally, we provide a detailed description of several inter- and intra-NIO procedures that are required to realize the NI Stratum, providing the required workflows and sequence diagrams.

The remainder of this chapter is structured as follows. Section 3.2 reviews and summarizes the research and standardization efforts in defining an AI-native architecture. Section 3.3 shows the complete design of the NI Stratum, including our methodology for defining NI and detailing the necessary functional blocks to orchestrate NI in an end-to-end way. Moreover, Section 3.4 presents internal and external interfaces needed to perform such orchestration. Using the functional blocks and the interfaces defined in the previous sections, Section 3.5 shows their interactions to perform the most important orchestration tasks (e.g., creation, management, and termination), including the identified challenges regarding knowledge sharing and conflict resolution. Finally, conclusions are presented in Section 3.6.

3.2 Research and Standardization Efforts Towards the Integration of NI in Mobile Networks

The vision of future 6G networks largely builds on conflict-free and synergic operation among various NI algorithms across network schedulers, controllers, and orchestrators [79, 82]. This section reviews current efforts from SDOs, industry and academia that aim to facilitate joint and end-to-end NI operation.

Two major stakeholders have proposed architectures for autonomous networks. On the one side, the International Telecommunication Union (ITU) in its Y.3172 recommendation [14] provides an architectural framework for integrating AI into networks. It outlines architectural requirements and components, including an ML pipeline and a ML management and orchestration functionalities. Their architecture proposes a sandbox, which is defined as an isolated domain that host separate ML pipelines to train, test and evaluate

NI before they are deployed in a production network. This work is complemented by the recommendation Y.3061 recommendation [16] that defines even a broader architecture based on controllers. A controller is defined as a workflow, closed loop or open loop of a system under control. To enable autonomy, the architecture proposes an exploratory evolution subsystem, which enables the creation and adaptation of controllers in response to changes in the underlay network. On the other side, the TMForum in its IG1230 document [83] proposes a technical architecture, including scenario realizations, and industry standards. The architecture is divided in three layers: business, service, and resource operations. While business are in charge of handling the interactions with costumers and business units. This layer exposes a set of business services based on the capabilities of a set of objects over which the autonomous domain exercises governance. The service operation is responsible for planning, construction, maintenance, and optimization, while the resource operation layer is responsible for the translation of upper-layer services and application intents into network behaviors, continuously ensures the Service Level Agreement (SLA) commitment of network connections or functions and implements autonomous control loop management of single-domain networks.

Current network management frameworks proposed by major SDOs present deficiencies when integrating and managing NI approaches. Our examination, detailed in [4] and in Section 5 and Appendix B of [84], reveals that standards and platforms from entities like ETSI, O-RAN, and 3GPP, including implementations such as Open Source Management and Orchestration (OSM) or Open Network Automation Platform (ONAP), lack mechanisms for coordinating intelligence across different network micro-domains and providing solutions for decentralized and unified data management across NI instances. Moreover, these frameworks show minimal support for managing the NI lifecycle (e.g., O-RAN, 3GPP NWDAF) and only early consideration for methodologies defining and representing NI models (e.g., ETSI-Experiential Networked Intelligence (ENI)). Table 3.1 summarizes the existing frameworks' functionalities for NI management in end-to-end control and orchestration of networks.

Noticing that current network management frameworks proposed by major SDOs are not yet AI-native, researchers have delved into various aspects, including the development of NI algorithms, the orchestration of NI instances, and the coordination of NI across diverse network domains. Investigations into the challenges, methodologies, and advancements in integrating NI aim to pave the way for more efficient and scalable network operations. By examining prior works, this chapter positions its contributions within the broader context of ongoing endeavors to enhance the intelligence and adaptability of network architectures through the seamless integration of the so-called Network Intelligence.

Ericsson, a major player in the telecom industry, addresses the escalating use of AI in networks in its whitepaper [85]. Their definition of "AI-native" emphasizes the comprehensive integration of AI, which requires a corresponding data infrastructure in every sub-component of an entity, as opposed to adding AI components to non-AI-based entities. This approach extends across multiple layers and domains, with a crucial requirement being a distributed data infrastructure supporting (re-)training of models. Ericsson also introduces an AI-native maturity model as a tool for assessing a product's position on the AI-native spectrum and planning the evolution of its implementation towards AI-native capabilities. Additionally, they emphasize the importance of Machine Learning Operations (MLOps) functions for comprehensive end-to-end model lifecycle management within an AI-native architecture.

In [86], Li et al. explored native intelligence solutions for 6G networks, delving into the distributed network architecture of native intelligence. It highlights the capabilities of individual intelligent nodes and the importance of collaboration among them. The discussion extends to cross-domain coordination and knowledge sharing and suggests potential solutions, emphasizing the combination of distributed learning and network functionalities. More than proposing an AI-native architecture, the paper lists the requirements that AI-based functionalities pose over the current network architecture and explains how it should evolve towards an AI-native architecture.

Rossi et al. [87] present a vision for future networks wherein AI attains the status of a primary commodity. The foundational principle revolves around “fast and slow" types of AI reasoning, each offering distinct capabilities for processing network data. Similar to the different timescales in data processing, they propose that intelligence should also operate in different timescales. Fast intelligence is used for perception tasks, where the bias in the NI models is not noticeable or can be tolerated. Instead of proposing a closed-loop control, the authors pose the Data, Information, Knowledge, and Wisdom (DIKW) pyramid as the main building blocks of native NI. In such a pyramid, there are two prominent closed loops: one between the data and the information, where the fast intelligence resides, and another between the information and the knowledge, where a more robust yet slower intelligence oversees and controls the fast intelligence instances.

Brito et al. suggest a network architecture for implementing AI-based applications across various network domains [88]. The aim is to prevent the formation of AI silos by providing reusable data and models, ensuring scalable deployments. Their AI-native architecture is composed of the Network and Service Automation Platform (NSAP) and the Connect-Compute Platform (CCP). The NSAP spans multiple network domains and is responsible for optimizing and managing NIFs. On the other hand, the CCP ensures the necessary lifecycle operations for each NIF. Besides delineating the architecture, the authors furnish workflows for the comprehensive management of AI-based applications and demonstrate the feasibility of the architecture through a vehicular use case.

Focusing on the architectural models proposed by the O-RAN Alliance, D’Oro et al. devised *OrchestRAN*, a NI orchestration framework for next-generation systems [89]. Specifically, the authors showed that the intelligence orchestration problem is Non-deterministic Polynomial time (NP)-hard and proposed three complexity reduction techniques. Deployed as a rApp in the non-Real-Time (RT) Radio Access Network Intelligent Controller (RIC), *OrchestRAN* empowers Network Operators (NOs) to define high-level control and inference objectives. *OrchestRAN* determines the optimal set of data-driven algorithms and their execution location, either the cloud or the edge. In this way, the framework can fulfill the intentions specified by NOs, ensuring compliance with desired timing requirements and preventing conflicts between different data-driven algorithms that govern the same parameter set.

Different flagship initiatives across Europe are also working on designing intelligence for next-generation networks. In particular, Hexa-X and Hexa-X-II are highlighted. Hexa-X [90] recognizes the power of AI and ML for managing the complexity of current 5G networks, optimizing multi-objective problems, and translating high-level service descriptions into technical solutions. With the help of virtualized, on-demand AI functionalities like analytics, prediction, and classification offered by an Artificial Intelligence as a Service (AIaaS) framework, 6G networks will be fully automated, and runtime and

lifecycle management will be facilitated. This framework includes many AI functions, each having a distinct purpose in improving network orchestration to handle NI effectively. These functions include the AI model repository, AI training, AI monitoring, and AI agent. Finally, they identified that managing multiple control loops introduces synchronization and conflict resolution challenges. As a result, solving loop interaction-related problems and guaranteeing smooth integration are necessary for the successful deployment of AI. Advanced AI approaches and modular design are two strategies to address these issues.

Continuing the work of Hexa-X, Hexa-X-II explores the AI enablers within a data-driven architecture [91]. As data is the fuel for AI algorithms, the 6G architectural design should define data collection, processing, and communication channels for AI modules. Therefore, Hexa-X-II proposes that robust data management and lifecycle support through Data Operations (DataOps) and MLOps is required for enhancing automation and collaboration. Finally, the AIaaS framework builds on MLOps, DataOps, and intent-based management to provide AI services to different parts of the network and end-users. To fulfill this purpose, newly designed APIs are required to expose AI services and ensure interoperability and efficient data transmission, optimizing resource usage and energy consumption.

National initiatives such as 6G-RIC in Germany and 6G Flagship in Finland agree that the transition from 5G to 6G is enabled by intelligence. On the one hand, 6G-RIC [92] suggests six technical innovation areas that serve as technology drivers for 6G. One of those areas is autonomous convergent networks, which emphasize the convergence of networking, communications, and computing through virtualization and cloudification. Thus, AI/ML facilitates optimizing and managing network, communication, and computing resources, meeting performance targets, and ensuring efficient service delivery. On the other hand, 6G Flagship [93] considers edge intelligence as the missing element in 5G design and provides some design principles and technological enablers toward 6G. Turning away from the network-centric design, their mandate-driven architecture considers an intelligent cooperation plane where distributed agents in each segment and end devices exchange information for autonomous negotiation and deployment of services. It also extends the management, control, and data plane separation to the application level, allowing applications to express communication needs and QoS requirements.

Differences with the related work: Unfortunately, the ongoing endeavors of prominent standardization bodies and academic institutions in formulating an AI-native architecture fall short in several aspects, as summarized in Table 3.1. One key element is how the NI is defined. Most works assume the NI instance as a single block, not a composition of multiple blocks. However, as it will be shown in Section 3.3.1, the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) control model allows the NI to be decomposed into atomic elements that can be deployed across the whole network infrastructure. This approach allows for the unified representation of NI instances independently of their inner models, facilitates the re-use of similar blocks among different Network Intelligent Services (NISs), and helps identify conflicts as demonstrated in the Radio Access Network Virtualization (vRAN) use case presented in [82]. A similar approach is presented in [87], where their DIKW pyramid resembles the MAPE-K but lacks the concept of multiple closed-loop control.

Another important aspect is the NI lifecycle management, where Brito et al. [88] pro-

Table 3.1: Main differences between our work and similar approaches proposed by major SDOs and academia.

Framework / Related Work	Methodology to define NI	Mechanisms to manage the lifecycle of NI	Mechanisms to coordinate NI across different network segments	Decentralized and unified data management for NI instances	Mechanisms to solve conflicts
ITU [16, 14]	Yes	Yes	Yes	Partially	Partially
TMForum [83, 15]	Yes	Partially	Partially	Partially	Partially
ETSI MEC	No	No	No	No	No
ETSI NFV	No	No	No	No	No
ETSI ENI	Yes	No	No	No	No
O-RAN	Yes	Partially	No	No	No
Open Source MANO (OSM)	No	No	No	No	No
3GPP	No	No	No	No	No
ONAP	No	No	No	No	No
Ericssson Whitepaper [85]	No	Partially	Partially	Partially	No
Hexa - X	No	Yes (D5.2)	Partially	No	Yes (D5.2)
Li et al. [86]	No	Partially	Partially	Partially	Partially
Rossi et al. [87]	Partially	No	Partially	Partially	No
Brito et al. [88]	No	Yes	Yes	Yes	No
D'oro et al. – OrchestRAN [89]	No	Partially	Partially	No	Yes
Network Intelligence Stratum (This Work)	**Yes [79, 4]**	**Yes (This work)**	**Yes [4, 94] (Extended in this work)**	**Yes [79]**	**Yes (This work)**

vided appropriate workflows for NIF/NIS instantiation and replacement. However, in Section 3.5, the necessary workflows for the complete NI lifecycle are provided from its creation until its termination. Such workflows are aligned with the ITU pipelines [14]. Although Li et al. [86] mention the NI lifecycle management as a main issue in AI-native architectures, they did not discuss how it can be solved using the proposed AI-native architecture. Regarding the European projects, only Hexa-X considers an AI orchestration function deployed as a network slice subnet management function responsible to instantiate and operate the various NIFs. Using the AI functions defined above, the AI orchestration function manages to configure and implement the proper pipeline to generate and execute new versions of a specific ML model.

A common denominator among research is the importance of data. In [85], the pervasive nature of the NI is intricately linked to a distributed data infrastructure. The ability to execute and, when necessary, train AI models relies on the ubiquitous availability of data and computing resources. Moreover, the data ingestion speed defines the "fast and slow" intelligence in [87]. Despite its importance, only Hexa-X-II states that the quality of the ML models heavily depends on the data quality and included DataOps as a technological enabler for the 6G architecture. Recognizing this gap, in [79], we analyzed the challenges and requirements imposed by the distribution and management of data among disaggregated infrastructure and the impact of operating at different timescales for control systems. Moreover, we analyzed the NI design concerning three fundamental aspects: data, decision-making, and decision enforcement, which are fundamental aspects of realizing the NI Stratum.

As ongoing initiatives, Hexa-X-II and the national research projects mentioned above have yet to propose a comprehensive architecture for next-generation networks. However, they unanimously agree that such an architecture should be based on modular principles, utilizing cloud-based concepts and integrating AI/ML from analytics to decision-making within the network infrastructure. In contrast, our work incorporates all these principles and establishes a foundational architecture that these initiatives can build upon. While Hexa-X's deliverable D1.4 [95] published a final release of its E2E architecture and provided numerous enablers and mechanisms for the orchestration of intelligence, these elements are not fully integrated into the architecture presented in D1.3.

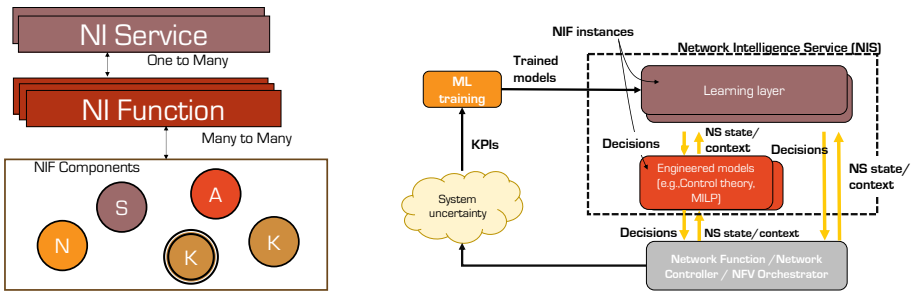


Figure 3.1: The high-level hierarchical taxonomy of NI algorithms. An NIF corresponds to an individual NI instance that assists a specific functionality.

3.3 Architectural Design of the NI Stratum

Our proposal revolves around the complete design of a *NI Stratum* designed to fulfill multiple objectives within an AI-empowered network infrastructure. Firstly, the NI Stratum aims to facilitate closed-loop NI systematically throughout the entire end-to-end network architecture. Secondly, it seeks to enable the coordination of various NI instances deployed across the network, fostering collaboration, exploiting synergies, and effectively managing conflicts. Thirdly, the NI Stratum defines essential interfaces that NI algorithms can utilize to interact with their respective local environments. In addressing the limitations of existing academic research and industry related to the NI Stratum, this framework is conceived as an orthogonal approach where NI instances can effectively be integrated into the traditional planes (data, control, and management) for easy adoption in the industry, complementing existing architectures.

3.3.1 Defining and designing intelligence in the network

From an architectural viewpoint, our conceptualization for NI management hinges upon similar design principles to those underpinning the management of NSs in 5G networks, e.g., ETSI NFV MANO [64]. This approach enables the adaptation of familiar concepts to the domain of network intelligence and facilitates the integration of the NI Stratum with existing 5G architectural frameworks. Building on this strategy, and in a manner analogous to the information model outlined for network management by entities like 3GPP, we introduce the notions of NIF and NIS, defined as follows.

NIF: This functional block within a NI instance implements decision-making functionality for deployment in a controller, NFV orchestrator, or individual NF. It features well-defined interfaces and behavior corresponding to an individual NI instance that serves a specific functionality.

NIS: An assembly of NIFs with a specific objective, often associated with a particular set of targeted KPIs.

To facilitate the modeling of any NI algorithm, we represent complex NI algorithms as a hierarchy of NISs that can be broken down into one or more NIFs. There is a one-to-many relationship between NIS and NIFs, as the former could be provided by one or

more instances of the latter. NIFs themselves could be of different kinds: they could be learning models based on, e.g., Deep Neural Networks (DNNs) or Engineered Models, or they could be built upon specific optimization algorithms such as the ones based on control theory or Mixed-Integer Linear Programming (MILP). The heterogeneous definition of NIFs is not limited to complex AI models but also encompasses traditional and interpretable models that are not necessarily data-driven.

The high-level interactions among the building blocks mentioned above are illustrated in Figure 3.1. A NIF may engage in two primary interactions with the underlying layers (i.e., control plane, user plane, or infrastructure, represented by an NFV Orchestrator): it may (i) contribute decisions and (ii) receive information concerning the network and the contextual state of such a configuration. These two interactions inherently implement a closed-loop NI. On top of this, a NIS represents a collaborative effort involving one or more NIFs, potentially organized in a hierarchical structure. As an example, a NIS might consist of a learning-type NIF providing decisions to an engineered model NIF, which, in turn, influences the underlying infrastructure.

When diving into the internal functioning of a single NIF, we employ a methodology akin to the MAPE-K feedback loop to break down the stages of the closed-loop operation performed by the NIFs. MAPE-K is recognized as one of the most influential reference control models for autonomic and self-adaptive systems [1]. One of the main functionalities of the NI Stratum and the NIO is to provide the lifecycle management of NISs and NIFs, as it will be shown later. Consequently, the NIO should ensure that the creation and implementation of the NIF also follows this closed-loop control. However, from the original definition of the MAPE-K it is not possible to elucidate if the NI is being trained or used in inference and distinguishing between different ML types (e.g., Supervised Learning (SL) and Reinforcement Learning (RL)). Therefore, we introduce an extended Network Monitor-Analyze-Plan-Execute over a shared Knowledge (N-MAPE-K) [79] model tailored to the NI environment, which extends the legacy MAPE-K with original training and closed control loops that a NIF may implement, as shown in Figure 3.2. The N-MAPE-K model allows capturing (i) the inference loop, (ii) a traditional supervised training loop, and (iii) a second training loop dedicated to online learning.

In particular, a learning block is introduced to optimize an objective function. Notice that the objective function optimizes the algorithm's learning (e.g., minimizing the cross-entropy) according to the learning problem (e.g., classification), and it is not necessarily related to the optimal solution of the networking problem (e.g., selecting the best modulation scheme to minimize interference). Moreover, it differentiates where the outcomes of the NI are being directed: the network, when the NI is used in inference, or its digital twin, when the NI is used in training or tuning. A digital twin network is a virtual representation of a physical network. Thanks to their access to real-time data, digital twins can accurately model complex systems. This way, multiple decisions can be tested safely without impacting the production network [25].

Mapping NI algorithm components into the N-MAPE-K representation allows highlighting the following fundamental classes of atomic Network Intelligent Function Components (NIF-Cs). The following chapter will exemplify how an NI can be decomposed into multiple components, as proposed here.

- **Sensor NIF-Cs** specify all the monitoring probes needed to gather the input mea-

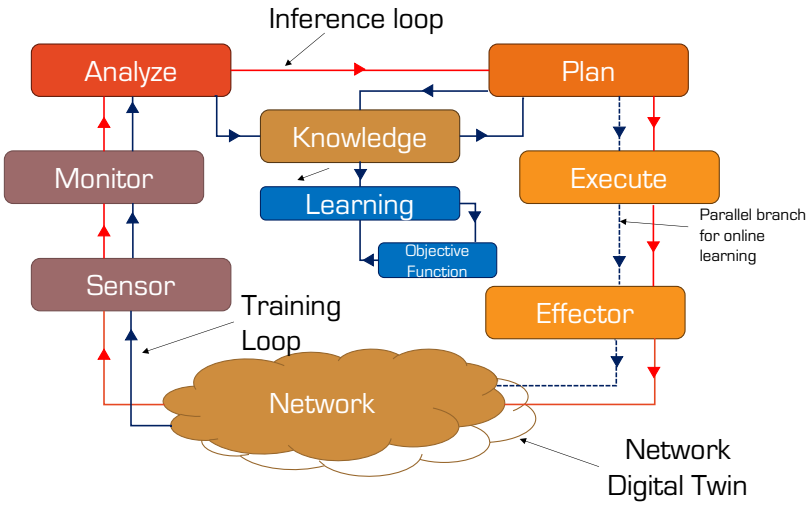


Figure 3.2: Extended N-MAPE-K abstractions for NI algorithms. The red loop represents the workflow when using the NI in inference, while the blue loop represents a supervised training loop.

surement data.

- **Monitor NIF-Cs** specify how each NIF interacts with the Sensor NIF-Cs and gathers their raw data.
- **Analyze NIF-Cs** include any pre-processing, summary, or data preparation for the specific NI algorithm implemented in the plan NIF-Cs.
- **Plan NIF-Cs** constitute the specific NI algorithm implemented by the NIF.
- **Execute NIF-Cs** specify how the algorithm will interact with the managed system and how to possibly change its configuration parameters.
- **Effector NIF-Cs** specify the configuration parameters updated in the NF, and the Application Programming Interfaces (APIs) to be used to that end.

3.3.2 Network Intelligence Stratum

In the supervision and coordination of NISs, NIFs, and NIF-Cs, which collectively constitute the overall NI, we adapted the layered structure of the ETSI NFV MANO framework. This adaptation tailors the components to the specific requirements of NI. The resultant framework forms the *NI Stratum* and is illustrated in Figure 3.3; it is structured into three levels, namely (i) the NIO, (ii) the NIF Manager, and (iii) the NIF-C Manager.

NIF-C Manager: This component is responsible for managing the lifecycle of the NIF-C. This management encompasses various operations, including onboarding, instantiation, termination, scaling, and state retrieval. The NIF-C Manager handles these operations uniformly, regardless of the type of NIF-C (i.e., whether it is a *Source*, *Analyze*, *Plan*,

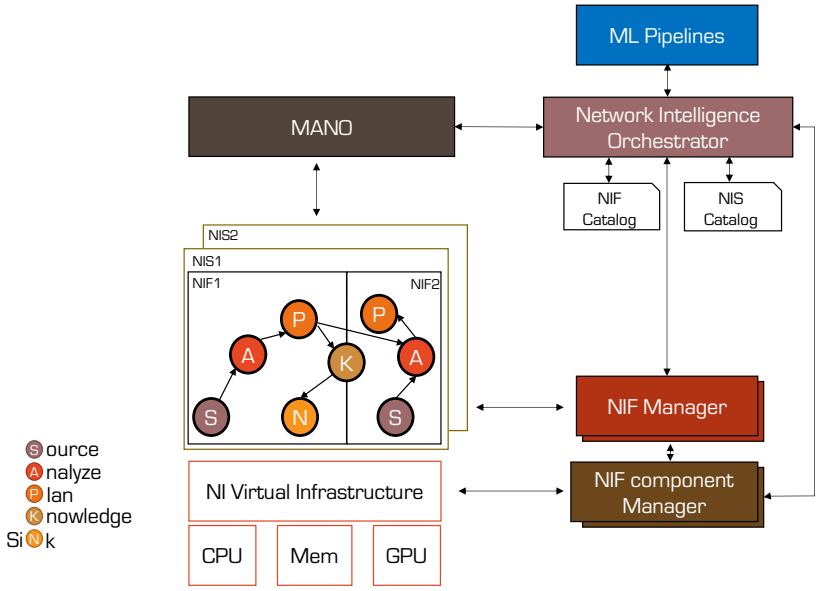


Figure 3.3: Architecture of the Network Intelligence Stratum.

Knowledge, or *Sink*) and its connection to the network infrastructure. For example, in the case of *Sources*, the IP addresses of different data producers need to be provided, while for *Sink*, specific configuration API endpoints must be configured. This component is an enhanced version of the Virtual Infrastructure Manager (VIM) in ETSI NFV MANO [64].

The instantiation specifics vary based on the context of this interaction. For instance, if the NIF operates from the core, *Sinks* and *Sources* integrate with the Network Repository Function (NRF) and the Network Exposure Function (NEF), synchronizing with the NWDAF [81], which captures analytics as a set of *Analyze*, *Plan*, and *Knowledge* components. Similar considerations apply to other network domains, such as the RAN, where this framework can be seamlessly integrated with the O-RAN xApps or rApps ecosystems [96].

NIF Manager: The NIF Manager, on the other hand, provides a comprehensive overview of the collective NIF-C set that forms each NIF. In addition to overseeing the lifecycle of the NIF, this module is tasked with monitoring the overall health of the intelligence functions. This monitoring involves continuous tracking of learning KPIs generated by the NIFs, including metrics like accuracy, particularly when the NIF is engaged in inference or serves as an online learning solution. Other metrics, such as loss and training loops, are monitored when the NIF is undergoing training. The NIF Manager is also responsible for configuring the meta-parameters of the models (via interaction with the NIF-C Manager) and conveying the health status of the NIF up into the hierarchy to the NIO.

Network Intelligence Orchestrator: This module is responsible for overseeing the lifecycle management of the NIS by effectively coordinating the NIFs that constitute each of them. This entails the ability to share NIF-C among different NIFs (e.g., two NIFs requiring the same input) and establishing arbitration policies when two NIFs share the

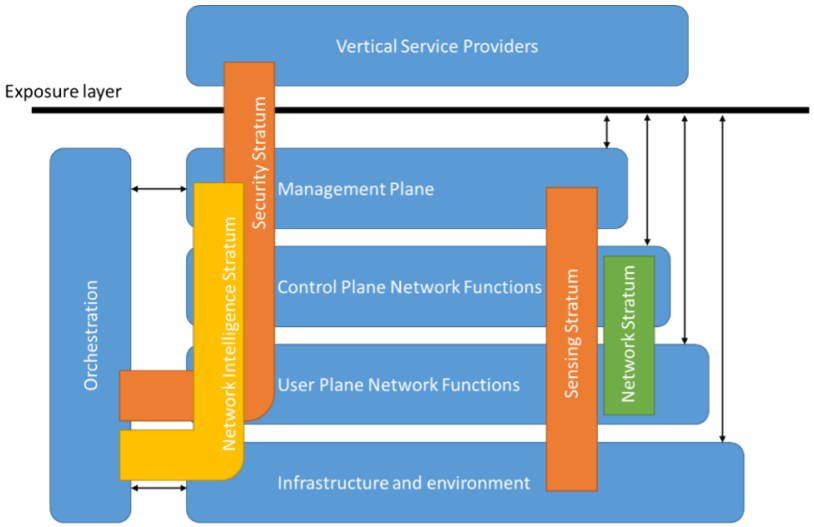


Figure 3.4: The 5GPPP Architectural WG framework [3].

same sink, specifically the configuration APIs. Importantly, this coordination occurs at the level of the NIO, no longer falling within the purview of the NIF Manager. It involves collaboration across NIFs, necessitating a higher-level perspective uniquely held by the NIO. The module also handles connections to network MANO frameworks for gathering crucial information, such as expected network KPIs for the managed slice and service and the status of the underlying network infrastructure. The NIO maintains catalogs of already onboarded NIS and NIFs. Notably, NIFs might require retraining to adapt to changing or diverse conditions, either periodically or on demand. In such cases, the NIO has the interfaces to an external platform to build ML pipelines and execute such operations, exemplified by an MLOps framework.

The proposed NI Stratum moves our previous network intelligence plane design [4] from a purely separate plane to a more orthogonal approach where NIFs and NISs can effectively be integrated into the traditional planes (data, control, and management) for easy adoption in the industry. This term, *Stratum*, has also been embraced as part of the comprehensive architectural framework that has been developed by the 5G Infrastructure Public Private Partnership (5GPPP) Architecture Working Group (WG), as illustrated in Figure 3.4 and reported in the whitepaper [3], released by the 5G Architecture WG in the 5GPPP. There, the term *Stratum* typically denotes a collection of elements that span various network domains. For example, *Network Access Stratum* encompasses all the elements involved in user registration and authentication across RAN and Core.

3.3.3 Functionalities Supported by the NI Stratum

The NI Stratum is a unified framework that brings together our earlier proposals for (i) the operational hierarchy of NI components in the NI Stratum and (ii) the N-MAPE-K representation of NIF-Cs. The variety of NIFs and NISs that can be deployed at the network generates new challenges in the way they should be managed that are not presented

in current management frameworks. Therefore, in [94], we discuss the need for specific NI Stratum procedures to address challenges arising from the concurrent instantiation of various NIFs and NISs. The challenges are exemplified by using two functionalities to improve the resiliency of a vRAN system [97], [98, Sec. 2.5]. The main challenges to be managed by the NI Stratum are Conflict Resolution, Knowledge Sharing among NIFs, Model Selection, Catalog, and Re-training. Moreover, the NI Stratum incorporates functionalities such as Data Analytics, Knowledge Management, Monitoring, NIS Lifecycle Management, NIS Creation/Selection, Optimization, and Instantiation, Model Explainability, Policy Interpreter and Configuration, NIS Workflow Configuration, Network MANO Framework, and Conflict Detection and Resolution. Although resource allocation for deploying NIF-Cs in the physical infrastructure is challenging, we assume appropriate solutions are in place, given the extensive research on the topic [99].

Conflict resolution capabilities are crucial for efficiently reusing and combining elements across NIFs to build NISs. By representing NIFs as atomic NIF-Cs within the N-MAPE-K framework, conflicts may arise in the sharing of different NIF-C elements when composing NIFs to create NISs. In the two NIF examples mentioned earlier, conflicts may occur when monitoring data, requiring the NIF Manager to ensure information arrives with the necessary granularity, and in policy enforcement, where different NI algorithms may configure the same network functions differently. The NIO is tasked with deploying conflict resolution policies to guarantee optimal decisions, overseeing individual NIFs, and monitoring access to data sources and policies to amend sub-optimal decisions.

Additionally, the NIO plays a crucial role in providing centralized coordination among multiple NIFs, enabling knowledge sharing for synergistic performance improvements. For example, the knowledge learned by a NIF can support other NIF's decisions, and vice versa. Such knowledge-sharing capability can extend across domains: for instance, in Section 4.2 of [100], the presence of an anomaly detection solution for Internet of Things (IoT) platforms, where the user plane traverses multiple domains; in this scenario, the NIO facilitates synchronization among parties involved in building the user plane for IoT devices, addressing challenges in root-cause analysis of anomalies.

Model Selection, Catalog, and Re-training are essential for NIS to adapt to the underlying environment. While not directly derived from NI algorithms' design, NIS may require knowledge of the software/hardware environment and device location. In a pure ML environment, tasks are handled by MLOps frameworks like Kubeflow [50] and mlflow [49]. However, in a AI-native architecture, close interaction with the orchestration environment is necessary. The NIO ensures that deployed NIFs match the specific hardware-software-environmental characteristics of network functions. It exchanges execution context information with the MANO system to select the appropriate model for inference within a NIF. This involves maintaining a model catalog from which the NIO selects the most suitable model based on the network's infrastructural status. If no model is available, the NIO can invoke training of a new model, fetching the required data as the target algorithm needs.

The NIO plays a central role in managing and coordinating NIFs to enable AI-native architectures. In response to challenges in deploying multiple NIFs concurrently, the NIO incorporates several key functionalities, which are summarized in Figure 3.5 and listed as follows.

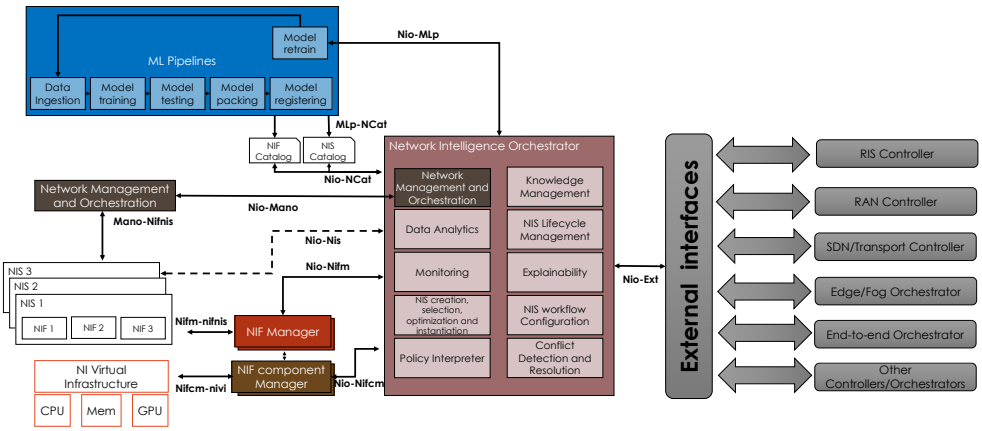


Figure 3.5: The NI Stratum and the functional blocks of the NIO and ML pipelines, with the internal and external interfaces of the Stratum.

- *Data Analytics*: Involves pre-processing or preparing data for the NIFs by computing statistical measures (e.g., averages, variance, maximum or minimum values) or more complex features (e.g., embeddings via autoencoders or dimensionality reduction techniques, aggregations via clustering algorithms).
- *Knowledge Management*: Critical for planning, organizing, acting, and controlling knowledge across all deployed NISs.
- *Monitoring*: Processes information from NISs, covering both ML-related (model-specific metrics, data drift) and non-ML-related aspects (QoE, QoS), monitoring NIs in both training and inference deployments.
- *NIS Lifecycle Management*: Handles deployment and maintenance of ML models, aligning with MLOps practices, including the creation of new ML pipelines for re-training models.
- *NIS Creation/Selection, Optimization, and Instantiation*: Involves selecting, optimizing, and instantiating NISs based on hardware constraints, with the ability to create a new NIS if it is unavailable in the catalog.
- *Model Explainability*: Provides methods for human experts to understand black-box ML algorithms within the NISs, aiding in comprehending decision-making processes.
- *Policy Interpreter and Configuration*: Interprets high-level user intent objectives associated with different NIS, performing changes in policy as needed.
- *NIS Workflow Configuration*: Integrates data engineering, ML, and Development Operations (DevOps) to operationalize deployment, monitoring, and lifecycle management in a modular and flexible way.
- *Conflict Detection and Resolution*: Provides a mechanism to solve trade-offs arising from conflicting objectives in control and user planes, allowing the NIO to compare policies among different NISs and perform conflict resolution.

The NIO interacts with the MANO framework, synchronizing network slices, tracking the state and health of network slices and functions, and obtaining real-time information about available resources. This collaboration optimizes network operations, enhances resource utilization, and ensures alignment with vertical service providers' requirements for specific vertical service domains. The NIO-MANO interaction involves eastbound-westbound interfaces, direct extensions to MANO modules, and mappings with MANO components such as Network Function Virtualization Orchestrator (NFVO), Virtual Network Function Manager (VNFM), and VIM in frameworks like ETSI NFV MANO.

The architectural design presented in this section is complemented next in the following ways: (i) in Section 3.4, by presenting and discussing the interfaces that are required to allow communication between internal NI Stratum components, and the NI Stratum components with external entities such as the RAN controller, Core system, and local and end-to-end management systems; and, (ii) in Section 3.5, by designing the set of procedures that address the needs and challenges introduced in [94] and that motivate the functionalities mentioned above.

3.4 Network Intelligence Stratum Interfaces

As shown in Figure 3.5, the NI Stratum is a composition of different functional blocks that aims for the native integration of NI in the network by providing the management and orchestration capabilities for NIF and NIS. Similar to other well-known frameworks for management and orchestration on specific domains, e.g., NFV-MANO [64] and O-RAN [101], the functional blocks of the NI Stratum have their own set of internal interfaces. Moreover, external interfaces will allow the NIO to communicate with external orchestrators, facilitating efficient resource coordination and orchestration of NI across diverse network environments for improved interoperability and scalability. In the following, we will provide a high-level definition of such interfaces and the interactions that they could enable. For more details, refer to Section 3.5.

3.4.1 Internal Interfaces

To successfully orchestrate and manage NI, it is essential to establish seamless communication and coordination among the various functionalities of the NI Stratum. In this subsection, we will outline and elaborate on the specific set of internal interfaces presented in Figure 3.5. These interfaces are the foundation for enabling effective communication and coordination among the different blocks within the NI Stratum, ensuring a harmonized and cohesive NI management framework. In the following, we present them according to their functional definition, although from an implementation perspective, they could be provided in a service-based fashion.

- **Nio-Nifm**. This interface allows communication between the NIO and the NIF Manager to effectively manage and orchestrate NIF instances within the NI Stratum framework. Among life-cycle management, the NIO relies on the NIF Manager to perform operations related to NIF instances, including instantiation, scaling, healing, and termination. Via this interface, the NIF Manager can also provide mon-

itoring information about the learning performance (e.g., the loss function when trained), or network performance indicators, and trigger healing actions in case of failures, degradations, or conflicts. Moreover, the NIO can gather information related to the status of the NIFs so it can derive analytics to proactively optimize the NIFs (e.g., by changing the learning model data feeding speed/timescale to mitigate limitation on available computing resources) or control it (e.g., by adding a new input representation of the data or ML model to couple it with other NIF when instantiating a new NIS). Finally, the NIO can also gather information from the NIFs related to explainable capabilities and use it to take better orchestration and coordination actions among NIFs. Finally, this interface will allow the NIO to perform ML workload management. In that way, the interactions enabled by this interface promote efficient utilization of network resources, optimized network service delivery, and enhanced scalability and flexibility of virtualized NIF.

- **Nio-Nifcm**. This interface allows the NIO to request the NIF-C for the allocation, placement, and lifecycle management of virtualized infrastructure resources. These resources include computing (Graphics Processing Unit (GPU), Field Programmable Gate Arrays (FPGA), Central Processing Unit (CPU), memory), storage, and networking components required to host and run NIF instances. It will also allow for gathering information about the utilization and performance of virtualized infrastructure resources. This includes monitoring the allocated resources' availability, capacity, and performance metrics and providing visibility into resource usage and potential bottlenecks. In case of the need for infrastructure policy enforcement, this interface allows the NIO to enforce policies and constraints on the virtualized infrastructure resources such as security policies, learning, and QoE/QoS requirements, or specific compliance regulations that need to be applied to the infrastructure hosting the NIFs (e.g., data privacy, data anonymity, model isolation or federation, etc.).

- **Nifm-Nifnis**. This interface enables the NIF Manager to manage the lifecycle of NIF instances. It allows the NIF manager to perform operations such as NIF instantiation, scaling, healing, termination, and update. In the case of configuration and monitoring, this interface allows the NIF Manager to provide configuration parameters and policies to the NIF through the interface. Additionally, it can collect monitoring data and performance metrics from the NIF instances to ensure their proper functioning and adherence to SLAs in terms of both networking (e.g., QoS and QoE) and learning (e.g., accuracy). This NIF Manager can also perform fault and performance management. The NIF Manager receives fault notifications and performance data from the NIFs through the interface, allowing it to detect and handle any issues that may arise based on policies defined by the NIO. This includes fault localization, resolution, performance optimization, and ensuring the desired performance of the NIF. Finally, the NIF Manager can manage the state and context of the NIF instances. It allows the NIF Manager to retrieve and update the state information of the NIFs, including their operational status, configuration parameters, and runtime data. This information is crucial for maintaining the consistency and continuity of the NIF operations. This interface can also provide the capabilities to monitor, manage, and orchestrate NIS based on abstract data information such as model knowledge (e.g., Neural Network (NN) weights, expert knowledge encapsulated in rule-based systems) and explainable model data. Moreover, it will gather information about the NIF composition to detect possible conflicts in NIS

before deployment, given its topological structure, or after re-orchestration of the NIS when NIF are added/removed/changed. In conjunction with the *Nio-Nifm* interface, this interface allows the NIO to also configure the NIS (e.g., adding a new NIF in the NIS). In some implementations, the interaction between NIO and NIS can be done via a specific interface, e.g., a Nio-Nis interface.

- **Nifcm-Nivi**. This interface allows the NIFs to interact with the NI virtualized infrastructure, which includes virtual machines, containers, resources for storage, and networking components. This interface allows NIFs to utilize the underlying infrastructure to perform their designated functions efficiently. For example, allowing an ML model to switch among different computing hardware (e.g., CPU, GPU, Tensor Processing Unit (TPU), or FPGA) and modes (training versus inference).

- **Nio-MLp**. This interface enables the NIO to enact ML model (re-)training via MLOps.

- **MLp-Ncat**. Via this interface, the ML pipeline framework in the NI Stratum can access the model register, which serves as a critical connection point in managing and organizing ML models empowering NIF/NIS within the pipeline framework. This interface enables seamless integration and coordination between the pipeline framework and the model register, facilitating efficient model versioning, storage, retrieval, and tracking. This interface streamlines the integration of ML models within the pipeline, enabling seamless collaboration, reusability, and scalability of models across the ML workflow.

- **Nio-Ncat**. This interface allows the NIO to access the catalog of NIF/NIS available to deploy in the network. By accessing the catalog, the NIO can effectively discover, select, compose, onboard, and manage the lifecycle of NIF/NIS within the NI Stratum. The interface enhances the agility, flexibility, and automation capabilities of the NI orchestration system, enabling seamless deployment and efficient management of NIF/NIS within the NI virtual infrastructure.

- **Nio-Mano**. The implementation and deployment of the NIO can determine whether MANO functionalities are integrated within the NIO or external to it, thus it can be seen as an internal or external interface. This decision primarily involves a trade-off between a self-contained orchestrator capable of creating, instantiating, and deploying legacy NF/NS, ML-only NIF/NIS, and hybrids NIF/NIS, and a lighter orchestrator that relies on external MANO for tasks such as managing legacy NIF/NIS as legacy NF/NS. The Nio-Mano interface is used in the later case. When the MANO is deployed as an external functional block of the NIO (e.g., in legacy systems where MANO functionalities are already in place), this interface provides the communication mechanism to exchange real-time information to track network slices, function states, and resource availability. This synchronization allows the NIO to dynamically adapt decisions and efficiently allocate resources based on the current network characteristics. By maintaining an up-to-date view of available resources, including computing power, storage, and network capabilities, the MANO can orchestrate network resources effectively and optimize resource utilization, thereby improving performance. Thanks to this interface, the NIO can be aware of such optimization.

- **Mano-Nifnis**. This interface allows MANO functionality (either internal or external to the NIO) to perform orchestration commands directly on NIF/NIS. For example,

operations such as deployment, scaling, updating, or decommissioning of ML blocks can be performed via this interface or monitoring and reporting of ML metrics for the data analytics and monitoring block. Also, advanced functions such as security, e.g., to enforce security policies against adversarial attacks on ML models or conflict detection after deployment, can interact with the NIF/NIS via this interface.

- **Nio-Ext**. This interface communicates between the NIO and external orchestrators/controllers in the network in the same or across multiple domains. This interface will be detailed in the following section.

To promote industry deployment, validation, and widespread adoption of standardized APIs, we highly recommend that these interfaces are designed following an OpenAPI representation in Yet Another Markup Language (YAML) and JavaScript Object Notation (JSON) where available (e.g., via ETSI or Institute of Electrical and Electronics Engineers (IEEE)), similar to the NFV-MANO core APIs. Moreover, tools to navigate the specifications and report bugs should also be provided to enhance the usability and effectiveness of the OpenAPI representation.

3.4.2 External Interfaces

The **Nio-Ext** interface will allow the NIO to communicate with external orchestrators/controllers to achieve efficient collaboration, resource coordination, and NIF/NIS orchestration across heterogeneous network environments (far edge, edge, RAN, transport, core, cloud, etc.). The interface enhances interoperability, scalability, and flexibility, effectively managing and orchestrating resources and NIF/NIS in complex network ecosystems.

This interface allows for efficient coordination of resources by exchanging information about available resources and their utilization across different domains. This promotes optimal resource allocation and utilization. Secondly, the interface enables collaboration in NIF/NIS deployments across multiple domains by facilitating the exchange of NIF/NIS-level information and dependencies between the NIO and external orchestrators/controllers. This enables the instantiation, management, and scaling of complex NIS across multi-domain and heterogeneous environments.

Additionally, the interface supports policy management by facilitating the exchange of policy information between the NIO and external orchestrators/controllers. This ensures consistent policy implementation and governance across different domain systems. Moreover, the interface enables the exchange of event and alarm information, allowing for proactive event handling, correlation, and remediation across domains. Finally, the interface facilitates information exchange and federation by enabling the sharing of network topologies, hardware capabilities, NIF/NIS catalogs, and other relevant data (e.g., monitoring information, model weights, etc.), improving decision-making and coordination capabilities among different orchestration systems. In this section, we will describe two specific cases of such interfaces.

3.4.2.1 O-RAN

The O-RAN Alliance is a global community of mobile network operators, vendors, and research institutions established in February 2018. Its primary goal is to drive the development of open, intelligent, and interoperable RAN technologies. Founded by AT&T, Orange, Deutsche Telekom, Docomo, and China Mobile, O-RAN is now supported by over 300 organizations, including major operators and vendors. Analysts predict that open vRANs could surpass the conventional RAN market by 2028, generating revenues close to $20 billion.

The O-RAN architecture is a new approach to building mobile networks that aims to increase flexibility, interoperability, and innovation. It is designed to enable multi-vendor deployments, reduce costs, and improve network performance. Key aspects of the O-RAN architecture are presented in [102], where a very important aspect of O-RAN is the integration of AI/ML workflows, i.e., NI that may be managed by the NI Stratum, with the following principles [101]:

- **Offline Learning**: In O-RAN, even for RL scenarios, some amount of offline learning (where a model is trained with offline data before deployment) is always recommended.
- **Pre-training and Testing**: Any model deployed within the network needs to be trained and tested beforehand. No completely untrained model should be deployed in the network.
- **Modularity in ML Applications**: As a best practice, ML applications should be designed in a modular fashion, with the capability to share data without knowledge of each other's data requirements. The location or nature of a data source should not bind them.
- **Service Provider's Deployment Choice**: The criteria for determining where an ML application should be deployed (Non-RT RIC or Near-RT RIC) may vary between service providers. Therefore, the service provider should decide the deployment scenario for a given ML application.
- **Optimization of ML Model for Efficiency and Performance**: To improve execution efficiency and inference performance, the ML model should be optimized and compiled considering the hardware capabilities of the inference host. There should be a balance between efficiency and inference accuracy, with acceptable accuracy loss as one of the optimization goals. The optimization parameters should be determined based on this threshold.

Figure 3.6 illustrates the general framework of AI/ML procedures and interfaces and its integration into the proposed NI Stratum, including the potential mapping between ML components and O-RAN components.

O-RAN suggests several AI/ML deployment scenarios that are relevant to our NI Stratum; they are summarized as follows:

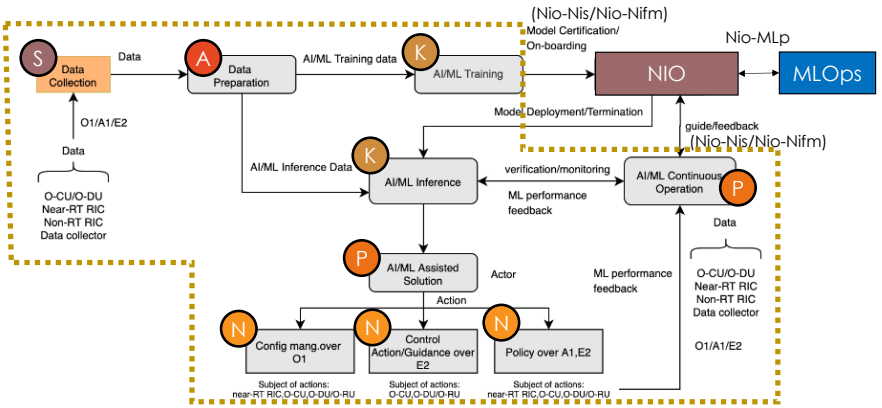


Figure 3.6: Integration of the NI Stratum and O-RAN AI/ML Lifecycle Procedures and Interface Frameworks

- **Deployment Scenario 1.1**: In this case, AI/ML Continuous Operation, Model Management, Data Preparation, Training, and Inference all take place within the Non-RT RIC.
- **Deployment Scenario 1.2**: Here, AI/ML Continuous Operation, Data Preparation for training, and AI/ML Training are located in non-RT RIC. However, AI/ML Model Management is outside non-RT RIC (either within or outside the Service Management and Orchestration (SMO)). Data Collection for inference, Data Preparation for inference, and AI/ML Inference are in the Near-RT RIC.
- **Deployment Scenario 1.3**: AI/ML Continuous Operation and AI/ML Inference are within non-RT RIC. Data Preparation, AI/ML Training, and Model Management are located outside the non-RT RIC (either within or outside SMO).
- **Deployment Scenario 1.4**: In this scenario, the non-RT RIC acts as the ML training host for offline model training, and the Near-RT RIC acts as the ML training host for online learning and also as the ML inference host.
- **Deployment Scenario 1.5**: Continuous Operation, Model Management, Data Preparation, and ML Training Host are in non-RT RIC. However, the Open Central Unit (O-CU)/Open Distributed Unit (O-DU) acts as the ML inference host.

Please note that the deployment of “AI/ML Continuous Operation" outside of non-RT RIC is still under study.

3.4.2.2 The 5G-Core

The 5G Core (5GC) is one of the most important domains in a 3GPP mobile system, hence we analyze how the proposed NI Stratum can interact with it. The imperative of network automation drove the design of the 3GPP system in R15, marking a significant departure from previous releases. In earlier iterations, data generation and analytics in the network primarily relied on proprietary interfaces for exchanges between network elements and

their respective managers. However, with R15 and subsequent consolidations, the architecture underwent a comprehensive overhaul to incorporate native support for collecting analytics. As explained below, these analytics can be effectively utilized to establish feedback loops through standardized or proprietary solutions. At the heart of this system lies the NWDAF, which performs three key functions: (i) aggregating data, encompassing metrics that reflect the current state of the network, sourced from another producer NFs; (ii) conducting analytics, involving the computation of refined statistics based on the gathered data; (iii) sharing the computed analytics with other consumer functions across the network.

The generated analytic reports serve as outputs that either present statistics based on historical data or provide predictions for specific metrics, depending on whether the requested timeframe is in the past or future, respectively. These outputs are crucial in optimizing the operation of NFs. Additionally, the output may include a confidence parameter, ranging from 0 to 100, which conveys information about the reliability of the prediction made. Factors determining this confidence parameter may include the volume of data utilized in generating the prediction, the age of the AI model employed, and other relevant considerations.

Figure 3.7 presents the interconnections among various components. The framework is divided into three closed loops and shows where the NI Stratum takes a role. The first closed loop, in red inside the 5GC, is where the NWDAF resides. Within this closed loop, other NFs of the core act as the primary producers and consumers of data and analytics. These NFs utilize the data and analytics to drive network operations in a data-driven manner. Thanks to the NWDAF, consumer NFs no longer need to directly communicate with every potential producer to compute analytics, as they can efficiently leverage the shared information. NWDAF is a specific (and very important) NIF, that can leverage on a number of NIF-C according to the analytics that are served.

The second closed loop, in orange, encompasses OAM activities, which involve modules such as Element Managers or Network Elements in pre-5G networks. Starting from R15, OAM effectively enforces network slicing through the service-based management architecture. The OAM closed loop can also supply the NWDAF with data from the RAN and 5G NFs, such as resource consumption. Unlike the pre-5G 3GPP RAN architecture, which lacks an analytics hub like the NWDAF, alternative architectures like O-RAN feature dedicated analytics modules. The Management Data Analytics Function (MDAF) serves as the module responsible for interacting with the NWDAF and provides Management Data Analytics Services (MDAS). As discussed, the MDAF collaborates with the NWDAF and other core NFs to generate management analytics information, which is subsequently consumed by other NFs or management procedures like the self-organizing network. From the perspective of the NI Stratum, the MDAF is a NIF that can be further split into several NIF-C which (i) interact with the NWDAF, effectively closing the loop with the core, and (ii) allows the internal interaction within the management domain.

The third closed loop, in green, encompasses the service domain, which is facilitated through the Application Function (AF). These functions outside the 3GPP trust domain play a crucial role in facilitating close interaction between service providers and network operators. This interaction is achieved through enriched service layers, which aid in commoditizing the network and enhancing the interplay between the service and network intelligence. Given the criticality of authorization and security, verifying whether AFs are

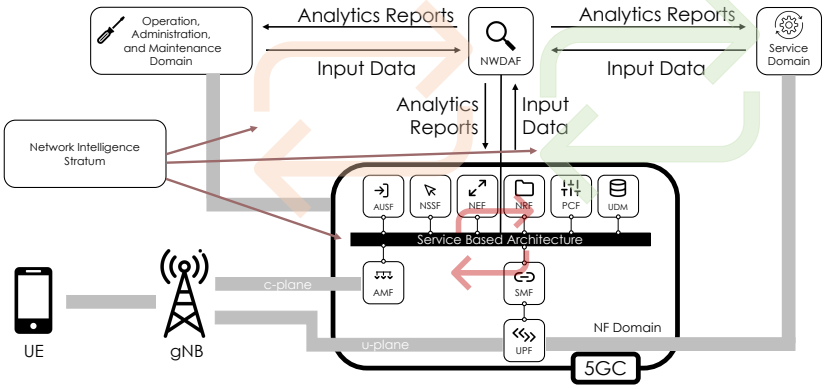


Figure 3.7: Interactions between the 5GC, the NWDAF and the NI Stratum.

appropriately authorized to interact with the NWDAF and engage in data exchange with third parties is essential. Authentication can be managed in three different ways. One is basic user-password authentication, where credentials are configured via a configuration file. Support of Transport Layer Security (TLS) protocol where there is a server-side authentication or mutual TLS authentication, where both server-side and client-side authentication are required. In this case, the AF can be seen as a specific NIF-C (either *Sink* or *Source*, depending on the context). Overall, any NF deployed within the 5GC, the OAM system or any AF can contribute input to the NWDAF and request analytic reports from it. This establishes a feedback loop where any NF, OAM component, or AF can provide input data to the NWDAF and receive analytic reports generated from the collective data obtained by the NWDAF. Through these feedback loops, the majority of automated network operations can be executed, as exemplified by the ones already provided by the NWDAF in the standard.

While the NWDAF facilitates the standardized integration of AI/ML into the core network, offering significant benefits through advanced analytics for network optimization, it does not encompass the lifecycle management of the AI/ML models performing these analytics. AI/ML techniques are essential for analytics because collecting and processing large volumes of data from various network functions can create significant overhead, potentially impacting network performance if not managed efficiently. In this context, the NI Stratum supports the management of these models through the procedures detailed in Section 3.5.1. Overall, the NI Stratum offers a broader approach than the NWDAF and the analytics framework of 3GPP, while maintaining compatibility with the standardized procedures it proposes.

3.5 Network Intelligence Stratum Procedures

In this section, we show how the architectural building blocks that compose the NI Stratum will interact with each other. Since the main component of the NI Stratum is the NIO, we will show how its internal functionalities cooperate to solve the challenges mentioned in Section 3.3. Additionally, we will explore how the different components

work together to create a cohesive system that can effectively orchestrate intelligence across multiple domains. Through these interactions, the NIO will be able to address the challenges that can emerge when NISs are deployed across different network domains and operating in multiple timescales.

Notice that all the procedures mentioned below are depicted using a process view. This view answers how the system behaves, addressing concurrency and synchronization aspects. Unified Modeling Language (UML) sequence diagrams were selected as the most appropriate form. Next, we briefly describe how combining some functional blocks can help address the challenges described in the previous section.

3.5.1 Inter NIO Procedures

One of the most essential management and orchestration capabilities is to handle the lifecycle of each of its entities. The NIO is not an exception. Regarding networking functionalities, NFV MANO [64] is the referent architectural framework to look up to since it is the most prominent architecture in networks that provides lifecycle management of NFs. Lifecycle management is generally responsible for the following operations: creation, instantiation or deployment, management (e.g., model selection and optimization), and termination. However, given the intelligent nature of the NI, several factors must be considered while addressing their lifecycle management. In the following subsections, we will discuss in detail how the NIO performs lifecycle management of the different NI.

3.5.1.1 NIS Creation

Given that a NIS is a composition of one or more NIFs, creating a NIS follows a similar approach as the Service Function Chaining (SFC). When creating a new NIS, the NIO should verify that all the NIFs from that NIS are available in the catalog. If a NIF is unavailable, a new training should be started, e.g., based on user-defined Network Intelligent Function Descriptor (NIFD)/ Network Intelligent Service Descriptor (NISD). This training is represented by triggering a new MLOps pipeline. The data ingestion for training this new NIF should be coordinated between the NIO and the MLOps pipeline. Notice that this procedure only contemplates the creation of the NIF and not its usage.

Figure 3.8 shows the required interactions to create a NIS/NIF. In the first step, the NIO should process a NIS/NIF creation request through its API. A sender can submit this request, which could be a human, an AI agent, or another process with administration rights to trigger orchestration operations in the NIO. The sender identifies that a new NIS/NIF is needed to perform a given network operation and submits this request to the NIO. As input for this process, the NIO should receive a NIFD/ NISD which includes, but is not limited to:

- Learning mode, if the ML model supports online learning or if the training is made offline.
- Data on which the model is trained (whether the learning is online or offline). This field also specifies the format in which the input data is expected.

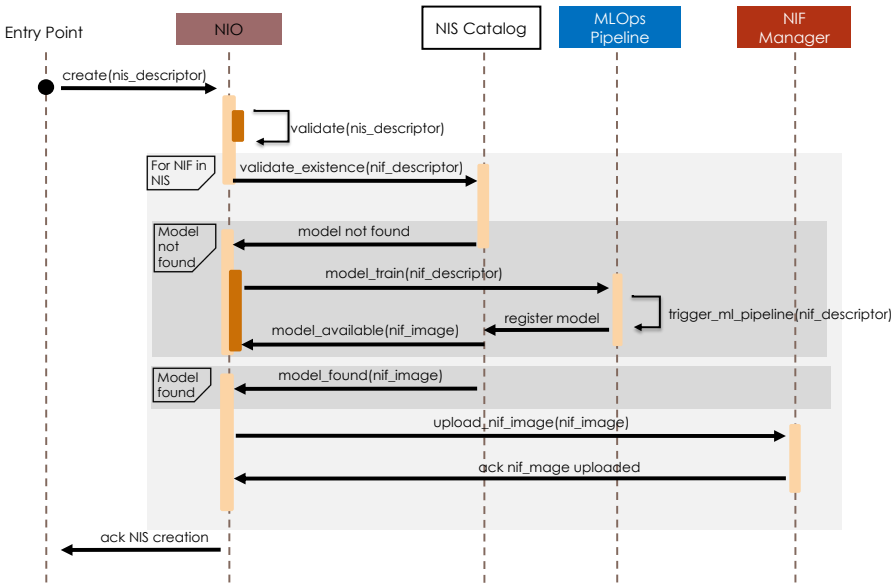


Figure 3.8: NIS creation process flow.

- Learning metrics. This typically includes accuracy, cross-entropy, or a known loss function, e.g., Mean Squared Error (MSE).
- Model performance upper and lower thresholds. Values on which the training can be concluded (upper threshold). It is assumed that once the upper threshold is met, the ML model is ready to be deployed in production. On the contrary, if the lower threshold is met, the ML model deployed in production should be updated. The definition of these thresholds may vary depending on the NI, but it should reflect the expected performance of the NI.
- Output format. This field specifies the format the ML will communicate its output. For instance, a classification problem can produce a vector with the probability of a given sample belonging to a class or the class itself.
- Last modification time. This field will indicate the age of the ML model. Given the constant evolution of network state and data, having an up-to-date ML model is crucial for network operation.
- Dependencies required for operation. ML models are created using specific libraries (e.g., NumPy, pandas, etc.). The right versions of such libraries must be available when instantiating the ML model in production.

As a second step, the NIO then processes the NIFD/NISD, by checking for the existence of mandatory elements (i.e., network operation, data requirements, output format, and accuracy) and validating the integrity and authenticity of the NIFD/ NISD. Afterwards, for every NIF in the NIS, the NIO verifies if the NIF model exists in the catalog. Two things may happen. If the NIF model is not present in the catalog, the NIO triggers a training operation from the ML pipeline resulting in the execution of a new data ingestion - model training - model testing - model packaging - model registering pipeline. Most

of the data needed to execute this pipeline is provided in the NIFD. Once the pipeline is completed, a new image from the NIF model is registered in the NIF Catalog. If the model is present in the catalog, it can be used in inference. For doing this, the NIO makes the NIS/NIF images available to each applicable NIF-C Manager. The NIF-C Manager acknowledges successful image uploading. Finally, the NIO acknowledges the NIS/NIF creation to the sender.

3.5.1.2 Instantiation or Deployment

Figure 3.9 shows the interactions required for instantiating or deploying a NIS/NIF. As in the previous step, the NIO receives a request to instantiate a new NIS. Then, several variants might be possible. If none of the NIFs belonging to the NIS is instantiated or deployed, the NIS instantiation will also include the instantiation of all the needed NIF instances through the NIF Manager. If all the needed NIF instances have already been created, the NIS instantiation would only deal with the interconnection of the corresponding NIF instances. Lastly, a combination of the above is possible where some NIF instances might exist, some might need to be created, and instantiated, and some network connectivity between the NIFs may already exist.

It is important to notice that if a NIF instance is already created, it can be shared between different NISs. In this case, the NIO should trigger the conflict resolution mechanism because they may be deployed on the same node and/or accessing the same resources. If no conflict is produced, the same NIF can instantiate the current NIS. If a potential conflict is detected, the NIO should proactively address it by deploying specific policies implementing rules or priorities (c.f., Section 3.5.2) to effectively solve the aforementioned conflict.

The main steps for NIS/NIF instantiation are as follows. First, the NIO receives a request to instantiate a new NIS/NIF. The NIO validates the request in terms of the request's validity, including validating that the sender is authorized to issue this request and validation of the parameters passed for technical correctness and policy conformance. For each NIF in the NIS, the NIO checks with the NIF Manager if an instance matching the requirements already exists. If such an instance exists, it will be used as part of the NIS. If the NIF instance does not exist, the NIO triggers the NIF creation procedure.

The NIO then should perform a feasibility check of the NIF interconnection setup. For doing this, the NIO requests to the NIF-C Manager the availability of resources needed for the NIF interconnection and reservation of those resources. The NIF-C Manager checks the availability of resources needed for the NIF interconnection and reserves them. The NIF-C Manager returns the reservation result to NIO. If the resources are not available, the NIS might not be instantiated, which results in a denial of the NIS instantiation. However, if the resources are available, the NIO requests the NIF-C Manager to allocate and interconnect the NIF instances. The NIF-C Manager instantiates the connectivity network needed for the NIS and finalizes with a completion acknowledgment. Finally, the NIO acknowledges the completion of the NIS instantiation.

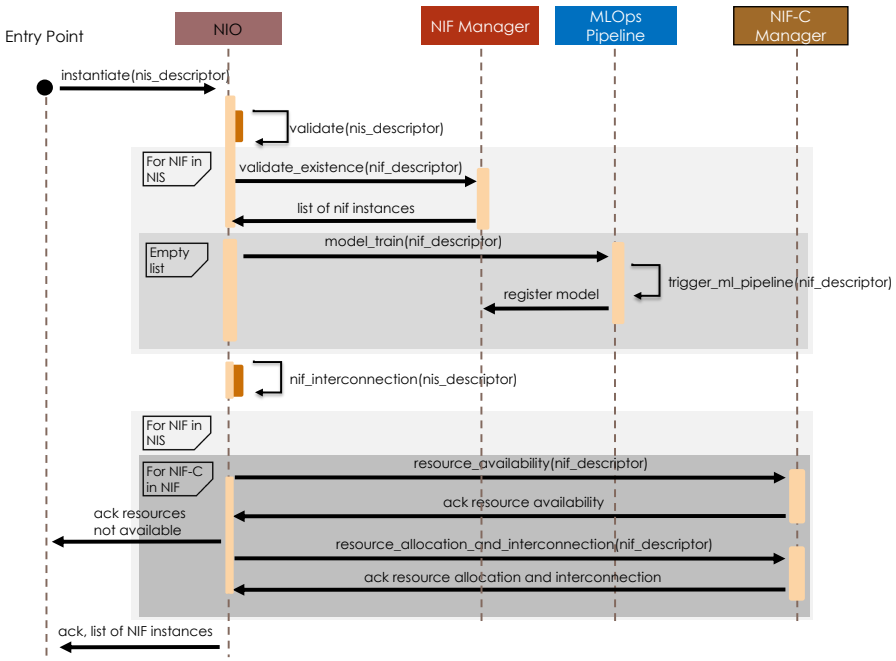


Figure 3.9: NIS instantiation process flow.

3.5.1.3 Management

Several operations can be considered as management procedures, such as NIS/NIF update, optimization, scaling, or migrating. NI solutions stored in the NIS/NIF catalog are trained on hardware and software platforms that may not match the ones available in the new environment where they need to be deployed. In such cases, the NIS creation/selection, optimization, and instantiation block will obtain networking and execution context information from its MANO block operating in the network and select the proper model to be used in inference within a NIF. Suppose a mismatch between trained and targeted hardware/software appears. In that case, the same block should perform the optimization/adaptation (e.g., compression of a neural network, change of inference library from GPU to CPU, etc.) to match the new environment. In case no model is available for the specific execution environment, the NIS creation/selection, optimization, and instantiation block will create a new NIS and then notify the NIS workflow configuration block to trigger a new training phase.

The proper functioning of NIFs depends heavily on the quality and representativeness of the ingested data. Since the network can constantly generate new data, two main issues may arise: changes in user behavior, external conditions, or operational environments may cause the input data's statistical properties to change over time, known as data drift. This leads to model degradation, where the ML model's performance deteriorates due to changes in the data or environment. The NI Stratum should ensure NIFs accuracy by ingesting recent, up-to-date data. To achieve this, the monitoring block of the NIO should incorporate data drift detectors [103], overseeing metrics like Age of Information (AoI) [104] to measure data freshness. High AoI can exacerbate both data drift and

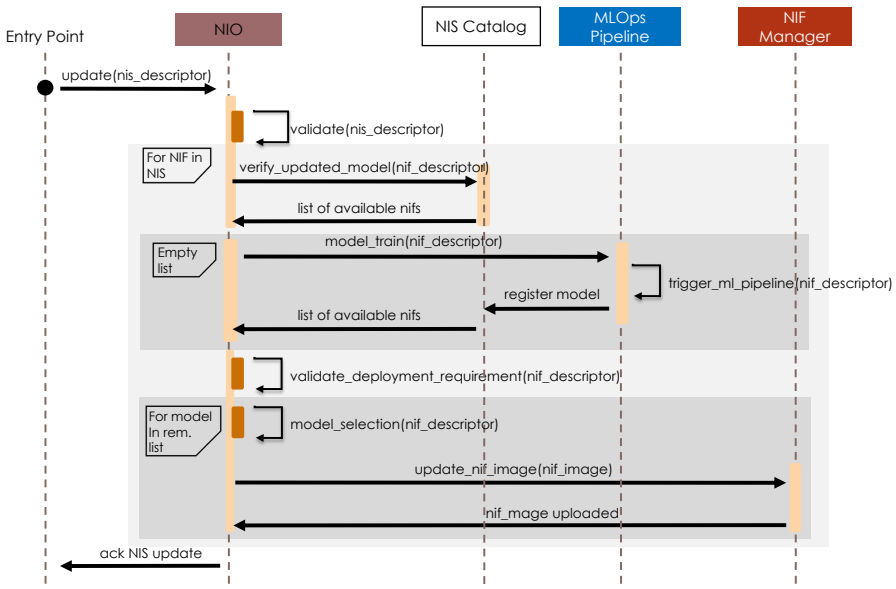


Figure 3.10: NIS update process flow.

model drift, reducing performance and accuracy. Once detected, the NIO should trigger NIF retraining or replace the ML model with one trained on more recent data. Here, we present the NIS/NIF update with model selection as the most relevant and generic procedure that may involve optimization, re-training, or selection.

Figure 3.10 shows the main steps for NIS/NIF updates. This procedure includes updating the parameters of the NIS/NIF. It is important to notice that the update process has similarities with the NIS/NIF creation. A request for NIS/NIF update is submitted from a sender, which could be a human, an AI agent, or another process in the architecture, such as a data analytics module that is detecting a mismatch of the statistics of the input data, or a monitoring module detecting that the current model's accuracy is lower than expected. The sender identifies that a new NIS/NIF needs to be updated and submits its request for an update through the NIO API. Then, the NIO processes the NIFD/NISD to check the existence of mandatory elements and validate the integrity and authenticity of the descriptor.

For every NIF in the NIS that must be updated, the NIO verifies that a new version of the NIF model exists in the catalog. Similarly, as in the NIS creation, if an updated model is needed but is not available in the catalog, the NIO triggers a re-training operation from the MLOps pipeline, starting a new pipeline. Once the model is re-trained, an image is registered in the NIF catalog. Consequently, the NIFD is updated with the new version and requirements (i.e., data format, hardware, software dependencies, etc.). On the contrary, if a model (or more than one model) is available, then the NIO verifies that the available models satisfy the deployment requirements in terms of data (e.g., input rate and format), computation platform (e.g., CPU, GPU, TPU or FPGA), dependencies (e.g., TensorFlow, PyTorch, etc.), and performance level. This process might return an empty list, meaning that no model satisfies the deployment requirement, and creating a new NIF is needed.

If more than one model satisfies the deployment requirements, model selection should be carried out. In this phase, the component will compute an ML test score, and depending on arbitration policies, the best-performing model is selected to update the NIF image. The ML test score can contain learning-related metrics (e.g., loss/reward function) and non-learning-related metrics (e.g., QoE, QoS, or stability in deployment). The arbitration policies are decision factors that the NIO considers primordial for model deployment, for instance, if model precision is preferred over energy consumption. If the model is updated, it should be registered in the catalog and can be used in inference.

Finally, once all the NIFs that compose the NIS are available, the NIO makes NIS/NIF images available to each applicable NIF-C Manager. Then, the NIF-C Manager acknowledges the successful uploading of the image, and finally, the NIO acknowledges the NIS/NIF update to the sender. Other management operations include optimization, scaling in/out, or migrating. The workflows are similar to those of NFV-MANO [64], requiring an extra step to update the NIS/NIF, which was shown above.

3.5.1.4 Termination

The request for terminating a NIS/NIF is received by the NIO. This request might come from a human, an AI agent, or another process in the architecture. When terminating a NIS/NIF instance, several variants might be possible. In case all NIF instances contributing to the NIS need to be terminated, a termination procedure is started for all the NIF, including the removal of the interconnectivity between these NIF. In case some NIF instances are contributing to other NIS instances, only those NIFs that do not contribute to other NIS instances must be terminated. The NIFs that are still in play are evaluated to check if an update is needed, e.g., scale-down their resources. The interconnectivity between them must be removed, leaving the other NIF instances in place and the interconnectivity between them intact.

For terminating a NIS/NIF instance, the NIO receives a request to terminate a NIS/NIF instance using the NIS/NIF Lifecycle Management interface. As in previous procedures, the NIO validates the request. It verifies the validity of the request (including the sender's authorization) and verifies that the NIS/NIF instance exists. The NIO then proceeds to request the NIF Manager to terminate any NIF instances that were instantiated along with the NIS instantiation, provided they are not used by another NIS. At the same time, the NIF Manager requests the deletion (release) of resources for this NIF instance to the NIF-C Manager, including the deletion (release) of their interconnection points. For all the NIF-C, the NIF-C Manager deletes (releases) the resources and then acknowledges the completion of resource deletion back to NIF Manager. This completes the deletion of the NIF. Once the NIS is terminated, the NIF Manager sends a confirmation to the NIO that the NIFs are terminated.

3.5.1.5 Other Operations

Operations such as deleting, querying, enabling, or disabling a NIS/NIF are also considered within the architecture defined by the NI Stratum. However, such operations are not different than those proposed in NFV-MANO as they do not involve or require interac-

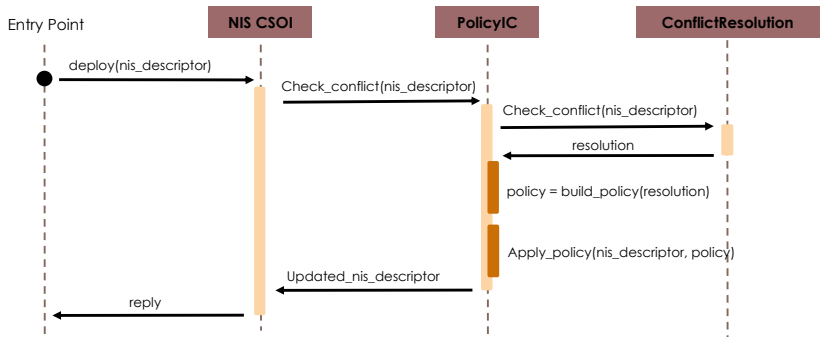


Figure 3.11: Conflict Resolution process flow.

tions with any NI-related block and the MANO block can perform it. The implementation of such procedures is shown in [64].

3.5.2 Intra NIO Procedures

As introduced above, a NIS is usually composed of different NIFs and hence, some of the NIS management functionalities take place only within the NIO itself. These intra NIO functionalities address the challenges that may emerge when NISs are deployed across different network domains and operating in multiple timescales, including conflict resolution and knowledge sharing among NIS.

3.5.2.1 Conflict Resolution

We introduced two specific conflict cases in Section 3.3.3: (i) when conflicts emerge when monitoring data, e.g., algorithms may need data from the same source but with different granularity, and (ii) when conflicts in the policy enforcement of different NI algorithms may act on the same network functions but configuring different values for the target parameters. In such situations, the policy interpreter and configuration block will gather information about the policy guiding the different NISs and pass their interpretation to the conflict detection and resolution module. In both cases, a conflict will be detected, and the NIO will identify and apply the conflict resolution rules associated with (i) multi-timescale coordination and (ii) parameter constraints and execution priority. After applying the rules, the outcome should provide a plan to trigger a configuration modification of the NIS policies. In the case of NIS empowered by black-box ML algorithms, the Model Explainability block will interpret policies associated with such algorithms.

Figure 3.11 shows the main steps for the case of NIS Conflict Resolution. This procedure includes checking the parameters of the NIS against the Policy Interpreter and Configuration (PolicyIC) to arbitrate the deployment of the NIS (e.g., if the NIS has different monitoring granularity in a Source shared with other NIS, or requires controlling a NIF that another NIS is already controlling with a different AI algorithm). When the NIO receives a request for the instantiation or updating a NIS, it will be the NIS Creation Selection Optimization and Instantiation (CSOI) component that will internally validate

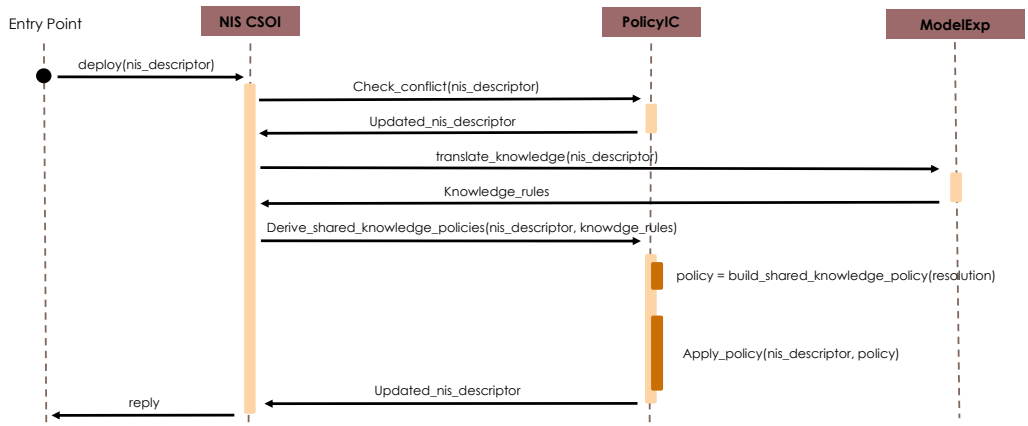


Figure 3.12: Knowledge Sharing process flow.

the NIS, indicating if there is a conflict and updating and resolving the NIS in case any conflict exists.

The validation command executed in the NIO when creating, instantiating, and updating a NIS includes the following steps internally. First, the NIS CSOI validates the NISD. This includes checking the comprising NIFs, identifying the N-MAPE-K types included in the NIFs, and the possible configuration actions that the NIFs can generate. If the NIS request is correct and sound, the NIS CSOI verifies through the PolicyIC if there is any conflict by gathering information about the policy guiding the different NISs and passing their interpretation to the Conflict Resolution component. Then, the Conflict Resolution component checks if the NISs to be deployed have conflicts with the existing NISs. This is achieved through predetermined conflict resolution policies, included in the NISD. The Conflict Resolution component globally solves trade-offs that may emerge from conflicting objectives in the control and user planes, e.g., establishing policies (at small timescales) versus enforcing such policies (at large timescales) [105]. For the case of conflicting NISs, the Conflict Resolution component compares policies among different NIS to detect conflicts that may appear with the new/ updated NIS. It performs conflict resolution based on comparison and resolution rules, providing a NIS configuration. This configuration will result from a trade-off or priority mechanism that the Conflict Resolution component will execute to harmonize the NISs' coexistence. The resolution will contain the last valid configuration if no feasible solution exists. Once the PolicyIC receives the resolution, the new policy is built and applied to the specific NIS. Then, the PolicyIC returns the NISD to the NIS CSOI. Consequently, the NIS CSOI further proceeds with the required NIS operation (i.e., creation, deployment, update, etc.). Eventually, the NIO acknowledges the NIS deployment to the sender.

3.5.2.2 Knowledge Sharing

In this context, knowledge refers to the data, policies, analysis, and decisions that the NISs have acquired through the learning or tuning process. NISs deployed in the same or across different domains use their knowledge to derive their execution plans. The knowledge management block will allow the NIO to understand the knowledge of each

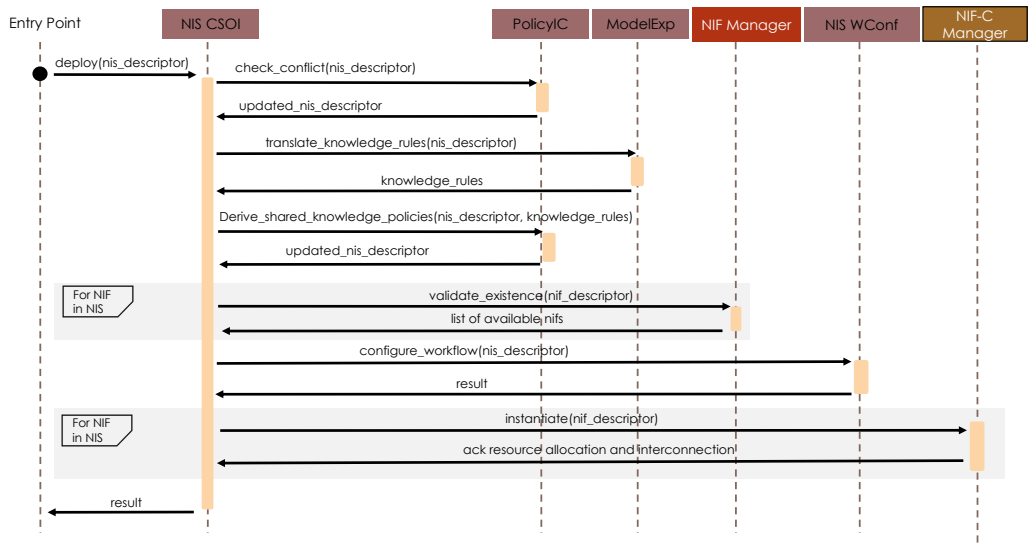


Figure 3.13: Intra NIO instantiation and deployment process flow.

NIS via the interaction with the Model Explainability block. Thanks to the knowledge of each NIS being shared, the NIO could potentially derive newer, global policies that guarantee E2E stability.

Figure 3.12 shows the main steps for the case of NIS Knowledge Sharing. This procedure includes checking the parameters of the NIS against the PolicyIC initially (and consequently also with the Conflict Resolution component internally). However, for the cases in which a NIS requires the use of knowledge coming from another domain, the NIS CSOI will first translate such knowledge in the Model Explainability block before building and applying the policies created using the shared knowledge [106]. This translation is required since the context where the knowledge is initially created is not the same as the one where the knowledge will be applied, e.g., knowledge from the RAN that affects and possibly be used in the transport for queue management.

When a new NIS instantiation is requested, the NIO will process this request similarly as described in the previous sections, i.e., validation, conflict resolution, and policy update. With the updated NISD, the NIS CSOI requests the translation of the external domain knowledge to the Model Explainability block. As a result, the NIS CSOI receives additional Knowledge rules. Then, the NIS CSOI sends the NISD again to the PolicyIC, but this time together with Knowledge rules to build and apply the shared knowledge policies. If shared knowledge policies must be built, this is done by the PolicyIC module, taking into account possible existing conflicts. When shared knowledge policies must be applied, this is also done by the PolicyIC block by returning the NISD to the NIS CSOI. Finally, the NIO acknowledges the NIS deployment to the sender.

Table 3.2: Summary of the procedures proposed to address the challenges described in Section 3.3 and the functionalities of the NIO that can be used to achieve it.

Procedure	Procedure type	Functional blocks
Creation	Inter NIO Procedures	NIO, NIS Catalog, ML Pipelines, NIF Manager
Instantiation or Deployment	Inter NIO Procedures	NIO, ML Pipelines, NIF Manager, NIF-C Manager
Management	Inter NIO Procedures	NIO, NIS Catalog, ML Pipelines, NIF Manager, MANO
Termination	Inter NIO Procedures	NIO, NIF Manager, NIF-C Manager
Other operations	Inter NIO Procedure	As in NFV-MANO
Conflict Resolution	Intra NIO Procedures	Policy Interpreter and Configuration, Creation Selection Optimization and Instantiation, Conflict Detection and Resolution
Knowledge Sharing	Intra NIO Procedures	Policy Interpreter and Configuration, Model Explainability, Knowledge management
Intra NIO Instantiation and deployment	Intra NIO Procedures	Policy Interpreter and Configuration, Model Explainability, ML Pipelines, NIF Manager, NIF-C Manager

3.5.2.3 Intra NIO Instantiation and Deployment

The previously described Conflict Resolution and Knowledge Sharing mechanisms are inherent to the NIS CSOI and Lifecycle Management components interacting with external components such as the NIS Catalog, the NIF Manager, or the NIF-C Manager. To illustrate how the external processes would occur inside the NIO, Figure 3.13 details the deployment interactions between the NIS CSOI with both internal and external components. The procedure includes the steps to validate the NISD and identify and solve possible conflicts before deployment. Also, domain-specific policies are built and applied, followed by training new models in case there are no instances of them already in the catalog. Finally, the NIS Workflow Configuration block combines them to build the NIS and start the instantiation and deployment.

The detailed sequence of steps is described as follows. First, a request for deploying a NIS is submitted from the sender (it could also be a NIS update). The sender identifies that a new NIS needs to be deployed and submits its request to the NIO through the NIO API. Then, the NIS CSOI component receives the request. The NIS CSOI processes the NISD, including, but not limited to:

- Checking for mandatory elements (i.e., network operation, data requirements, output format, accuracy).
- Validating the integrity and authenticity of the descriptor.

If the NIS request is correct and sound, the NIS CSOI will proceed with the validation of the NIS. Please note the validation command executed in the NIO for the creation, instantiation, and update processes described in Section 3.4.2 also includes the procedures of Conflict Resolution and Knowledge Sharing. Then, the NIS CSOI verifies against the PolicyIC if there is still any conflict. As described above, the PolicyIC internally requests the Conflict Resolution component to check further if the NIS has any conflict with existing NIS. Once the PolicyIC receives the resolution, the new NIS domain-specific policies are built and applied, and an updated NISD is returned to the NIS CSOI.

With the updated NISD, the NIS CSOI requests the translation of the external domain

knowledge to the Model Explainability block. As a result, the NIS CSOI receives the additional Knowledge rules. Later on, the NIS CSOI sends the NIS descriptor again to the PolicyIC, but this time together with Knowledge rules. Hence, shared knowledge policies are built and applied as previously described in the above sections. The NIS CSOI now iterates for every NIF in the NIS and checks if an instance of the given NIF already exists in the NIF Manager. If no instance exists, a new model will be trained for that NIF in the MLOps Pipeline.

Next, the CSOI proceeds with the interconnection of all NIFs in the NIS. This mechanism involves requesting the NIS Workflow Configuration block to virtually link the NIFs and define their interactions. Finally, the NIS CSOI starts the instantiation of every NIF in the NIS in the NIF-C Manager. As previously described, this involves checking the resource availability for that NIF in the infrastructure by the NIF-C Manager. If resources are available, allocate the resources for that NIF, and interconnect with the other NIFs instances through the NIF-C Manager. If resources are not available, notify the sender accordingly. Eventually, the NIO acknowledges the NIS deployment to the sender.

Table 3.2 summarizes the functionalities proposed in Section 3.3 for the NIO and which procedures use them. Notice that the procedures described in the previous sections are also based on the related challenges in Section 3.3. However, further procedures can be defined based on other use cases, e.g., orchestration of NI in federated domains or intelligent orchestration of NISs, where the decisions of the NIO are empowered by AI-based decision-making models). We expect that these procedures can be further extended to more complex cases or used as a reference to define new ones.

3.6 Conclusion

This chapter presents the architectural design of the NI Stratum, an evolution of a NI plane, and an end-to-end orchestrator for NI. This design serves multiple purposes: it supports closed-loop NI across the entire network infrastructure, enables coordination of NI instances to exploit synergies and manage conflicts, and defines necessary interfaces for NI algorithms to interact with local environments. This architectural approach is pivotal in structuring NI more effectively and is instrumental in enhancing the efficiency and effectiveness of NI deployment in network infrastructures.

We build upon the MAPE-K framework on defining NI, highlighting the importance of data and the mechanisms required for coordinating NI instances. We introduced a comprehensive workflow for managing the NI lifecycle. This includes detailed procedures for resolving conflicts among multiple NI instances and strategies for knowledge sharing. The methodology outlined in this chapter is an important step forward in the field, addressing key challenges in NI coordination and conflict resolution.

The tools, concepts, and procedures discussed in this chapter will be directly applied in the subsequent technical chapters. Specifically, Chapter 4 introduces a NI solution for managing interference in wireless networks, while Chapter 5 addresses the scaling challenges in NFV networks using Deep Reinforcement Learning (DRL). Although ML-based approaches for these problems are well-documented in the literature, our proposed NI solutions leverage the N-MAPE-K framework. This decomposition facilitates component

reuse and ensures alignment with emerging standardization efforts.

Additionally, Chapter 6 explores a novel business model involving multiple NIF developers. A key challenge in orchestrating such NI solutions arises from the varying designs and objectives of these developers, making it difficult to select the optimal NIF for deployment. This scenario extends beyond the scope of current MLOps frameworks, which primarily focus on evaluating individual models.

Chapter 4

Network Intelligence for Interference Management in Wireless Networks

The content of this chapter has been partially published in:

[107] **P. Soto**, M. Camelo, J. Fontaine, M. Girmay, A. Shahid, V. Maglogiannis, E. De Poorter, I. Moerman, J. F. Botero, and S. Latré, "Augmented Wi-Fi: An AI-based Wi-Fi management framework for Wi-Fi/LTE coexistence," in IEEE International Conference on Network and Service Management (CNSM). IEEE, 2020, pp. 1–9.

[108] **P. Soto**, M. Camelo, K. Mets, F. Wilhelmi, D. Góez, L. A. Fletscher, N. Gaviria, P. Hellinckx, J. F. Botero, and S. Latré, "ATARI: A graph convolutional neural network approach for performance prediction in next-generation WLANs," Sensors, vol. 21, no. 13, p. 4321, 2021 [IF 3.847]

Wireless networks operating in the unlicensed spectrum mainly rely on IEEE 802.11 (Wi-Fi) to provide high performance with a high density of connected devices to support emerging demanding services, such as virtual and augmented reality. However, two main issues impede Wi-Fi to operate correctly. First, many Wi-Fi networks are being deployed to provide coverage to hundreds of users in limited indoor areas such as residential buildings, offices, and malls. Second, other wireless technologies, such as Long-Term Evolution (LTE), could operate in these bands as a way to offload the ever-increasing mobile traffic demand. Both situations overcrowd the unlicensed spectrum and compromise Wi-Fi performance. This chapter presents two Network Intelligence (NI) approaches to handle the interference generated by the same technology, as in overcrowd Wi-Fi deployments, but also by different technologies, as in the LTE case. Moreover, using the concepts presented in Chapter 3, the NI approaches are decomposed using the Network Monitor-Analyze-Plan-Execute over a shared Knowledge (N-MAPE-K) approach. Notice that the intelligence in both solutions is focused on *analyzing* the input data. However, there is a key difference in the contributions. In the first solution, the development of the *plan* function is left for future work, making the development of the NI the main contribution of this dissertation. In contrast, the second solution builds upon an existing NI [109] and proposes a control algorithm for Wi-Fi. These two contributions demonstrate the effectiveness of NI in enhancing Wi-Fi performance in challenging co-existence scenarios and dynamic dense deployments. Specifically, this chapter aims to solve the following Research Questions (RQs):

[RQ_3] **How can Machine Learning (ML) algorithms be tailored to incorporate network-specifics (e.g., topological information) in their learning objectives or adapting already gained knowledge from the ML field to networking to enhance their effectiveness in network management tasks such as interference management or resource allocation?**

[RQ_5] **How can data-driven approaches be developed for real-time, context-aware optimization of network control parameters in dense network deployments to effectively manage interference, optimize spectrum use, and ensure coexistence and high performance of various wireless devices in unlicensed bands?**

4.1 Context and Problem Statement

Wi-Fi has been the preferred choice in the unlicensed spectrum (2.4 and 5 GHz) as a broadband access technology due to its low deployment cost and decent performance. Consequently, the global number of Wi-Fi hotspots (including home spots) increased in the last years, providing connectivity to 16.2 billion devices with approximately 3.6 networked devices per capita [110]. Additionally, Machine-to-Machine (M2M) communications account for 50% of those networked devices, which represents an increase of 17% compared to 2018. This means that not only in residential settings, Wi-Fi is one of the preferred technologies that give internet access to myriad networked devices to support bandwidth-hungry services, such as augmented and virtual reality, online gaming, and video streaming.

Despite the continuous growth and reliance on Wi-Fi, the allocation of unlicensed spectrum has not kept up with its expansion. In addition to Wi-Fi's popularity, recent LTE versions propose to use unlicensed bands as a traffic offloading solution to cope with the increasing mobile traffic demand [34]. This situation adds more stress to the already scarce spectrum as more devices will fight for medium access. To maintain Wi-Fi's economic and societal contributions, there's a need for a globally harmonized spectrum that accommodates Wi-Fi's growth and ensures equitable coexistence with other unlicensed spectrum technologies.

On the one hand, Wi-Fi itself has been evolving as network-oriented applications are increasingly more stringent in their Quality of Service (QoS) and Quality of Experience (QoE) requirements. Future Wi-Fi generations are proposing a set of features from which Channel Bonding (CB) is highlighted. CB is a technique introduced in IEEE 802.11n that enhances the channel capacity by bonding a maximum of two primary channels in a given transmission. Newer Wi-Fi versions allow even more primary channels to be bonded to increase bandwidth [58]. However, strategies for CB might be counterproductive in highly dense scenarios as end-user Stations (STAs) might overlap. As a result, the STAs may experience a high number of collisions and suffer excessive starvation [111]. This situation can cause drastic performance deterioration and increase the packet error rate.

On the other hand, it has been studied that the deployment of LTE in the unlicensed spectrum can negatively influence Wi-Fi performance. In this scenario, most of the works propose LTE mechanisms that adapt to the wireless environment, protecting Wi-Fi communications. Such mechanisms include Transmission Opportunities (Tx-Opps)

during muting periods in a duty-cycled manner (Long-Term Evolution Unlicensed (LTE-U) [112]), channel assessment procedure before a transmission (Long-Term Evolution Licensed Assisted Access (LTE-LAA) [113]), and standalone operation in unlicensed bands (MuLTEfire [114]). However, such LTE mechanisms are asymmetric in terms of medium access. LTE uses significant long times to transmit and short muting periods, typically of several ms, compared to a standard Wi-Fi transmission that lasts a few hundreds of µs [115] without using frame aggregation.

Hence, developing intelligent spectrum management strategies is crucial for meeting the QoS demands of all connected devices. These strategies need to address both intra-technology interference from devices using the same technology and inter-technology interference from devices using different technologies. In this chapter, we introduce two NI approaches to managing interference in wireless networks, focusing primarily on Wi-Fi. Inspired by specific use cases, one for each type of interference, we employ the N-MAPE-K framework to modularly design the NIs, assigning intelligence functions to the *Monitor*, *Analyze*, or *Plan* components accordingly.

The remainder of this chapter is structured as follows. An overview of the related work is presented in Section 4.2, where two different subsections are written, one for each use case. Section 4.3 discusses the first use case, the NI for intra-technology interference, where a novel approach for predicting the throughput in next-generation Wireless Local Area Networks (WLANs) is presented as the main contribution (c.f., Section 4.3.1.1). Accordingly, Section 4.4 discusses the second use case, the NI for inter-technology interference, where a Wi-Fi management framework is proposed to improve the Wi-Fi/LTE coexistence in unlicensed bands. Finally, Section 4.5 presents the conclusions.

4.2 Existing Solutions to the Interference Management Problem in Wireless Networks

This section reviews the relevant literature for our explored use cases, divided into two focused subsections. Given the broad scope of "interference management in wireless networks," we narrow our review to the works that are relevant to our main contributions. Specifically, we examine network models predicting wireless link throughput under Wi-Fi environments in Section 4.2.1 and explore the Wi-Fi/LTE coexistence through ML approaches in Section 4.2.2. Our goal is to position our research within the existing scholarly dialogue, pinpointing unaddressed areas our study addresses, as detailed at the end each subsection.

4.2.1 Intra-technology Interference Management in Wi-Fi Networks

Several works have shown that using the same set of bonded channels in each transmission might be counterproductive, as neighboring nodes might interfere and prevent other nodes from transmitting in a given channel (or set of channels) [111]. These works also showed that the system performance could improve if dynamic CB policies can be applied based on current spectrum usage.

Under dynamic CB, the node will transmit over the most extensive contiguous set of channels sensed to be idle right before transmission. In that way, a node can adopt several channel configurations, depending on how many channels are free on a per-packet basis. An evaluation of different channel configurations is presented in [116]. Through simulations, the authors found that CB does not represent a significant increase in the overall throughput when considering a high amount of contending nodes. Instead, better performance is achieved when transmitting over single channels.

CB benefits from throughput prediction models, given that different channel configurations can be evaluated before being applied. The best configuration can be selected to increase the overall WLAN performance. Typically, Markov chains have been used as throughput prediction models. For instance, a continuous-time Markov-chain was proposed in [117] to evaluate the performance in fully overlapping deployments. Using this Markov chain, the authors could calculate the throughput achieved by an Access Point (AP) and its associated STAs and explain some fundamental properties in dynamic CB, such as their sensitivity to the backoff and transmission time distributions. Similarly, in [61], the authors considered partially overlapping deployments and proposed two more CB policies, including a probabilistic selection of channels. The obtained results showed that adaptive policies are needed that benefit from the information they can gather from the medium.

Learning models were also proposed for Wi-Fi performance prediction. In [118, 119], classical ML techniques (e.g., shallow Feed Forward Neural Network (FNN), Support Vector Machine (SVM), Random Forest (RF), Gradient Boost (GB), among others) are used to predict the throughput in Wi-Fi networks in varying environmental and network conditions. Authors clearly state the importance of the set of input features in ML models, as in [118], their best results were obtained with an FNN at the expense of a linearly increasing feature set and computation time with the number of STAs in the network; in [119], the authors found that using features such as Received Signal Strength Indicator (RSSI) are only meaningful for prediction in coarse time-scales (e.g., typically seconds).

Different CB policies can be evaluated through Reinforcement Learning (RL), where APs learn what channels they need to bond depending on the expected throughput. Therefore, RL approaches need to evaluate the impact of the chosen configuration. For instance, in [120], based on a simple interference model, the authors proposed a Deep Q-Network (DQN) that learns how many channels need to be bonded to satisfy future demands. Here, the authors assumed that different WLANs are connected to a central controller, allowing information exchange. The results showed that the algorithm learns to avoid interference, given that the most-used channel configurations were the ones where most neighboring APs did not overlap with each other.

As can be seen, different solutions have been proposed for modeling Wi-Fi performance. Most of them are mathematical models based on simple assumptions that lower their complexity and do not fully capture the wireless interactions. Some learning models have also been proposed for the same purpose, and the importance of the input features in the prediction accuracy has been clearly stated. The key differences between our work and previous works are summarized as follows:

- Instead of defining an analytical model that does not scale when the number of

nodes in the environment increases, we propose several ML approaches that learn from data. The proposed approaches take care of the complex task of defining wireless interactions.

- Unlike [118, 119], we focus on highly dense scenarios, where a variable number of nodes are sparse over a given area. Our scenarios range from 40 to 240 nodes per deployment in six different configurations of CB. We show these scenarios in Section 4.3.2.2.
- Unlike [120], our approach does not define the best CB policy that a WLAN needs to apply to maximize its reward. Our goal is that the ML model learns how the wireless interactions affect the throughput of a given WLAN without any explicit assumptions, such as user demand or user mobility patterns. This approach can later be used as a function approximator in such RL approaches.

4.2.2 Inter-technology Interference Management from the Wi-Fi/LTE Coexistence Perspective

Given the design differences between Wi-Fi and LTE, several studies [121, 122, 123] verify through simulations and mathematical analysis that the performance of Wi-Fi can be severely affected by the standard operation of LTE in the unlicensed spectrum. Therefore, efficient mechanisms are needed to guarantee coexistence between the two competing technologies by fairly sharing the spectral resources.

In the last years, only a few works have provided experimental results of LTE's deployment negative impact on Wi-Fi transmissions. The authors in [124] proposed an architecture based on Software Defined Networking (SDN) for inter-network cooperation on radio resource management to facilitate coexistence. An analytical model was developed to evaluate Wi-Fi and LTE's performance, and its validity was tested through experiments using real hardware. Afterward, the framework was used to exchange information between the co-located networks to jointly optimize the transmission power of the APs for fair spectrum access. Capretti *et al.* [125] deployed a small testbed using Software Defined Radios (SDRs) and open-source Wi-Fi hardware. They used different configurations of the two technologies, such as channel bandwidth, Modulation and Coding Scheme (MCS), packet size, and transmission power. They showed that coexistence could be achieved if LTE employs duty-cycling.

More focused on Artificial Intelligence (AI)/ML-based approaches for improving coexistence, authors in [126] presented DM-CAST, a Q-Learning algorithm to adjust the parameters of a Duty-Cycling LTE to guarantee a fair coexistence with Wi-Fi. The solution was implemented using simulations in ns-3, where they considered different traffic load scenarios. The obtained results showed that the algorithm efficiently adapts to changing conditions while maintaining aggregated throughput. Maglogiannis et al. [127] proposed mLTE-U, a novel adaptation mechanism that determines LTE transmissions in the unlicensed spectrum for a variable Tx-Opp followed by a variable muting period. A Q-Learning scheme was implemented for auto-tuning the Tx-Opp and muting periods of mLTE-U for fair coexistence with co-located Wi-Fi. The authors showed that the proposed algorithm adapts well to the changes in a dynamic wireless environment through simulations.

Authors in [128] trained a Convolutional Neural Network (CNN) using Commercial Off-The-Shelf (COTS) hardware to detect co-located LTE and Wi-Fi transmissions. Its results aid in selecting the appropriate mLTE-U configurations that enhance fair coexistence. In that way, Wi-Fi can freely transmit during the muting periods of LTE. The proposal was validated in an isolated real deployment. We refer the reader to [129] for more details on the use of ML in mobile and wireless networks and to [130] for more details on wireless inter-technology coexistence.

Previous literature has focused on providing LTE with the necessary means to adapt its internal settings to the spectrum's shared use in unlicensed bands. However, few studies tackle the coexistence problem by considering Wi-Fi adaptation to changing conditions. This is also founded on design differences. Typically, Wi-Fi is designed to share the spectrum with other Wi-Fi networks using the Carrier-sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism. Meanwhile, LTE is considered to operate in the licensed spectrum without more players involved, and therefore, significant modifications to LTE have to be implemented to support coexistence in unlicensed bands. Moreover, a minor effort has been made to prove the proposed models' validity in real-life deployments. Key differences with previous works are summarized as follows:

- Unlike [121, 122, 123], our approach's validity is tested using COTS Wi-Fi and LTE hardware.
- Although [124] and [125] validated their proposals on hardware, multiple Wi-Fi parameters such as the frame or packet size, the AP's transmission power, network load, and MCS were manually varied to investigate different operation scenarios. Such parameters are not easy to modify under real conditions on legacy devices.
- Authors in [126, 127, 128] employed several ML techniques to help the LTE decision-making algorithm to give more Tx-Opps to Wi-Fi. Contrary to these works, our proposal implements a ML technique to help Wi-Fi adapt to changing conditions by providing information on the spectrum's use.

4.3 Network Intelligence for Intra-Technology Interference Management in Next-Generation WLANs using Channel Bonding

This use case considers a highly dense wireless environment, where multiple APs and STAs transmit using Wi-Fi with CB support. CB is one of the features [131] introduced since the 802.11n Wi-Fi amendment [58, 132] to cope with the stringent demands from next-generation services. Consequently, a WLAN is composed of M nodes, one AP, and $M-1$ STAs (i.e., end-devices) associated with that AP. Multiple APs have to be deployed in an indoor location to provide connectivity to all the STAs in the location. Then, every AP is assigned a contiguous subset of channels from F channels of equal bandwidth, according to a transmission policy.

Assuming that the coexisting WLANs can transmit using Dynamic Channel Bonding (DCB), if the network conditions allow it, each AP uses the largest subset of available

transmission channels allowed by its policy and depending on the wireless environment. The more channels it can bond, the higher the degree of freedom in selecting channel configurations. In this way, the AP tries to increase the overall throughput. Given the high number of WLAN deployed in crowded deployments and the combinatorial nature of CB, it is not always trivial to select the right channel configuration.

Consequently, a careful selection of the transmission channels is needed since selecting the widest subset of available channels can be harmful in the overall long-term throughput and fairness among WLANs [61]. Next-generation WLANs will be able to assess the impact of a given configuration before every transmission takes place through performance prediction models. In dense environments, multiple interactions among devices occur in a matter of microseconds. These interactions vary on a per-packet basis, making each transmission unique. Moreover, the wireless channel capacity where that transmission occurs depends on the channel effects, such as noise and interference, and the number of bonded channels. Therefore, figures such as Signal-to-Interference plus Noise Ratio (SINR), RSSI, bandwidth, MCS, and the amount of time a node can transmit are crucial to model the wireless environment.

4.3.1 Framework Architecture

Intelligent next-generation WLANs can tackle intra-technology interference, like the one presented in DCB, by using interference management applications in a SDN framework, resulting in a Software Defined Wireless Local Area Network (SD-WLAN). This approach is compatible with the International Telecommunication Union (ITU) framework for ML in future networks, including International Mobile Telecommunications (IMT)-2020 [133], as shown in Figure 4.1. This framework comprises several steps; first, different measurements are constantly collected via the collector (C) block. These measurements might come from the terminals connected to the WLAN, i.e., STAs, but also from the WLAN's data plane (e.g., transmission capabilities and sensed interference from the APs). They also constitute the source of data (SRC in Figure 4.1).

The collected measurements are later aggregated in the pre-producer (PP) block and compose the network state. To achieve this task, ML approaches are suggested. In this step, the ML method builds an accurate network model (M) that allows the interference management application to predict/estimate the performance (e.g., throughput) of the WLAN. Afterward, the model produces the expected WLAN performance under the given circumstances, which are then used on the controller side to create channel configuration policies (P). The decisions from the controller are distributed (D) to the different APs through an appropriate Application Programming Interface (API), so they configure the channels (SINK) according to the controller's decisions.

The N-MAPE-K framework detailed in Chapter 3 closely matches the ITU framework [133], where *Sensor* nodes equate to source nodes, and both the collector and pre-producer align with the *Analyze* block. Similarly, the policy aligns with the *Plan* block, the distributor with the *Execute* block, and sink nodes with *Effector* nodes. The intelligence in our intra-technology interference management system is concentrated in the *Analyze* block, introducing a unique method using Graph Neural Networks (GNNs). GNNs are Neural Networks (NNs) designed for graph-structured data, leveraging node relationships within the graph. This section outlines our GNN model to forecast WLAN performance,

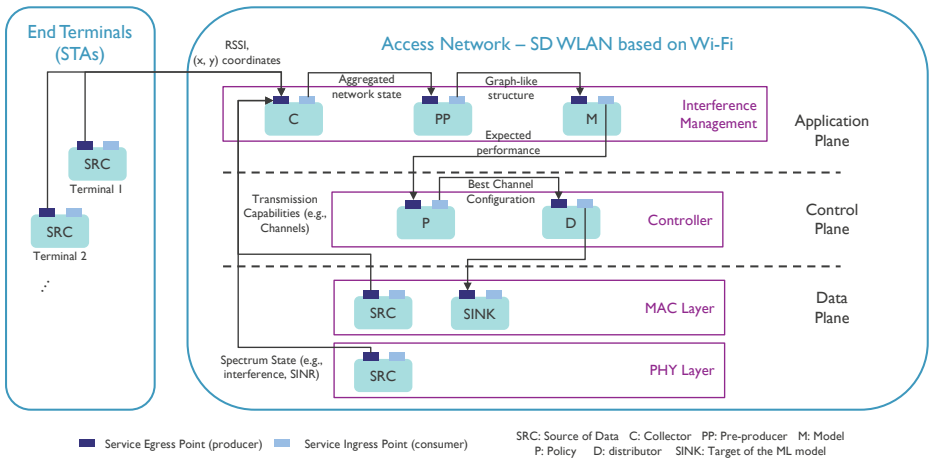


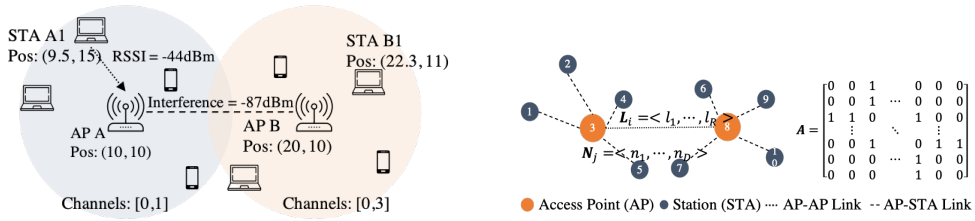
Figure 4.1: Reference architecture for our NI for intra-technology interference management

pivotal to our interference management framework.

4.3.1.1 A GNN Model for Performance Prediction in Next-Generation WLANs

In several applications, data can be defined by graphs where the relationships among samples (or nodes) are represented by the edges between them. Networks are a clear example of such graphs; nodes have their own properties or features and relate to other nodes via the links between them. If there is no link between two nodes, we can generally say that those nodes are not related to each other. Despite their success, ML faces challenges when learning from structured data represented as graphs, as the relationships among nodes are not captured or have to be represented differently. For example, typical data pre-processing steps include the generation of a fixed-size matrix (e.g., images of a given size, audio samples or sentences of a fixed length), which represents the sample's information and serves as training data.

Figure 4.2a shows a specific WLAN deployment of two Basic Service Sets (BSSs) (A and B). It consists of two APs, located at the center of the cell, and four STAs each, distributed around the coverage area of the APs (blue and orange circles, respectively). Information such as the position (x-y coordinate), channel configuration and the type of device, among others, can be used as features in a data set. To predict WLAN performance using a ML approach, a data set is created, using the aforementioned information, i.e., a vector represents the features of a device while a matrix represents a WLAN or a BSSs. Unfortunately, all deployments are different in terms of topology but also in terms of its size. By having a variable number of APs and/or associated STAs, the size of the matrix will also change, violating the requirement of ML of a fixed-size input. A ML practitioner could solve this issue by fixing the matrix size to the biggest deployment size on the data set and performing re-scaling or data padding on smaller deployments. Data padding can be seen as the introduction of new devices that do not alter the transmission pattern of the BSS, which is not totally true. Moreover, neural networks are usually trained in a fixed-sized batch-wise fashion where the size of the batch is determined by



(a) Topological view.

(b) Graph representation.

Figure 4.2: Example of a WLAN deployment. On the left, the topological view of the deployment, some measurements are included. On the right, the graph representation of the deployment with a node feature $\mathbf{N}_j$, an edge feature $\mathbf{L}_i$ and its adjacency matrix $\mathbf{A}$.

the capabilities of the computing machine. Normally, the bigger the size of the batch, the more computing power the machine should have. Typical values of batch size are 32 or 64 samples. This means that only 32 or 64 devices are considered per training batch, regardless of whether they belong to the same deployment or not.

On the contrary, approaches such as GNNs [74] have been proposed as NN that operate on graphs. Mapping the GNN terminology to our specific example of Figure 4.2a, each deployment is considered as an undirected graph where STAs and APs are the graph's nodes. Position, channel configuration, and device type are considered to be the node features (more details in Section 4.3.2.2). The graph's edges are established, according to the wireless interactions between nodes, e.g., the RSSI received by the AP and the interference levels between BSSs. In this case, Figure 4.2b shows the resulting graph of the deployment shown in Figure 4.2a. The graph has ten nodes (five per BSS) and nine wireless links, four per BSS plus one between APs. Once the data are formatted into a graph, there are several GNN frameworks (e.g., [134]) that operate on the node features, edge features, and the adjacency matrices by creating sparse matrices.

For the sake of the argument, let us assume two WLAN deployments. The first deployment is shown in Figure 4.2 (called deployment a), while the second deployment (called deployment b) is an exact copy of the first. The adjacency matrix of deployment a, the node features, and the target (throughput) are represented by matrix $\mathbf{A}_a$, matrix $\mathbf{X}_a$ and matrix $\mathbf{Y}_a$, respectively. The same is valid for deployment b. Then, the GNN framework creates a new adjacency matrix, a new node feature matrix, and a new target matrix as the concatenation of the two (a and b) around the node dimension as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_a & \\ & \mathbf{A}_b \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} \mathbf{X}_a \\ \mathbf{X}_b \end{bmatrix}, \qquad \mathbf{Y} = \begin{bmatrix} \mathbf{Y}_a \\ \mathbf{Y}_b \end{bmatrix}.$$

In this way, the operations (e.g., message passing) among two nodes of different deployments are not possible and eliminate the padding in the node or edge features. Only the adjacency matrices are allowed to be padded, which has no change on the original topology (a zero in the adjacency matrix means that there is no edge between two pair of nodes).

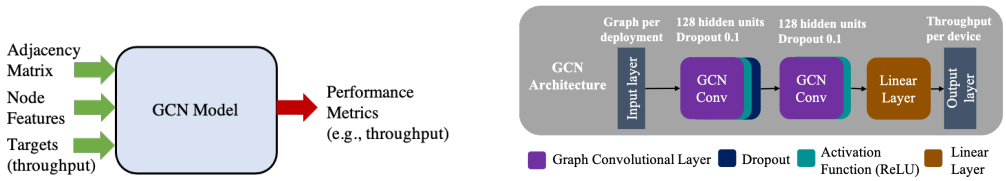
To summarize, we advocate the use of GNNs in WLAN performance prediction, given the following:

- GNNs have been successfully applied to combinatorial optimization problems [135], and can achieve relational reasoning [136].
- CB is a problem with a combinatorial action space in dense deployments, where the complexity increases exponentially with the size of the deployment and the number of possible channel configurations.
- The relationships between STAs and APs (connectivity, interference, among others) can be captured via a graph representation, i.e., there is a one-to-one mapping between the network topology and the graph representation.
- GNNs are also proposed to solve multiple network optimization processes [137], given their ability to generalize to large-scale problems.
- GNNs can easily operate and generalize over environments with a changing topology and a variable number of nodes.

We model a WLAN deployment as shown in Figure 4.2b. The links between STAs and APs represent the connectivity within the BSS through wireless links, while the link between APs represents the interference that BSSs have among themselves. Notice that the same set of links can be used for downlink interference since the GNN model can use bi-directional links. As it can be inferred, APs form a mesh graph, while devices within the BSS form a star-like graph. We define our GNN model using Graph Convolutional Networks (GCNs) [138], a CNN variant operating in graphs. GCN follows a multilayer approach, just like FNN, with the difference that it processes graphs instead of images, time series, or text sequences. The number of Graph Convolutional Layers (GCLs) is typically defined by the diameter of the graph, which tells in how many hops any node is reachable from any other node; so, in this particular case, the graph diameter is three for inter-BSS interaction (two nodes of different BSS are communicating) and two for intra-BSS interaction (two nodes of the same BSS are communicating).

Despite GNNs being able to operate with edge features [136], GCNs do not naturally support them. As a consequence, all available features (detailed in Section 4.3.2.2) are defined as node features. Thus, the node features represent the deployment characteristics, such as node positioning, node type, channel configuration, and interference, among others. The deployments' topology allows us to consider two GCLs to propagate information from STAs to AP and between APs. Additionally, as a third layer, we use a linear function that operates only on the node's features. All GCLs use REctified Linear Unit (ReLU) as an activation function, and only the first GCL uses dropout for regularization purposes. Figure 4.3 shows a summary of our GNN model. Figure 4.3b shows the resulting architecture, with the best values found in a hyper-parameter search for the GCL. Figure 4.3a summarizes what our GCN model does; it takes as input a topology of WLAN deployment, represented in an adjacency matrix, the node features and the targets. It learns the wireless relationships between BSSs and predicts the performance of that WLAN deployment. Thanks to the generalization properties of GNNs, our model is able to predict the performance of unseen deployments, as shown in Section 4.3.2.

As a summary, we provide in Table 4.1 a decomposition of our intra-technology interference management framework. Firstly, different measurements are collected from the network. The collected measurements constitute the network state. Notice that the features might be collected in different time scales, and aggregation or averaging might be



(a) High-level overview.

(b) Model architecture.

Figure 4.3: Summary of our GNN model. A high-level overview is presented on the left, while the model architecture is presented on the right.

Table 4.1: N-MAPE-K representation of the intra-technology interference management App

N-MAPE-K module	**Solution module**
Sensors + Monitor	Link State comprises the signal-to-interference-plus-noise ratio (SINR), and the received signal strength indicator (RSSI). The AP's available channel configurations, distance, associations among users and APs, and users' traffic demands.
Analyze	Inputs are aggregated and converted into a graph-like structure to capture all the topological properties of the WLAN deployment. A GNN predicts the throughput of the WLAN. A network model is produced.
Plan	A controller creates and evaluates channel configuration policies using the GNN. Then, the best configuration is selected to maximize the throughput for all the WLANs.
Execution + Effector	The selection is communicated to the APs, using appropriate APIs.
Knowledge	Monitored data, current setup, best performance prediction, possible channel configurations, and learned channel configuration selection policy.
Training Loss / State – Actions – Rewards	Loss Function: Root Mean-Squared Error.

used. For instance, long-term (e.g., days) metrics, such as device energy consumption, do not require short (e.g., ms) monitoring intervals. Although some features are difficult to get, we expect that introducing SDN in wireless networks will facilitate the data collection campaigns, where the collected data can later be used for intelligent management decisions.

Then, the evaluation of different configurations will be executed in a timely manner if a triggering condition is met, forming a closed control loop; for example, if the controller (see Figure 4.1) perceives that 50% of their managed devices have lower throughput than expected. In this case, the ML model will generate a prediction for every evaluated configuration. Note that a traditional model (e.g., Markov model) takes a long time to generate a prediction, longer than the time required to optimize the spectrum usage to decrease interference and improve the user experience.

Undoubtedly, the role of the controller is vital for network optimization, as it is in charge

of making management decisions. Nonetheless, an accurate and fast network model is also a key element within the architecture. In this section, we addressed the latter and left the controller out of the scope. In particular, we present, train, and evaluate a GNN model that is well suited to the WLAN performance prediction problem, where information is also embedded in the topological representation. We believe that the combination of powerful data-driven models as controllers and predictors enables the vision of self-organized, self-monitored, and self-healing beyond Fifth Generation (5G) networked communication.

4.3.2 Experimental Validation

This section summarizes the experimental validation of our proposed GNN model for throughput prediction in next-generation WLANs. More precisely, Section 4.3.2.1 presents the ML models, which are used to evaluate the validity of our approach. The dataset on which our model is trained and validated is summarized and analyzed in Section 4.3.2.2. Finally, the experimental results are shown in Section 4.3.2.3.

4.3.2.1 State-of-the-Art ML Models as Benchmark for the GNN

Here, we design two Deep Learning (DL)-based models and an ML-based model that serve as a baseline to our GNN model. Figure 4.4 (top and bottom) shows the DL-based model architectures. The input layer of all models receives the characteristics extracted from the deployment data. However, as each deployment does not have the same number of devices, the data have to be processed differently, as these models require a fixed size input. Here, data are pre-processed into a matrix of $N_{DEV} \times D$, where N_{DEV} represents the number of devices in all WLAN deployments and is fed to the models using batches. In turn, the columns (D) represent the device's considered characteristics (e.g., positioning, RSSI and set of bonded channels, among others). At the output layer, the throughput per device is predicted based solely on each device's characteristics. This means that pre-processing does not consider whether a device belongs to a given deployment or not. We do not advocate for using data padding or resizing, as the variability in the deployment size is too big (see Section 4.3.2.2 for further details), and it would be wasteful in terms of computing resources, as most of the devices on a deployment would be missing.

As our GNN uses convolutional layers, we design a CNN that uses two 1D convolutional layers, followed by batch normalization and dropout layers to accelerate training and to reduce over-fitting, respectively. Thereafter, three linear layers followed by dropout layers are added. All layers have ReLU activation functions. Figure 4.4 (top) shows the resulting CNN architecture. The second model is a well-known FNN, shown in Figure 4.4 (bottom). It consists of four linear layers, the input and output layers, and two additional hidden layers. The number of neurons in the layers follows a decreasing configuration, i.e., 32-16-8-1. The rationale behind this configuration is that at each layer, the most representative data features are extracted. Each layer uses ReLU as activation function, followed by a dropout layer to reduce over-fitting. Lastly, the predicted throughput is obtained at the output layer, configuring a regression model.

The last model is based on GB. Boosting algorithms are ensemble methods that minimize

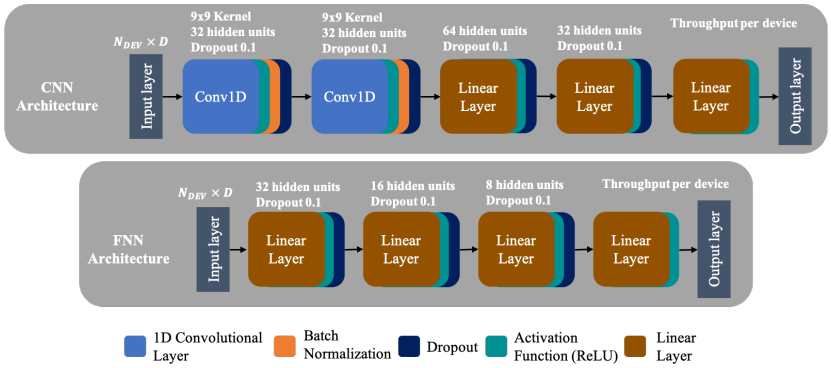


Figure 4.4: Architecture of the state-of-the-art ML models.

the model's loss by adding decision trees, using a gradient descent procedure. In this method, we use default hyper-parameter values, defined by sci-kit learn[1]. The FNN and GB models are typically used in Wi-Fi performance prediction [118, 119].

4.3.2.2 Dataset

Simulated Datasets for Training ML-Based Networking Solutions Typically, currently available Wi-Fi data sets[2] do not include the novel CB feature nor represent dense deployments. Working with data sets with sparsity (e.g., deployments with a limited number of transmitters and receivers) prevents finding insights on next-generation high-density deployments, where the severity of the interference is expected to generate a great impact on the performance. Therefore, to train the models presented in Sections 4.3.1.1 and 4.3.2.1, we used the dataset in [139], which was provided in the context of the 2020 edition of the *ITU AI/ML for 5G Challenge*[3], where our team ATARI was awarded with the 4th position in the Grand Challenge Finale among 33 finalists selected from 911 teams competing in 23 open challenges. This data set includes simulated data from IEEE 802.11 WLAN deployments applying DCB. More specifically, the simulated data set was generated using Komondor [140], an IEEE 802.11ax-oriented open-source network simulator that was conceived as a cost-effective simulation tool for studying the performance of novel features, such as DCB or Spatial Reuse (SR) in dense scenarios. As shown in [141], important features in Komondor, such as the Distributed Coordination Function (DCF) operation and DCB, were validated against ns-3 [142] and the well-known Bianchi and Markov models [143, 144].

Moreover, the simulator was already used in real-world testbeds [24]. In this case, Komondor characterized the WLAN testbed and generated a simulated network twin. An ML approach for transmit power control was trained, using the data generated by Komondor, and then it was applied to the real testbed. The results showed that in using

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html` (accessed on 21 June 2021)

[2] `https://data.europa.eu/data/datasets/jrc-netbravo-netbravo-od-eu-wifi?locale=en` (accessed on 21 June 2021)

[3] `https://www.itu.int/en/ITU-T/AI/challenge/2020/Pages/default.aspx` (accessed on 21 June 2021)

Table 4.2: Summary of data set characteristics.

Data set	Scenarios	Map Size	№ Deployments	Total Devices	APs Per Deployment	STAs Per Deployment	Mean–Std–Min–Max STA Throughput	Mean–Std–Min–Max AP Throughput
Training and Validation	1a	80 × 60 m	100	78,078: 6000 APs 72,078 STAs	12	[10–20]	[6.93–6.99–0-88] Mbps	[83.29–52.24–0-400] Mbps
	1b	70 × 50 m	100		12	[10–20]		
	1c	60 × 40 m	100		12	[10–20]		
	2a	60 × 40 m	100		8	[5–10]		
	2b	50 × 30 m	100		8	[5–10]		
	2c	40 × 20 m	100		8	[5–10]		
Testing	1	80 × 60 m	50	9831: 1400 APs 8431 STAs	4	Random	N/A	N/A
	2	80 × 60 m	50		6	Random		
	3	80 × 60 m	50		8	Random		
	4	80 × 60 m	50		10	Random		

the ML approach, the real deployed WLANs experienced an increase of at least 76% in its throughput.

Unlike the existing measurement campaigns of DCB WLANs (e.g., [145]), Komondor provides synthetic traces depicting different types of deployments, ranging from low to ultra-high density. As shown in [145] through real traces, the 5 GHz band used for DCB in 11ax and 11be amendments is deeply underutilized, even in highly populated areas. Accordingly, network simulators can contribute to filling the data gap in next-generation network deployments and, therefore, provide comprehensive data of interactions among devices in future deployments at which channel occupancy is expected to increase significantly. The broad range of situations captured with large simulated datasets can be very useful to train data-hungry ML models, such as CNNs or FNNs.

The role of network simulators in future 5G/6G networks has been pointed out as a key tool to assist the increasingly adopted ML operation in communications [24]. The fact is that network simulators can represent unknown situations that may not be present in real traces (due to the limitations in acquiring data from real networks), thus allowing to support procedures, such as training, testing, and validation of ML models.

Dataset Generation The data set is divided into two sets, i.e., training and testing. In both cases, enterprise-like scenarios containing a different number of APs and STAs applying DCB are generated, thus depicting multiple situations that could be used for training ML models. The topology of these enterprise-like scenarios is composed of a building floor of a given size (e.g., map size), which is divided into equal-sized offices or cells. The APs' positioning is fixed to the center of the office or cell, while the STAs are randomly placed around the AP's coverage area. Such a topology setting is typically adopted in the simulation scenarios provided by IEEE 802.11 task groups [146]. The data set includes useful information about each deployment, such as the obtained throughput, the RSSI, the airtime in each channel, the interference among devices, or the SINR.

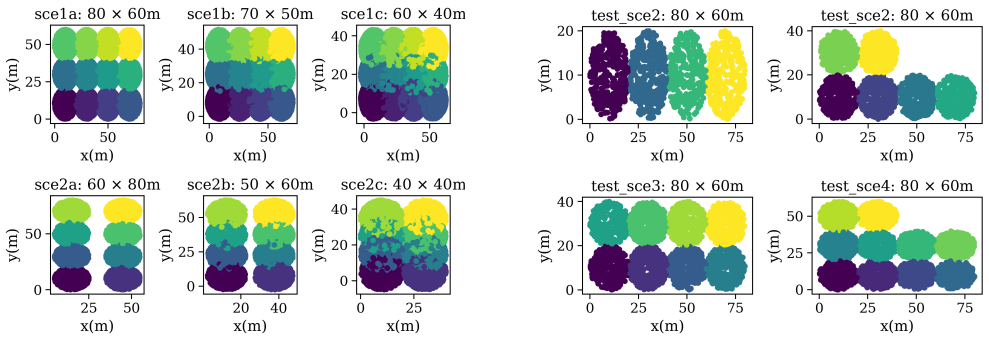
In total, 600 deployments were simulated, containing 78,078 devices (6000 APs and 72,078 STAs). Simulating each CB deployment for training and testing took in the order of tens and hundreds of seconds, depending on the deployment features (e.g., number of nodes, traffic load, and packet losses). Table 4.2 summarizes the main characteristics of the entire data set. Moreover, Table 4.3 details the simulation parameters used for generating the data sets. For more details regarding the parameters used to characterize 11ax frames,

Table 4.3: Simulation parameters used to generate the training and test datasets.

	Parameter	Value	
		Training	***Test***
Depl.	# APs	{8, 12}	{4, 6, 8, 10}
	APs location	Fixed to the center of the cell	
	# STAs	{5–10, 10–20}	5–10
	STAs location	Uniform at random	
	Traffic profile	Downlink UDP	
	Traffic load	Full buffer mode	
	Channel allocation	Uniform at random	
PHY	Central freq.	5 GHz	
	Path-loss model	See [147]	
	Bandwidth	{20, 40, 80, 160} MHz	
	# spatial streams	1	
	Allowed MCS indexes	1–12	
MAC	Contention window	32 (fixed)	
	Data and ACK length	12,000/32 bits	
	RTS and CTC length	160/112 bits	
	Max. A-MPDU	1	
	DCB policy	Dynamic (see [61])	

please refer to [140]. Notice, as well, that 100 and 50 random deployments were simulated for each type of scenario in training and test data sets, respectively. In all the cases, 10 s simulations were considered. (There is a trade-off between the simulation time and the stability of the simulated results. In particular, 10 seconds was shown to properly address this trade-off by providing accurate results at a low execution time in Komondor [24].)

Regarding the DCB configuration, each BSS can use up to $N = 8$ basic non-overlapping channels of 20 MHz in the 5 GHz band. Compliant with the 11ax amendment, a given transmitter can bond channels of width 20 MHz, 40 MHz, 80 MHz, and 160 MHz, thus leading to channelization $C = \{\{1\}, \{2\}, \ldots, \{8\}, \{1-2\}, \{3-4\}, \ldots, \{7-8\}, \{1-4\}, \{5-8\}, \{1-8\}\}$, for basic channels indexed from 1 to 8 (see Figure 2.3b). In each simulated deployment of the data set, both the primary and the total channel width are selected randomly. As for the applied DCB policy, the maximum possible channel width is dynamically used, provided that the involved channels were free during at least the point coordination function interframe space (PIFS) period. For instance, let us assume that a given transmitter has randomly allocated to channels $\{1-4\}$, with primary channel 1. Then, such a device would perform carrier sensing in the primary channel (1) and, provided that the channel was sensed to be free during the backoff, would assess whether the rest of the channels were also found to be free during the



(a) Training scenarios.

(b) Testing scenarios.

Figure 4.5: Spatial distribution per training and testing scenarios. Different colors represent the coverage area of a BSS. APs are located in the center of the circle.

PIFS interval. If only channels $\{1-2\}$ are idle at the moment of starting a transmission, then the transmitter proceeds to use both of them, leaving channels $\{3-4\}$ for future transmissions.

Having multiple random deployments with several APs and STAs, using different DCB configurations, motivates the use of GNNs. The fact is that GNNs exploit the graphs' topological information, independently of how many nodes the graph has, contrary to other types of NNs, where the input data should be fixed in size. Moreover, each scenario introduces a given amount of interference among BSS, as shown in the spatial distribution in Figure 4.5a. For instance, training scenarios *1a* and *2a* only consider inter-STA interference, while training scenarios *1c* and *2c* consider overlapping among BSSs. In the former (e.g., *1a* and *2a*), phenomena such as collisions by hidden-node are prone to occur due to the downlink traffic profile and given the fact that APs do not sense each other. Moreover, for the latter case (e.g., *1c* and *2c*), flow starvation can occur if two or more APs monopolize the channel, thus making other APs contend for the channel excessively. With DCB, this situation may be generated by a chain effect, given that APs select the transmission channels according to their utilization.

Dataset Analysis In the pre-processing step, the relevant data were extracted and appropriately formatted for later processing at the ML approaches. Based on Shannon's theorem of channel capacity, we defined two extra features: the distance between AP and its associated STA; and the bandwidth of an AP, defined as the number of bonded channels multiplied by the bandwidth of an individual channel (20 MHz). We also noticed that the *Primary Channel, Minimum*, and *Maximum Allowed Channels* define a configuration of wireless channels. Therefore, these features are one-hot encoded into one categorical variable (*Channel Configuration*), representing the set of channels that a node uses. We identified that from fifteen possible channel configurations (see Figure 2.3b) , only six channel configurations (from C0 to C5) were used in training and testing data sets. From those, configurations C4 and C5 were mostly used in both data sets. Table 4.4 summarizes the features used during training.

We performed a correlation analysis on the training data set to see which of the defined

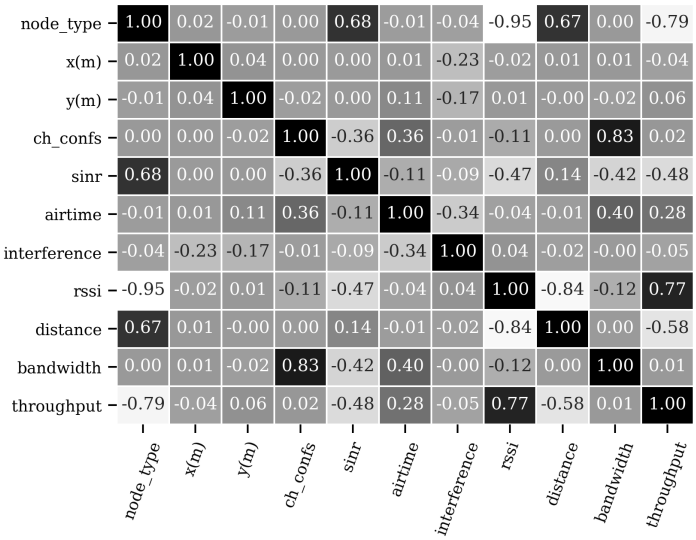
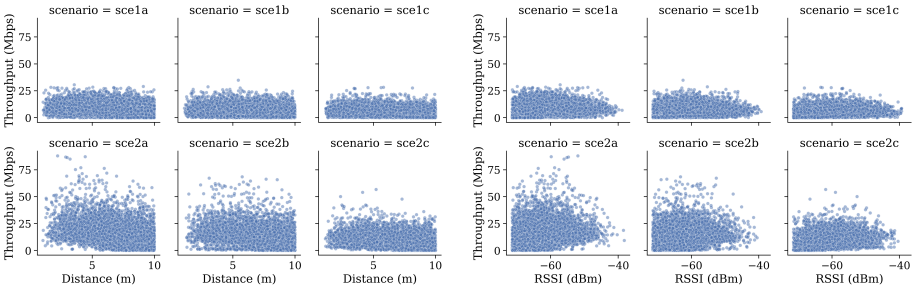


Figure 4.6: Correlation between features and throughput.

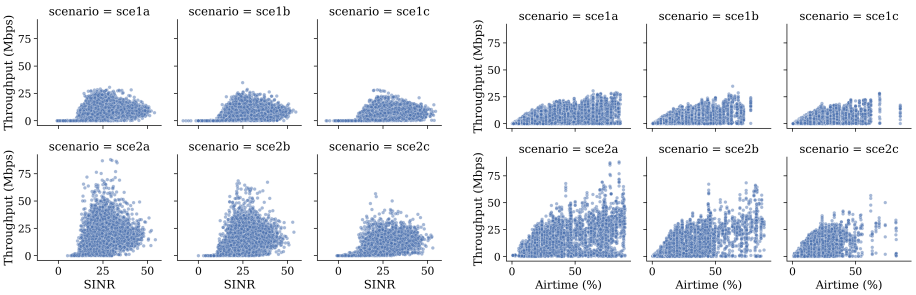
features (independent variables) has more impact on throughput prediction (output variable). Figure 4.6 shows the correlation matrix, using the Pearson correlation. A value close to one (or negative one) shows a strong (negative) correlation, while a value close to zero implies no correlation between variables. It is worth mentioning that this analysis was not carried out on the raw data, which include several undefined values (e.g., RSSI is set to ∞ for APs), but it was already pre-processed and ready to serve as input to the different models. For instance, distance is not correlated with the x-y coordinate, as it is measured from the AP to the STA; hence, the distance of APs is always set to zero. Moreover, due to interference phenomena, it is not clear that RSSI, SINR, and distance keep a linear relationship with the throughput.

As can be seen, features such as node type, SINR, airtime, RSSI, and distance are more relevant for correctly predicting the throughput. Good signal strength is one prerequisite for higher performance, but it is not the only factor determining it. In wireless transmissions, the strength of interfering transmissions is also important in deciding which channels are bonded. Moreover, when contending for a transmission slot, the more airtime is given to an STA, the higher the amount of data it transmits. Hence, there is a positive correlation between the airtime and the throughput. The correlation matrix provides a good starting point for feature selection, as features with correlation values closer to zero can be disregarded. However, if two variables are highly correlated, they may add noise to predicting the output variable, negatively impacting the model's performance.

Figure 4.7 graphically shows the relationship between the most relevant continuous variables and throughput per scenario. It is shown that, in general, scenarios *2a*, *2b*, and *2c* present higher throughput than scenarios *1a*, *1b*, and *1c*, regardless of the value of other variables, showing the problem's severity in highly dense scenarios. Figure 4.7a



(a) Distance. (b) RSSI.



(c) SINR. (d) Airtime.

Figure 4.7: Relationship between throughput and main features per training scenario.

shows that STAs near the APs typically have better throughput in less crowded scenarios. However, this condition does not hold in more crowded scenarios, where the throughput is more or less similar for all the STAs. Regarding the RSSI, Figure 4.7b shows that it is not linearly related to throughput. Therefore, higher RSSI values do not always represent higher throughput. In fact, higher throughput values are found around -60 dBm, which confirms that indeed, RSSI alone is not a good indicator of an adequate link quality [148]. Similarly, in Figure 4.7c, higher throughput values are found at mean values of SINR. A more direct relationship can be found in Figure 4.7d, where generally, higher values of airtime are associated with higher values of throughput.

Based on these conclusions and the correlation matrix, we defined several experiments where different features are used to train the models, and are summarized in Table 4.4. The experiment selection was also motivated because features such as node type and RSSI can be easily gathered in currently available WLANs. By contrast, features such as distance or airtime are more difficult to obtain. For instance, experiment 2 (E2) uses the most relevant features found during data exploration, namely, node type, SINR, airtime, RSSI, and distance. The remaining experiments are defined by taking one of the most relevant features (e.g., SINR in E5) and its uncorrelated features (e.g., node position and interference), while its counterpart (E6) tries to measure the relevant feature's impact by repeating the same experiment (E5) without including it.

Table 4.4: Features used per experiment.

Feature	Definition	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16
Node Type	Wireless node type, AP = 0, STA = 1	✓	✓	✓				✓	✓					✓	✓	✓	
x(m)	x-coordinate of the wireless node	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	
y(m)	y-coordinate of the wireless node	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	
Channel Configuration	Combination of Primary, minimum and maximum channel	✓		✓	✓							✓	✓	✓	✓	✓	
SINR	Signal to Interference plus Noise Ratio	✓	✓			✓										✓	
Airtime	Percentage of time each AP occupies each of the assigned channels (mean)	✓	✓	✓	✓			✓		✓	✓	✓	✓				✓
Interference	Inter-AP interference sensed from every AP (mean)	✓		✓	✓	✓	✓			✓	✓	✓	✓	✓		✓	
RSSI	Received Signal Strength Indicator	✓	✓					✓	✓	✓						✓	
Distance	Euclidean distance between AP and STAs	✓	✓					✓	✓			✓				✓	
Bandwidth	Maximum channel bandwidth	✓		✓	✓							✓	✓	✓	✓	✓	

4.3.2.3 Results and Discussion

This section summarizes the results obtained and compares them with those of the proposed models. We also provide insights into the impact of different features in the models' performance on throughput prediction.

Training and Validation The models described in Sections 4.3.1.1 and 4.3.2.1 were trained on the corresponding data set with a fixed split (80% for training and 20% for validation). In our GNN approach, we considered each deployment as a graph, which means that 480 graphs were used for training and 120 for validation. Every model uses the Root Mean-Squared Error (RMSE) as a loss function, defined below. The error was obtained across the predictions ($\hat{x}_i$) compared to the actual results (x_i), where N is the number of devices in the batch. Note that the batch size for the GNN model is 32 graphs, while traditional DL approaches use a batch size of 32 and 128 devices for the FNN and the CNN, respectively.

The used data set considers several deployments in similar settings (e.g., fixed 12-AP setting in scenarios *1a*, *1b*, and *1c*. See Table 4.2), which allows the ML approaches to finding patterns in the data to build a general representation of the network. This representation does not perfectly fit every single training datapoint because otherwise, the models will overfit. On the contrary, this representation minimizes the error between the datapoint and the prediction.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}}$$

However, during data analysis, we found that the mean STA's throughput is much lower than the mean AP's throughput (see Table 4.2). Therefore, the throughput of the APs is considered an outlier for any ML approach. Additionally, the initial exploratory results showed that the models are mostly focused on correctly predicting the throughput of the APs, given that the RMSE is minimized on large values. Consequently, we proposed a

masked loss where only the STAs contributes to the loss. In other words, the network does not need to learn to predict the AP throughput. The AP's predicted throughput is computed at a post-processing step by summing its associated STAs' predicted throughput. The RMSE is calculated, using the STAs' predicted and the APs' computed throughput.

The model's (FNN, CNN, and GNN) hyper-parameters were selected, using a hyper-parameter search over hundreds of executions. The best results of this search were obtained using Adam optimizer with a learning rate of 0.001. The models were implemented in Python, using Tensorflow 2.1.2 for FNN and CNN and PyTorch Geometric [134], a geometric deep learning extension library for GNN. The training was accelerated by using GeForce GTX 1080 Ti GPUs in our GPULab facility [4]. We trained the models on each set of features defined in Table 4.4 to quantify how they affect the prediction accuracy. This procedure was performed ten times per experiment to observe each model's convergence.

Performance Evaluation The testing data set also includes several deployments in enterprise-like scenarios with same channel configurations. However, changes regarding the training data set, including map size, deployment size, and each deployment's user density, were introduced and are shown in Figure 4.5b. In this case, four scenarios were simulated, according to the number of considered APs in a fixed map size (see Table 4.2).

As a baseline for all ML approaches, we use a random guesser. We assume that the throughput can be obtained from a normal distribution with the mean and standard deviation found during data analysis to build this random guesser. Given that the throughput in STAs varies between 0 and 88 Mbps, we use a truncated normal distribution between those values. This random approach represents a naive and cheap way to generate predictions in this particular problem.

The trained models were used to predict the throughput of all devices in the test data set. Figure 4.8 shows the mean RMSE across all test scenarios' deployments in a given experiment for all the generated models and their standard deviation. The standard deviation in all models is very low, having a maximum of 0.66 in the FNN. As can be seen from the figure, GNN outperforms all other approaches in all defined experiments. The random approach performs more or less the same, independent of the features used. Regarding the other ML models, it can be seen that learning from data represents at least a 20% improvement regarding the random approach, using all trainable features (E1). Focused on E1, i.e., the experiments with all features, GNN can obtain up to 64%, 56%, 55%, and 54% when comparing it against the random approach, the CNN, the FNN, and the GB, respectively. Surprisingly, GB performs slightly better in several experiments when compared to the CNN and the FNN. Despite its complexity, the CNN does not perform better than the FNN and the GB. This poor performance might be due to data representation. CNNs outperform other ML approaches when dealing with high-dimensional data (e.g., images, time series). Even though the wireless environment is too complex to be modeled, the provided data do not include an extra dimension (e.g., time) that CNN can benefit from.

In some experiments (e.g., E5, E6, E8, E13, E14, and E15), the random guesser performs better than some ML approaches since the combination of input features does not benefit from the learning process. In fact, the Gaussian assumption works well in some cases,

[4] `https://gpulab.ilabt.imec.be` (accessed on 21 June 2021)

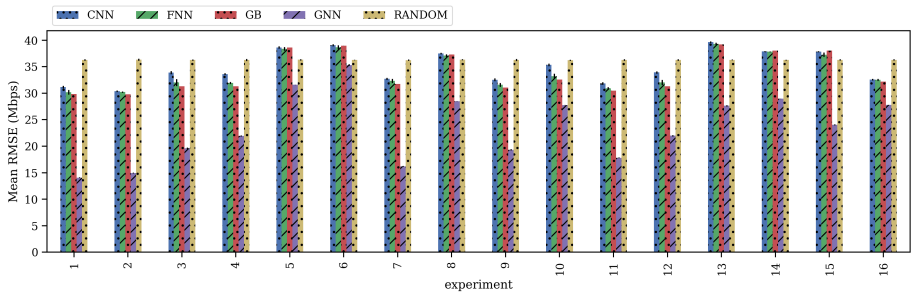


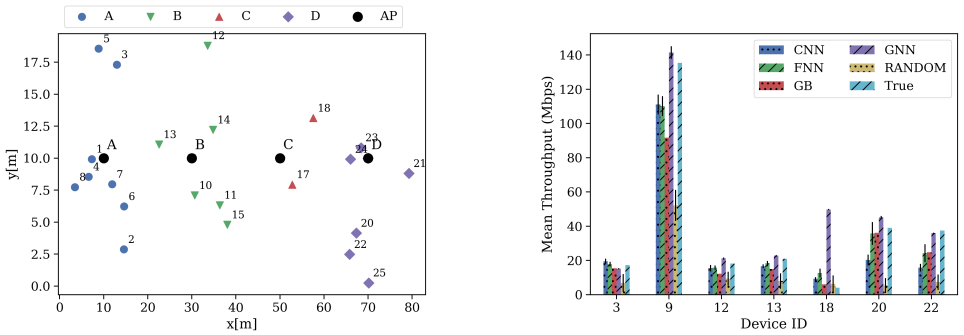
Figure 4.8: Mean and standard deviation of the obtained RMSE by all models on the test data set.

and it is widely adopted in many aspects of communications. Moreover, the random guesser performs more or less the same, independent of the input features, while for some experiments, the ML approaches benefit from certain features (e.g., E1, E2). This random guesser includes some basic information and performs better than a best-effort approach. It serves as an upper bound to the ML approaches, as it shows if the ML model is actually learning.

In terms of feature relevance, including airtime, there is a strong improvement to the results of all ML models. For instance, the only difference between E1/E7 and E15/E8 is that the latter does not consider airtime while the former does. It can be seen that not considering airtime represents between 85% and 87% decrease for the GNN, while other ML approaches perform even worse than the random approach. Nonetheless, considering only airtime as an input feature does not ensure good performance. For example, in E16, GNN decreases its performance by more than 100% regarding E1, except for other ML approaches, where using airtime as an input feature performs even better than considering the rest of the features (see E15).

Analyzing other features, RSSI and distance give more information about the throughput than SINR, node type, and interference. For instance, in E9 and E10, all models improve their performance by including RSSI: 48%, 5%, 10%, and 6% in the GNN, GB, CNN, and FNN, respectively. Similarly, in E11 and E12, GNN obtains around 26% improvement, while CNN and GB obtain 2.4% and 7.5% improvement, respectively, when considering the distance. Interestingly, the GNN is the only model in which features such as node type (E3 vs. E4) and SINR (E5 vs. E6) are relevant and improve its performance. Even when considering interference (E13 vs. E14), a factor that seems to decrease other ML models' performance, GNN obtains a 5% improvement.

Based on our experiments, we observe a low correlation between the location of APs and the performance of STAs. Some experiments use the devices' location for training the models (e.g., E1), but this information is not used in other experiments (e.g., E2). There is a slight drop in accuracy between E1 and E2, but, in general, the position of the APs does not greatly impact the performance of the models. This conclusion is also motivated by the correlation matrix shown in Figure 4.6 that suggests that the positioning of the nodes (x and y features) is not a relevant feature for determining the throughput. In turn, other features are more representative and might encode the positioning of the nodes (e.g., SINR, distance, and RSSI).



(a) WLAN deployment.

(b) Predictions.

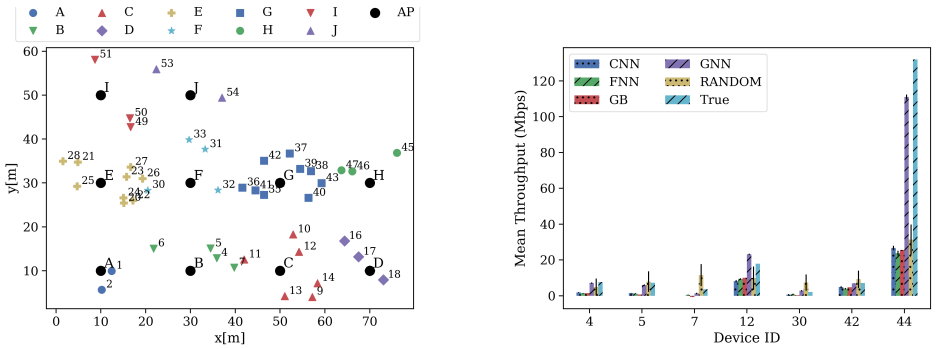
Figure 4.9: Deployment and predictions of some devices of test scenario 1.

Zooming in on the quality of the predictions, Figure 4.9 shows a random deployment of the scenario *1* in the test data set (Figure 4.9a), and the mean predicted throughput by all the models (Figure 4.9b) in E1. For visualization purposes, only seven devices per deployment were randomly selected. As explained in the previous section, APs' throughput is relatively higher than STAs' throughput since it is aggregated from its associated STAs. This can be seen in device 9, which represents the AP *B* (see Figure 4.9a), where it aggregates the throughput of STAs 12 and 13. As it can be seen, all the models accurately predict the expected throughput of the BSS. However, GNN predictions are, in general, less accurate than the predictions performed by other models (excepting the random guesser). In less crowded scenarios, GNN presents more difficulties in predicting the significantly lower STAs' throughput (device 18). On the other hand, GNN is better than other approaches at predicting large throughput values (devices 9, 20 and 22). This could be due to the ability of GNNs to exploit the connectivity of the nodes in the aggregation stage of the message passing. The less crowded scenarios have fewer neighboring nodes and less information to be aggregated, which lowers the quality of the predictions.

Focusing on more crowded scenarios, Figure 4.10 shows a random deployment of the scenario *4* in the test data set in E1. Figure 4.10a shows the deployment, while Figure 4.10b shows the predictions. In this case, GNN manages to accurately predict smaller throughput values (devices 4, 5, and 42), yet, it presents difficulties in predicting larger throughput values (devices 12 and 44). Nonetheless, its predictions are significantly better than the ones made by the other models in these same devices.

Note that the RMSE in Figure 4.8 is lower than the one that can be inferred from Figures 4.9b and 4.10b; because it is computed using all the predictions from all scenarios' deployments and from all runs, i.e., the considered number of devices, N is larger than the number of devices in a deployment.

In general, the accuracy of the ML models strongly depends on the features used for training them. Therefore, based on the available features, a management system might select a given model from the marketplace, sacrificing model accuracy. The results also showed that approaches such as GNN benefit from having more features, as the model obtained better performance in each performed experiment that included a given feature versus its complement. Nonetheless, GNN was able to outperform other ML models,



(a) WLAN deployment.

(b) Predictions.

Figure 4.10: Deployment and predictions of some devices of test scenario 4.

independent of the used features, given its ability to learn relationships among nodes in a graph.

4.4 Network Intelligence Inter-Technology Interference Management Framework for Wi-Fi/LTE Coexistence

This use case considers two geographically co-located wireless technologies, Wi-Fi and LTE, operating simultaneously in the same spectrum. The interfering technology (i.e., LTE) operates in legacy mode, which degrades the throughput of the incumbent technology (i.e., Wi-Fi) due to the interference. On the other hand, Wi-Fi operates according to the standard, i.e., there is no modification in its standard Medium Access Control (MAC) layer to support spectrum-sharing mechanisms.

Under these assumptions, previous literature [122, 149] has shown that according to the CSMA/CA implementation of Wi-Fi, transmissions are only possible after a channel occupancy estimation. In the case of LTE transmissions in a coexistence environment, Wi-Fi nodes detect either that the channel is busy or that there are collisions, resulting in a negative impact on their performance. Therefore, efficient and effective coexistence mechanisms that allow both networks to increase overall performance are vital. Such mechanisms should share information on co-located networks' activities to avoid transmission corruptions and enable global/local optimizations [150].

In that sense, cognitive radio approaches based on DL techniques [151, 152] help to detect the technology that is accessing the medium. Hence, different management decisions can be made given that some technologies are more benevolent than others [153]. In that way, radios can learn to identify where and when to access the spectrum resources opportunistically. Such DL techniques usually do not require feature engineering, making them more attractive to implement in a multi-technology coexistence framework.

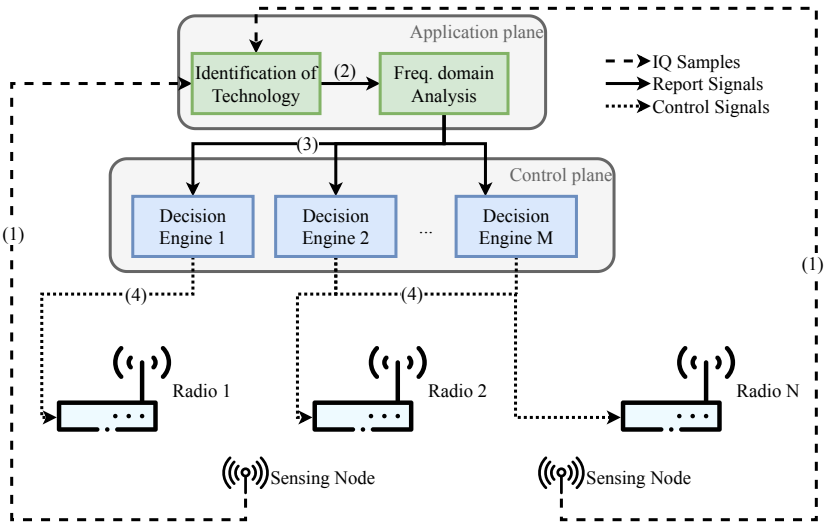


Figure 4.11: Proposed spectrum management framework

4.4.1 Framework Architecture

To provide different wireless networks with a broader view of the spectrum resources and help them to make better decisions, we propose a distributed spectrum management framework, as depicted in Figure 4.11. This framework comprises several steps; on the first step (1) to enable an inter-technology coexistence, a Technology Recognition (TR) module is required [109]. This module must detect which technologies are accessing the medium. For this purpose, the TR module receives raw In-phase and Quadrature (IQ) samples from the different co-located radios and identifies their respective signatures. Afterward, (2) a per domain Frequency Analysis (FA) is performed where each identified technology's spectrum occupancy is calculated in each of the monitored frequency channels; (3) this information is shared with different spectrum decision engines that make better-informed decisions suitable to the current working conditions. Once a decision is made, (4) the radios can be notified through control channels. We also highlight in the figure how this management framework can be implemented in an SDN-like architecture. For instance, the TR and FA modules form the application plane, while the decision engines form the control plane. Controllers can instantiate applications on an on-demand basis. It is worth mentioning that the decision engines can be separated from the radios or integrated into MAC layer algorithms to access the medium. The latter is the case of the LTE network.

Initially, the TR module monitors different channels, including the ones where the transmissions are taking place and the free ones. The TR and FA modules acting together send reports of spectrum occupancy of the detected technology on the monitored channels. The two co-located technologies, i.e., Wi-Fi and LTE, simultaneously transmit, causing interference to each other. The Wi-Fi controller is connected to the TR module through a publisher/subscriber paradigm. It also implements a rule-based management algorithm that changes its central frequency to a less crowded channel when other technologies' spectrum occupancy is higher than a threshold. In this way, both transmitting technolo-

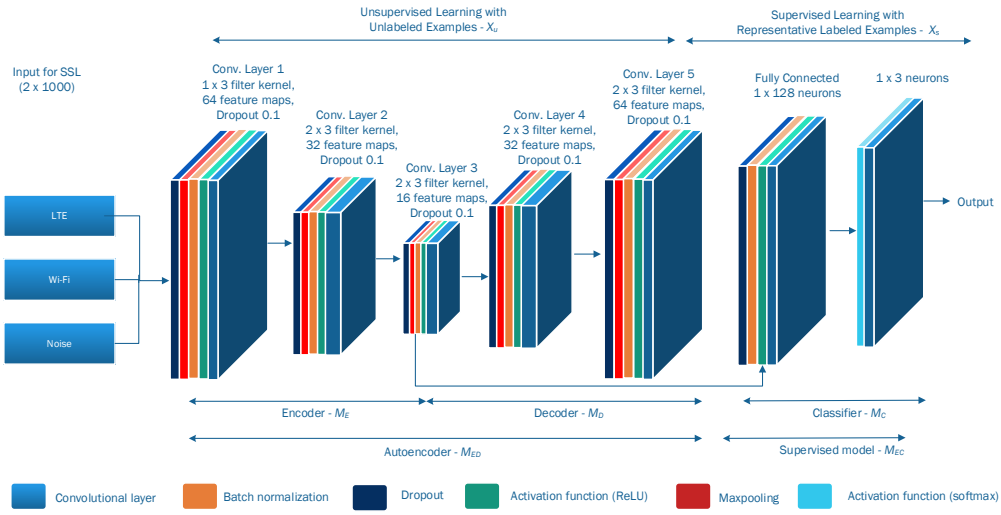


Figure 4.12: Semi-supervised learning architecture implemented using AEs.

gies can efficiently use the spectrum without performance degradation. The details of each of the modules are given as follows.

4.4.1.1 Technology Recognition (TR)

The TR module is a ML architecture based on CNNs [109]. The model is trained in a semi-supervised way using Autoencoders (AEs). The architectural design of the TR module is shown in Figure 4.12. Thanks to the semi-supervised learning approach, the offline training for wireless technologies recognition is simplified as the amount of labeled data is reduced. The input of the model consists of an IQ vector that is obtained as follows: (i) a Universal Software Radio Peripheral (USRP) B200 mini senses the wireless spectrum at a sampling rate of 20 Msps, (ii) the binary data is interpreted as an IQ stream of data, (iii) the IQ stream is reshaped into a 3-dimensional array with $n = IQ_{length}/1000$ vectors, two dimensions for the IQ components and 1000 samples in the time domain. These vectors are provided to the model, which has been trained to predict three classes: LTE, Wi-Fi, and Noise. The model is trained using both a supervised and unsupervised dataset. The advantage of the model's unsupervised learning capabilities is feature learning without the need for extensive labeled datasets. The AE M_{ED}, as shown in Figure 4.12, learns these features by trying to reconstruct the input vector and by having reduced dimensionality at the last layer of the encoder M_E. Afterward, a supervised model M_{EC} is created by attaching a classifier M_C to encoder M_E, which is trained to predict the signal classes based on the supervised dataset.

The AEs have the following architectural design. The encoder has two convolutional layers with ReLU activation function, each one followed by a batch normalization, which accelerates the training, and a dropout layer, which improves generalization. For down-sampling the input, the encoder uses Max-pooling layers and obtains an intermediate code that is 16x smaller than the original input. The decoder follows the same pattern

Algorithm 1: Wi-Fi ruled-based spectrum manager

```
input  : Spectral Occupation per technology per channel
output: Operating Channel
while report in queue do
    Receive report from TR ;
    if occupation from other technologies > threshold then
        Get current working channel ;
        Rank Channels using Equation 4.1 ;
        if Best-ranked channel ≠ working channel then
            Change to best-ranked channel ;
        end
    end
end
```

as the encoder but in reverse order; transposed convolutional layers replace the convolutional layers. The supervised part of the architecture comprises the encoder M_E followed by a classifier M_C composed of two dense layers. The last layer has a SoftMax activation function for classification.

4.4.1.2 Rule-Based Wi-Fi Management

The Wi-Fi controller then follows Algorithm 1, which is enhanced by AI by having a detailed report of the status of the channels that TR is monitoring. Once a report is received, it checks if the spectrum occupation of different technologies in the current working channel is higher than a given threshold. This threshold represents the maximum spectrum occupation of a technology that Wi-Fi tolerates; it can be dynamically calculated or learned and adjusted per Wi-Fi client to guarantee QoS requirements.

$$rank = \alpha \cdot \sum_{i \in I, i \neq WiFi} SpOcc_i + \beta \cdot SpOcc_{i,i=WiFi} + \gamma \cdot SpOcc_{free} \quad \forall c \in C \tag{4.1}$$

If this condition is met, the algorithm ranks every reported channel using Equation 4.1, where I is the set of all technologies identified by the TR module, and C is the set of channels that TR is monitoring. α is the weight assigned to the spectrum occupation of other technologies except for Wi-Fi; β is the weight assigned to the spectrum occupation of Wi-Fi and γ is the weight assigned to the unoccupied spectrum. These weights represent the preference of the algorithm. For instance, an unoccupied channel is preferred over a channel with only Wi-Fi transmissions. If the current working channel is different from the best-ranked channel, the controller instructs the AP to change channels. We make use of the Channel Switch Announcement (CSA) procedure defined by the *h* amendment in 2003, which was fully integrated into the IEEE 802.11-2012 standard [115], the most widely supported standard for Wi-Fi transmission. In this procedure, the AP informs the associated Wi-Fi users that the operating channel will change in a given number of beacon frames, so users do not disassociate from the current AP.

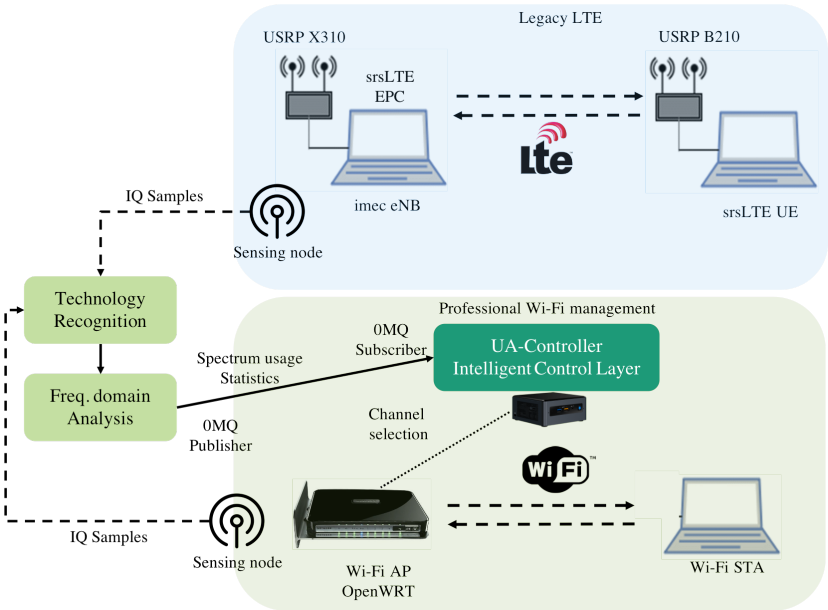


Figure 4.13: Small testbed composed of a Wi-Fi setup, a legacy LTE setup, and a TR module

4.4.1.3 Long-Term Evolution (LTE)

In this study, a private LTE network implementation that can operate in both private and unlicensed bands [154] has been used to introduce interference in the unlicensed channel in which Wi-Fi operates. The private LTE network operates in different frame configurations, with different uplink, downlink, and blank subframes combinations. By varying the frame configuration of the LTE transmission, the proposed coexistence scheme can be evaluated for different scenarios.

4.4.2 Experimental Validation

This section reports the results obtained during the experimental evaluation of the Wi-Fi performance when LTE is interfering by considering two coexistence approaches. In the first approach, no collaboration beyond the standard MAC layer algorithms nor communication exists between the two co-located technologies (non-managed, legacy solution). The second approach integrates the Wi-Fi controller following Algorithm 1 with the proposed spectrum management framework to mitigate the LTE interference (our solution). Notice that the TR module is validated in Camelo et al. [109].

4.4.2.1 Experimental Setup

We implemented the proposed management framework using open-source hardware. To test our solution, we build a small testbed, as shown in Figure 4.13. In particular,

we use a Netgear WNDR4300 as AP running OpenWRT 19.07 and a wireless card with Atheros AR9344 chipset using IEEE 802.11n in the 2.4 GHz band. Due to frame aggregation, oversized frames such as video are partitioned into multiple sub-frames in IEEE 802.11n. In the case of interference, some sub-frames might be lost. However, on average, the receiver aggregates most of the data to reproduce the original frame, reducing the probability of data loss. Technologies such as IEEE 802.11b/g cannot aggregate frames; thus, the data loss is expected to be more significant. As a consequence, it is expected that the throughput will be even more affected in such technologies than in IEEE 802.11n. On the contrary, technologies such as IEEE 802.11ac allow frame aggregation, and therefore the improvements in throughput are expected to be more or less the same as the ones shown here. However, IEEE 802.11n in the 2.4 GHz band was selected because it is a good transition between legacy networks (i.e., b/g) and recent networks (i.e., ac). Moreover, the adoption of IEEE 802.11ac is not as broad as it is for IEEE 802.11n.

As a Wi-Fi controller, we employ an Intel NUC7i5BNH running Ubuntu 18.04 LTS with the Algorithm 1. As a Wi-Fi client, we use a MacBook Air 7,2 with a wireless card Broadcom BCM4360 1.0 (7.77.37.31.1a9). The controller, AP, and the client are connected in the same sub-network. We use *iperf* to generate traffic between AP and the client.

The TR module continuously monitors the spectrum and captures IQ samples using USRPs at a sampling rate of 20 MHz. Three USRPs were used and centered at three non-overlapping Wi-Fi channels 1, 6, and 11, respectively, monitoring the complete 60 MHz Wi-Fi spectrum. The IQ samples from each USRP are transformed into data vectors and are provided to the AEs. Finally, the FA module calculates the spectrum occupancy information per technology on each of the three channels from the identification results and publishes it.

As for the LTE part, SDRs have been used for the Evolved Node B (eNB) and the User Equipment (UE). The eNB transmits and receives LTE IQ samples using a USRP X310, while the UE uses a USRP B210. Each USRP is attached to a host computer where the corresponding eNB and UE software runs. At the UE side, srsLTE UE software implementation has been used. On the eNB side, the private LTE implementation proposed in [154] is used. The LTE frame configuration was varied to illustrate different traffic scenarios such that the LTE uses different portions of the spectrum. Furthermore, srsLTE core network implementation has been used for the LTE Evolved Packet Core (EPC).

We communicate every system using the publisher/subscriber paradigm implemented in ZeroMQ (0MQ) [155]. Here, the TR module acts as the publisher and is connected to the Wi-Fi controller using another sub-network. The messages containing the spectrum occupation per technology and per channel are formatted using Google Protocol Buffers [156]. They are sent approximately every 4s to a 0MQ queue that later is processed by the Wi-Fi controller.

4.4.2.2 Performance Evaluation

We performed two sets of experiments on two different days. In the first set of experiments (Day 1), we tested our solution's effectiveness compared to a non-managed solution. During this day, the experiment's room was not completely isolated from other wireless interactions (e.g., other Wi-Fi networks and Bluetooth connections). We generate

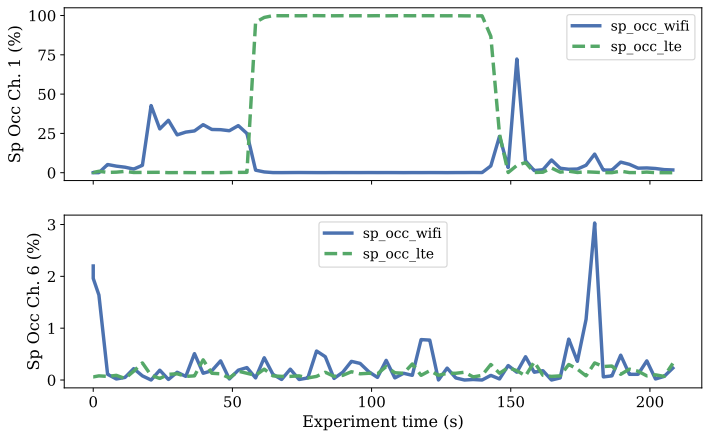


Figure 4.14: Spectrum Occupation given by TR in both reported channels, no Wi-Fi management activated

30 Mbps of Wi-Fi User Datagram Protocol (UDP) traffic from the client to the controller on channel 1. After a while, we start the transmission of LTE, generating 30 Mbps of bi-directional UDP traffic. According to Algorithm 1, our solution is activated if the occupation of other wireless technologies reported by TR is above a threshold, which we set to 40%. Then, the reported channels are ranked according to Equation 4.1 in which the values for α, β, and γ are set 0.4, 0.3, and 0.3, respectively. The value of α must be low enough so the impact of other technologies' interference remains low, while the value of γ must be high, so an unoccupied channel is preferred. Later we will show the effect of different parameter values in the threshold. Every experiment has an approximate duration of 2 min (120 s).

Figures 4.14 and 4.15 show the results obtained with the non-managed solution. More specifically, Figure 4.14 shows the reports on spectrum occupancy received by the Wi-Fi controller from the TR and FA modules in both channels over time. As can be seen, Wi-Fi occupies around 30% of channel 1, where the transmission was started. Once the LTE traffic starts, the occupation of Wi-Fi drops completely. After the LTE transmission stops, Wi-Fi occupation increases during a short period, probably because it tries to resume its transmissions with a lower MCS.

Wi-Fi performance, in terms of effective throughput, is shown in Figure 4.15. As expected, when LTE is not interfering, Wi-Fi freely uses the channel achieving around 20-25 Mbps of throughput. This shows the impact of the interference of more benevolent technologies like Bluetooth. Once the LTE transmission starts, the Wi-Fi throughput decreases as there is no mechanism activated to mitigate LTE's effect. Despite that other channels remain unused, the spectrum occupation of Wi-Fi drops to zero, and contrary, LTE takes control over the entire channel. At the beginning of the LTE transmission, the throughput does not reach its maximum level (30 Mbps) as it has to share the spectrum with Wi-Fi. It is not until Wi-Fi performance degrades that the LTE throughput reaches its maximum.

Figures 4.16, and 4.17 show the results obtained with our solution. Now, the Wi-Fi

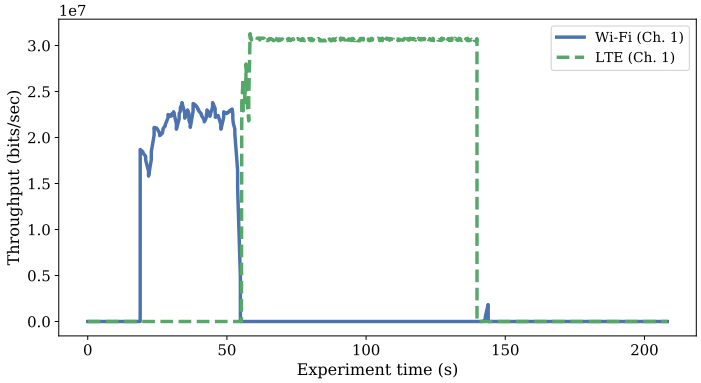


Figure 4.15: Effective throughput of Wi-Fi and LTE clients, no Wi-Fi management activated

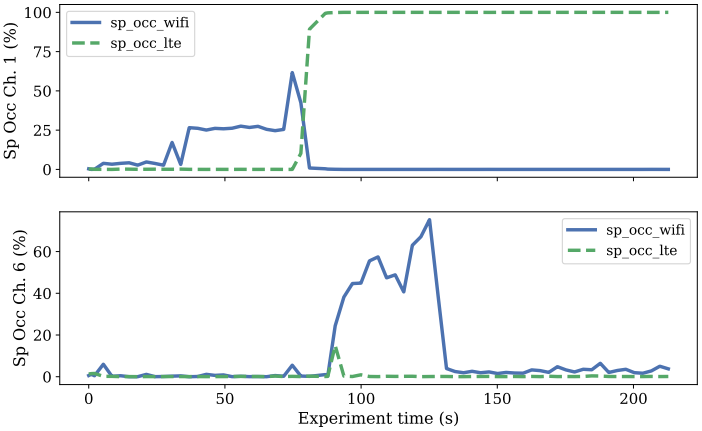


Figure 4.16: Spectrum Occupation given by TR in both reported channels, Wi-Fi management activated

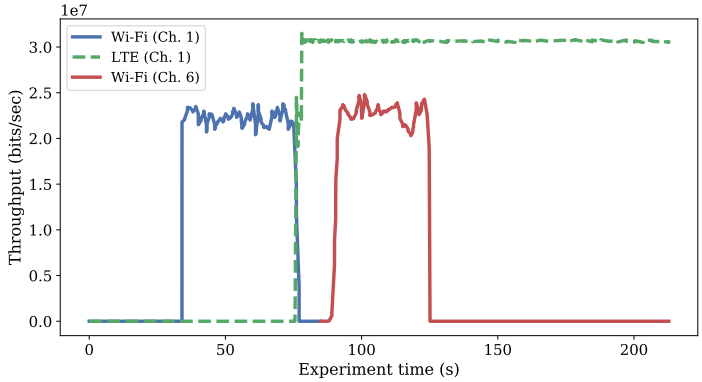


Figure 4.17: Effective throughput of Wi-Fi and LTE clients, Wi-Fi management activated

controller takes action to mitigate LTE interference. The messages received by the Wi-Fi controller from the TR and FA modules reporting the spectral occupation per technology and per channel are shown in Figure 4.16. Wi-Fi spectral occupation is around 30%, as in the first case. However, once the LTE transmission starts, the spectrum occupancy increases for a short time as the SDR is lowered to protect current communications. Unlike the first case, this time, the Wi-Fi controller decides to change the AP and its associated clients to channel 6 as it is reported to be free. As it can be observed, the reported Wi-Fi spectral occupation on channel 6 increases and remains relatively high until the experiment ends.

The performance regarding throughput in Figure 4.17 confirms the effectiveness of our solution. After the threshold of spectrum occupancy of other technologies is reached, the controller decision is triggered, and the CSA is activated. Wi-Fi is briefly affected by the decision to change channels, which is due to CSA. However, the throughput before the channel change was again achieved. Wi-Fi's reaction time is difficult to assess, given that the experiments were not performed in an isolated environment. Although we defined three beacons to perform the CSA, the time to process this instruction depends on the processes that are present in the AP, plus the time to receive the report from the TR at the controller side. Notice that none of the entities involved in the experiment is synchronized in time.

In the second set of experiments (Day 2), we tested our solution's behavior under different LTE Tx-Opps and compared it with the non-managed solution. This time, an environment without many interfering technologies was set, and 20 Mbps of Wi-Fi UDP traffic and 30 Mbps of bi-directional LTE UDP traffic were generated. Values for α, β, and γ are maintained as in the first experiment.

Figure 4.18 shows the averaged throughput of the non-managed and our solution with different threshold levels. An LTE Tx-Opp of 0.8 means that Wi-Fi uses 20% of the spectral resources. As can be seen, the more resources are given to Wi-Fi, the better the performance. More specifically, the throughput of the non-managed solution decreases while our solution can preserve the throughput as the LTE Tx-Opp increases. When LTE operates at a 20% Tx-Opp, the Wi-Fi throughput is more or less similar for both solutions given that Wi-Fi can use 80% of the spectral resources. Unfortunately, LTE performance

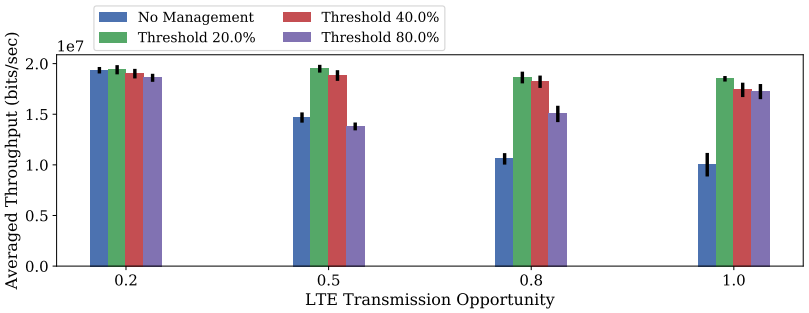


Figure 4.18: Averaged Wi-Fi throughput of the non-managed solution vs. our solution varying the threshold level under different LTE Tx-Opps

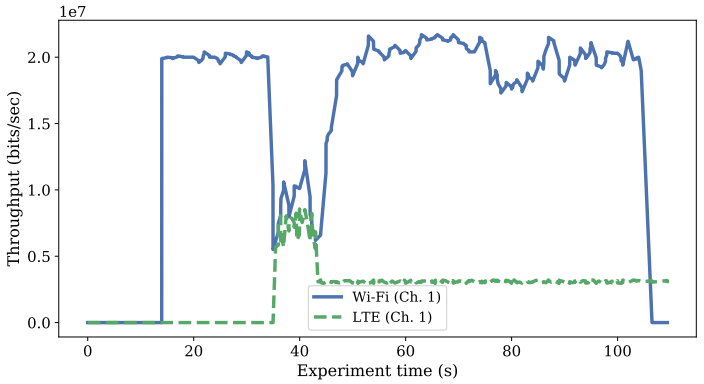


Figure 4.19: Effective throughput of Wi-Fi and LTE clients, Wi-Fi management activated, 80% threshold, 20% LTE Tx-Opp

is severely affected in this configuration. Its effective throughput lies around 3 Mbps, which is a decrease of almost 90% of its expected performance, as shown in Figure 4.19. This situation is not desirable as, ideally, a fair co-existence must be achievable. It can also be seen how the Wi-Fi throughput is severely affected by the control signals emitted by LTE at the beginning of the transmission.

Cases where our solution does not perform that well are given when an 80% threshold is used, and LTE is not using all the spectral resources (i.e., a Tx-Opp of 0.5 or 0.8). In such cases, the spectral occupation of other technologies reported by TR is not higher than the threshold. Therefore, no channel transition is made, causing a decrease in performance. Figure 4.20 shows the spectrum occupancy on both channels when our solution uses an 80% threshold and LTE is giving 50% Tx-Opp. As it is observed, the reported Wi-Fi occupation is around 80% on channel 1 until LTE starts its transmission, where both technologies fairly share the spectral resources. However, as shown in Figure 4.18, Wi-Fi performance is around 13 Mbps, translated into a decrease of 25% of its expected performance (20 Mbps).

Finally, Figure 4.21 shows the effect of the weights (α, β, γ) in the overall throughput. For

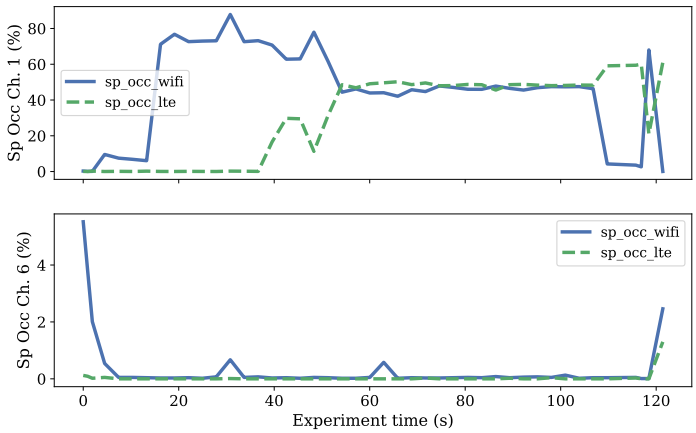


Figure 4.20: Spectrum Occupation reported by TR in both channels, Wi-Fi management activated, 80% threshold, 50% LTE Tx-Opp

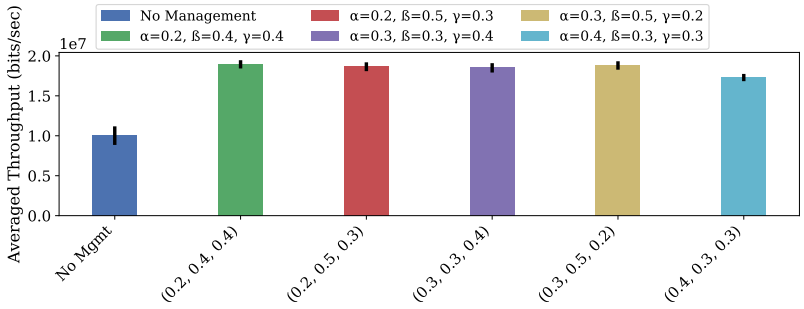


Figure 4.21: Averaged Wi-Fi throughput of non-managed solution vs. our solution varying the parameters (α, β, γ) to rank channels

this experiment, we configured LTE on legacy mode and varied α, β, and γ, as shown in the figure. As the channels' rank is based on the weights and the spectral occupation reported by TR, and as both reported channels were only occupied by the transmissions involved in the experiment, the effect of the ranking on the Wi-Fi performance is not that pronounced. The values of α, β, γ used for the first set of experiments represent a non-ideal setting as the achieved throughput is the lowest among the used configurations. This can be explained as follows. α is relatively high; thus, the channel's rank where LTE is causing interference will be higher than the free channel. More significant differences are expected when more wireless traffic is accessing the medium and more channels are available. However, independently of the chosen configuration, the overall throughput of our solution is at least 40% higher compared to the non-managed solution as our proposal has an enhanced view of the spectrum and can transition to a better channel to maintain its throughput.

4.5 Conclusion

The development of the 5G era has exacerbated network complexity for several reasons. One is the paradigm shift toward programmable networks that can be holistically orchestrated to offer end-to-end services. This network programmability is accompanied by an extreme set of requirements, such as ultra-low latency, highly reliable communication, and improved user experience. Therefore, to address this large set of requirements and diversity of network services, networks should be fully automated to accelerate service delivery while meeting economic goals.

However, current network architectures lack a high degree of automation, leaving network providers reliant on manual configuration. To address this, network models and optimization algorithms are developed. Network administrators provide an intent [157, 19] as input to the optimization algorithm, which then determines the network configuration that meets the requirements. In future wireless networks, where complexity is expected to rise due to extreme densification and higher performance demands, AI/ML will play a crucial role in modeling and guiding the behavior of these networks.

In this chapter, we developed two approaches for intelligent spectrum management, focusing on intra-technology and inter-technology interference use cases. The latter aims to handle the interference that is caused by two different wireless technologies, such as Wi-Fi and LTE, while the former handles the interference caused within the devices using the same wireless technology, i.e., Wi-Fi.

Regarding the intra-technology interference, traditional Markov models and other mathematical models have been developed to predict the performance (e.g., expected throughput) WLANs in the presence of interference, given a deployment (e.g., topology) and a particular configuration (e.g., set of channels that could be bonded). As discussed in Section 4.2.1, these models are based on simplifications and are limited by smaller deployments in order to keep feasibility.

Knowing that NNs are known to be good function approximators [158], in Section 4.3.1.1, we propose a NI approach whose intelligence is placed in the *Analyze* block. The main component is a GNN model that predicts the performance of a next-generation WLAN,

applying CB techniques, i.e., the aggregated throughput of the APs. The model adapts well to graph-based problems that exhibit combinatorial behavior, such as the ones presented in wireless communications.

Additionally, we compared our GNN model to more traditional ML models and analyzed the impact of different features on the model's performance. Depending on the available data, a controller might use a trained model with a given set of features or another. According to the evaluation, our GNN approach can obtain a 64% increase in the performance regarding a naive approach and around 55% for other ML approaches when using all training features.

Regarding the inter-technology interference, the proposed NI is a spectrum management framework that leverages an existing TR module [109], where the intelligence is placed in the *Monitor* and the *Analyze* blocks. This framework improves the coexistence in the unlicensed bands between two widely used technologies, namely, Wi-Fi and LTE. The NI is a modular solution composed of a TR and Frequency Analysis (FA) modules that identify each technology's signatures using a CNN and report the spectrum occupancy per channel. These modules are connected to a Wi-Fi controller through a publisher/subscriber paradigm that enhances the controller's view of the spectrum resources.

Moreover, we deployed a testbed to test our solution using open-source software and standard-compliant hardware. We have demonstrated that Wi-Fi can also adapt to guarantee coexistence with other wireless networks if an augmented view of the spectral resources is provided. Results showed that using our approach, Wi-Fi maintains its throughput even when LTE uses all the spectral resources of a given channel. These results can be extended to other Wi-Fi technologies such as IEEE 802.11b/g/ac by retraining the TR module to recognize new technologies.

Chapter 5

Network Intelligence for Scaling in Service-Based Network Architectures

The content of this chapter has been partially published in:

[159] **P. Soto**, D. De Vleeschauwer, M. Camelo, Y. De Bock, K. De Schepper, C.-Y. Chang, P. Hellinckx, J. F. Botero, and S. Latré, "Towards autonomous VNF auto-scaling using deep reinforcement learning," in Eighth International Conference on Software Defined Systems (SDS). IEEE, 2021, pp. 01–08.

[160] **P. Soto**, M. Camelo, D. De Vleeschauwer, Y. De Bock, C.-Y. Chang, J. F. Botero, and S. Latré, "Network Intelligence for NFV Scaling in Closed-Loop Architectures," IEEE Communications Magazine, vol. 61, no. 6, pp. 66–72, 2023. [IF 8.3]

The Fifth Generation (5G) of networks proposed significant architectural changes and improvements in network performance and capabilities. Network Function Virtualization (NFV) plays a crucial role in this advancement, enabling the deployment of traditional Network Function (NF) as software components on general-purpose servers, known as Virtual Network Functions (VNFs). This service-based approach facilitates flexible and scalable network service provisioning. However, managing and orchestrating NFV-based networks becomes increasingly complex due to the dynamic nature of services and the sheer number of VNFs involved. Such complexity calls for a solution that adapts to the needs of those network services. This chapter presents a Network Intelligence (NI) solution for achieving autonomous network management in 5G, specifically focusing on the scaling problem through Machine Learning (ML). We show how different scaling methods can fit the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) and highlight architectural differences between three prominent scaling methods, one based on Reinforcement Learning (RL), a Threshold (THD)-based algorithm, and another based on classical control theory, showing that only the data-driven approach is adaptable enough to achieve automation. Specifically, this chapter aims to solve the following Research Questions (RQs):

[RQ_3] **How can ML algorithms be tailored to incorporate network-specifics (e.g., topological information) in their learning objectives or adapting already gained knowledge from the ML field to networking to enhance their effectiveness**

in network management tasks such as interference management or resource allocation?

[RQ_4] **What methodologies can be developed to evaluate and select the most effective Artificial Intelligence (AI)/ML models for networking problems, ensuring optimal performance and automated testing and validation under changing network conditions, enhancing current Machine Learning Operations (MLOps) frameworks to provide comprehensive lifecycle management of such models?**

5.1 Context and Problem Statement

Mobile networks constantly evolve due to society's desire for connectivity, capacity, and newer services. The 5G and future generations of mobile networks are transitioning to a service-based architecture, where traditional NFs, i.e., Physical Network Functions (PNFs), are being replaced by small software pieces that can be enclosed into granular containers, i.e., VNFs, able to run in Commercial Off-The-Shelf (COTS) hardware. Moreover, VNFs can be composed to form a Network Service (NS), e.g., remote secured access composed of a firewall and a Virtual Private Network (VPN) gateway. With these softwarization and cloudification trends enabled by technologies like Software Defined Networking (SDN) and NFV, the network is becoming a general-purpose platform that provides connectivity to many heterogeneous devices while guaranteeing Quality of Service (QoS).

Such granular decoupling has numerous benefits. Multiple containers can share the underlying infrastructure since containers are not designed to run on specialized hardware or vendor-specific operating systems. Since a single platform may potentially be used for multiple applications, users, and tenants, hardware-software decoupling will lower equipment costs and energy consumption. Combined with SDN, these service-based network architectures allow for more flexible and dynamic network management [161]. Network engineers now have more options and better control over network performance, allowing quicker and more effective use of network resources. Network softwarization also creates automation and system integration possibilities, resulting in more efficient workflows across frameworks. The network then becomes a large pool of software, storage, and computing resources shared by different actors that can be programmed to optimize different objectives.

Nevertheless, while NFV and SDN bring improvements in network management over traditional methods, Network Operators (NOs) often use simplistic strategies to address challenges, which may not suffice in new or changing conditions due to their inflexibility. As a result, there's a pressing need for more sophisticated management solutions in NFV-based networks, aiming for a more adaptable framework where the network can autonomously adjust to changes with little to no human supervision, advancing towards the goal of zero-touch networking [80, 19].

In the context of the Sixth Generation (6G) networks, where Management and Orchestration (MANO) will be based on NI, this chapter focuses on scaling, a fundamental MANO procedure for the correct operation of NFV-based networks [162]. In NFV, VNFs can be scaled and moved on demand. Service Level Agreements (SLAs) are used in networking

to formalize stakeholder relationships; for instance, an SLA might specify how much latency a given NS can tolerate. Then, Communication Service Providers (CSPs) can adjust the size of their NSs or individual VNFs to satisfy SLAs while reducing Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) by enabling dynamic resource provisioning [163].

Resource scaling can be classified as horizontal or vertical. New VNF instances are added or removed in horizontal scaling as needed. In vertical scaling, the size of the VNF (e.g., its assigned computing and storage resources) is changed accordingly. Moreover, the scaling mode can be proactive or reactive. Proactive scaling requires the ability to infer the upcoming workload so that the resources are scheduled beforehand. In contrast, in reactive scaling, the resources are changed in response to the perceived variations. All the scaling strategies mentioned above try to find a balance between resource cost and fulfilling a given SLA (e.g., service latency). A NO can easily guarantee an SLA by over-dimensioning the number of VNFs but at the expense of increasing the economic cost. At the same time, this cost decreases when under-dimensioning the number of VNFs, but seriously compromises the SLA fulfillment.

Recently, ML strategies are being proposed for flexible resource scaling in NFV-based networks, given their ability to learn from data and past experiences. Most of the proposed strategies focused on predictive scaling, exploiting the historical data by using time-series forecasting [164]. Although it has been shown that predictive scaling produces better results in reducing the boot-up time of a new instance and yielding few SLA violations, reactive scaling is still widely used by the cloud industry due to its easiness of deployment, decent performance, and low computational cost. Moreover, predicting the workload can be an easy task if the workload follows regular patterns (e.g., during the daytime, peak hours on weekdays, etc.). However, the prediction accuracy might degrade under unseen workload patterns.

Simultaneously, scaling is considered a decision-making problem as it determines the number of replicas that fulfills the SLA. Decision-making problems are frequently modeled as Markov Decision Processs (MDPs). An MDP is a discrete-time stochastic framework that finds an optimal policy or strategy that maximizes the expected long-term reward, a discounted sum of immediate rewards. The optimal policy is found after many interactions with the environment until the scaler has converged. While MDPs offer the optimal solution to decision-making problems, the resulting MDP for continuous state problems becomes excessively large, necessitating alternative methods, including RL.

This is the main reason why RL is also explored as a solution for scaling network resources. An agent's objective in RL is to learn a policy that maximizes an expected Reward Function (RFn) by interacting with an environment through actions. Following the learned policy, the agent proactively adapts the network resources, similar to proactive scalers, but without any prior knowledge of the system.

Nonetheless, advanced predicting modules and modern RL approaches are nowadays based on Deep Neural Networks (DNNs), which require powerful computing processing. These resource-hungry DNNs might not suit resource-constrained environments (e.g., Multi-access Edge Computing (MEC) hosts). In this sense, selecting an appropriate scaler is a problem that depends on the network context. Therefore, an orchestrator might select a scaler depending on the available infrastructure, SLAs, and operational budget, among others.

The remainder of this chapter is structured as follows. Section 5.2 reviews the existing ML solutions for scaling. The mapping of the scaling solutions to the proposed Network Monitor-Analyze-Plan-Execute over a shared Knowledge (N-MAPE-K) framework is presented in Section 5.3. Section 5.4 details the scaling agents used in this study and their properties. Section 5.5 describes the discrete-time event simulator we use to train, tune and test the proposed solutions and summarizes the evaluation of the approaches.

Finally, Section 5.6 presents the conclusions.

5.2 Existing Solutions for the Scaling Problem using Machine Learning Techniques

Scaling is a challenging problem, mainly because it decides the exact amount of resources that a running service requires to meet an SLA. Although different techniques have been explored over a decade, we focus on the use of ML to solve the scaling problem, with a small emphasis on other techniques. Lorido-Botran et al. [165] and Chen et al. [166] review the proposed solutions for scaling in cloud environments, while Duc et al. [164] give a broader view on resource provisioning in edge-cloud computing using ML.

Usually, scaling can be divided into reactive and proactive. Reactive scalers respond to the current system status. These techniques manually and statically set the number of replicas, frequently by over-provisioning. Other reactive approaches include threshold-based methods [167], in which the scaler is told what to do if the network Key Performance Indicators (KPIs) are above or below a given threshold, using predefined rules. This threshold represents an optimal operation point, which is derived from expert knowledge. Such approaches only work in small configurations and are difficult to maintain, as either they are based on open-loop architectures or closed-loop but slow (e.g., the expert, as human-in-the-loop, needs to redetermine the threshold).

Control theory can also be used for scaling [168]. Here, the controller regulates a control variable, e.g., the number of resources, to keep a (measured) variable controlled, e.g., the KPIs specified in the SLA, as close as possible to a target value within some allowed boundaries. For instance, a Proportional–Integral (PI) controller calculates an error, which represents the deviation between the target and the measured value, and a trend, which captures how the measured value evolves. Since the objective is to keep the measured variable close to the target, the correction signal, i.e., the change that needs to be applied to the control variable, is a weighted sum of the error and the trend. There is also a third term for some controllers, but we do not consider it here. One of the advantages of using PI controllers for scaling is their steady response to small changes in the input variable (i.e., incoming traffic load). Moreover, if the system to be controlled is linear, the controller's parameters can be optimally set using Fourier or Laplace domain analyses. However, for non-linear systems, tuning the controller's parameters can be lengthy and computationally expensive.

In contrast, proactive scaling involves scheduling resources for upcoming network states. Such states are generally represented by the inferred future demands that the network needs to provide. Data-driven approaches are mostly used in proactive scaling, where such approaches employ methods to extract meaningful features from data, which can

later be leveraged in decision-making, emulating human intelligence. ML algorithms belong to this kind of approach [169, 170, 171].

Proactive scaling is the main area where ML is applied [172]. In proactive scaling, modern Supervised Learning (SL) techniques such as time-series forecasting are applied to find hidden patterns in the traffic load. The discovered patterns are then used to predict the expected load, which could be translated into scaling decisions (e.g., by applying a step activation function) so that MANO platforms can anticipate and set up the resources needed for future demand. More complex SL techniques, such as Convolutional Neural Networks (CNNs), are known for their automatic feature extraction (e.g., patterns) and function approximation (e.g., mapping to scaling decisions). To achieve this, they require enough labeled data to learn to identify such patterns and adapt accordingly. Data labeling is often a lengthy and manual process that, in most cases, also requires expert knowledge.

For instance, Subramanya and Riggio [173] developed a proactive scaler focused on a distributed MEC-NFV deployment. In their work, a Neural Network (NN) was proposed whose input is the traffic load in a time-series form to determine the number of VNF instances per cell at a given time. However, as the authors characterized the scaling as a classification problem, building the training dataset requires defining how many VNF instances are needed to serve a given traffic load, a non-trivial task requiring expert knowledge. Additionally, the traffic load traces seen in testing might differ from the traces seen in training; therefore, a considerable amount of training data must be available and labeled. As a final remark, they leveraged their access to real traces from a network operator, but their results cannot be replicated due to privacy.

Given its decision-making nature, RL [174] is also explored as a scaling solution. The focus of our review is on these kinds of techniques. However, for an in-depth analysis of such approaches, we recommend the reader to Garí et al. [175] where they reviewed RL-based scaling techniques in the cloud. In RL, an agent explores and analyzes the effects of different actions in an environment. This process can be seen as trial and error, where after many attempts, the agent learns a strategy that allows it to choose better actions over time. The main element in this learning process is the RFn, which tells the agent if the action it took was appropriate. This function guides the agent towards choosing the actions that maximize an expected reward, equivalent to achieving the learning goal. Unlike threshold-based or rule-based solutions, where a scaler is directly instructed on what to do, an RL-based scaler proactively adapts the NS instances according to a learned strategy. This behavior is similar to that of predictive scalers since, in a given state, the RL agent can anticipate future changes. However, RL does not require *a priori* knowledge of the system or expert knowledge to define the optimal operation points, making it more adaptable than static solutions.

Q-Learning is one of the most used RL methods for autoscaling. Q-Learning assigns a Q-value to an action-state pair (Q-Function) in a tabular representation. This tabular representation helps the agent improve its policy by selecting the actions with the highest Q-value. In continuous state or action problems, tabular Q-Learning is not scalable, i.e., the size of the Q-Table explodes in complex problems. To overcome this limitation, the Q-Table is replaced by a NN as a Q-Function approximator [176], creating Deep Q-Networks (DQNs).

Lee et al. designed in [177], a multi-tier scaling module. In NFV-based networks, services

can also be provided by composing multiple VNF in Service Function Chaining (SFC). In these multi-tier applications, the auto-scaler must decide how many instances of VNF are inside the SFC and the tier to scale. The authors proposed an RL-based scaling method based on DQN that uses an own-defined NN. The NN's input and output are defined as the status of each tier and the actions the agent can take (e.g., add or remove one VNF per tier or maintain them), respectively. The definition of the NN is tightly coupled with the problem size since it depends on the length of the SFC, which makes their approach not general enough to be implemented. Although they showed that the DQN method outperforms two THD-based approaches, DQN creates more VNF instances than the other two, incurring in over-provisioning.

Another RL-based scaler is proposed in [178]. The authors used Gaussian processes to improve the scaling policy to reduce the agent's errors during the exploratory phase in training. The Gaussian processes allowed them to model the system as a regression function that predicts the workload; then, they used those predictions to run hypothetical interactions in the training of the RL-based agent. The agent runs two processes: policy improvement and policy evaluation. The former allows the agent to foresee the results of its action, while the latter is the current execution of the policy. The authors showed that, the scaling policy could be improved by using a Gaussian Process as system model, leading to a more stable response in terms of mean response time and number of created instances.

A scaler for video conferencing systems is proposed in [179]. There, Gabriela et al. defined the selective forwarding unit of a video conferencing system as the VNF to be scaled. They used a DQN-based auto-scaler consisting of three layers: a Long Short-Term Memory (LSTM), a linear, and an output layer. The LSTM layer allows capturing the trend of the traffic. To do this, the authors deployed a time-series database so the features of the last four-time steps are stored. To define the state of the RL agent, the authors did not use infrastructure-level metrics such as Central Processing Unit (CPU) usage, but application-level metrics such as the number of conferences currently existing and the total number of participants in the meetings. These application-level metrics are indirect estimators of the workload.

Rossi et al. [180] proposed a model-based and a model-free RL algorithm for horizontal and vertical auto-scaling in a docker swarm environment. Regarding the model-free approach, they use *Dyna-Q*, a Q-Learning variant where the state transitions are simulated, speeding up the learning process. In the model-based approach, the authors directly solve the Bellman equation by estimating the transition probabilities using CPU utilization. Although *Dyna-Q* experiences slow convergence, the model-based approach adeptly learns the optimal adaptation policy following user-defined deployment objectives.

He et al. [181] show the potential of using RL for scaling decisions by considering the interrelationship between multiple replicas that are chained to compose a network service. In this case, the authors used a Graph Neural Network (GNN) to create chain-aware representations, and then, using RL, the scaling decisions are made based on the representations. The prototype is implemented in OpenNetVM, a high-performance platform for NS chaining, where offline training and online inference are carried out. The evaluation results show that the combination GNN-RL outperforms the baseline algorithms regarding overall system cost and packet processing rate. The reasons behind

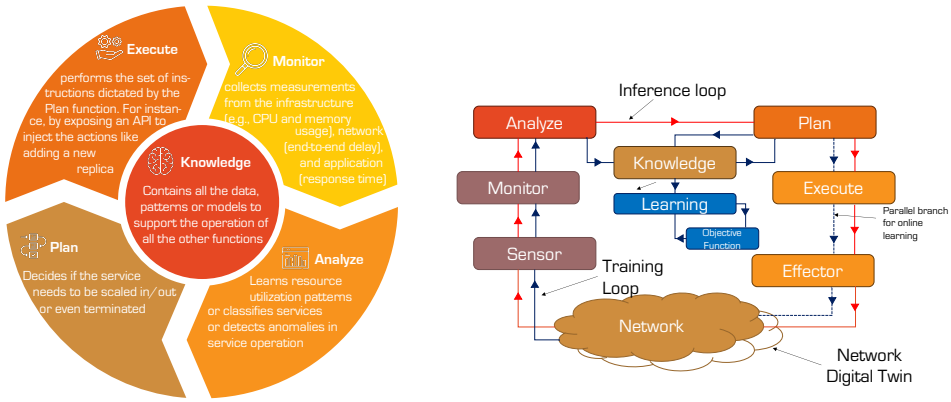
the improvement are based on the prototype's ability to i) learn from past experiences, ii) better demand prediction thanks to the composite features, and iii) the incorporation of the global chain information into scaling decisions.

The authors in [182] propose an RL algorithm with QoS guarantees to control a Kubernetes (K8s) cluster, reducing the microservices response time up to 20%. The RL algorithm identifies the right threshold values for pod scaling, which are then fed to the K8s Horizontal Pod Autoscaler (HPA). For training purposes, the authors simulated the HPA by taking the CPU and memory logs of two pods and adding a random factor on the resource usage and response time to increase the dataset size. The results indicated that the agents successfully reduced response time below the QoS target while adhering to a specified CPU utilization goal.

In [183], authors study the impact of microservice interdependencies in scaling mechanisms. They developed *gym-hpa*, a framework that integrates the popular OpenAI [51] platform, and a K8s cluster, allowing them to train RL agents on real cloud environments. Experimental findings demonstrate that the RL agents trained using the *gym-hpa* framework, on average, decrease resource usage by at least 30% and diminish application latency by 25% when compared to default scaling mechanisms in K8s.

With NFV, network resources can change dynamically to cope with the workload. Before NFV, traditional solutions were based on over-dimensioning the network resources to support workload peaks. Reactive and proactive approaches have been proposed for the scaling problem. Due to limited space, we have reviewed recent scaling methods that employ ML techniques in this section. However, it is worth mentioning that most of the solutions are focused on predictive scaling, exploiting the capabilities of ML to forecast/predict the future workload. The key differences between our work and previous works are summarized as follows:

- To achieve high prediction accuracy in ML-based predictive scaling, a large amount of data is needed. However, there is no guarantee that the obtained ML model will generalize in unseen workloads. Contrary, an RL-based scaler does not require any knowledge from the workload, making it more flexible under unseen data.
- Scaling can also be solved with a SL approach. In this case, labeled data is required. However, to define how many instances of a given VNF are needed based on a set of input features is not a trivial task, requiring expert knowledge. A RL-based agent does not need to have labeled data to learn since the agent learns by interactions with the network, simplifying the training process.
- The works showing RL techniques for scaling are opaque in reporting the work done at fine-tuning their RL agents. They lack a detailed overview of the parameters used in their agents, a practice highly discouraged by RL practitioners since it hampers the reproducibility of such techniques and further development. On the contrary, we thoroughly describe our training and testing procedures and follow the standard implementations of Stable-Baselines3 (SB3) [184], which already deliver the tuned hyper-parameters as intended in their original versions.



(a) General MAPE-K (b) Extended MAPE-K

Figure 5.1: The (a) general and (b) extended MAPE-K framework for NFV scaling, where the main differences are the learning function (in blue), a training and an inference loop which inject their outcomes to the network or its digital twin, depending on which loop is considered.

5.3 Closed-loop Control for NI-based Scaling

MAPE-K is an architectural framework for autonomic and adaptive systems consisting of a closed loop of four phases. During the first phase, the *Monitor* function collects data from the resources through sensors distributed over the system. In the second phase, the *Analyze* function correlates the received information and creates a system model, which could be used to predict future situations. Then, the *Plan* function decides the actions needed to achieve the system's objectives. Finally, the *Execute* function performs the planned actions over the resources. The *Knowledge* is represented by the data shared among the previous functions.

As explained before, scaling is mainly designed following a closed-loop control, similar to that proposed by MAPE-K. Thus, the scaling solutions can be naturally elaborated using the MAPE-K framework, as depicted in Figure 5.1a, where we show, in a general way, what every MAPE-K function expects of a scaling solution. The MAPE-K can represent the interactions between different components in a unified way, allowing their coexistence in the same network infrastructure.

Nonetheless, MAPE-K presents some limitations when contemplating AI/ML-based solutions, especially in determining if the NI is being trained or used in inference and distinguishing between different ML types (e.g., SL and RL). Therefore, we extend the definition of the MAPE-K, as shown in Figure 5.1b, to include such specificities coming from the integration of ML and networked systems. We introduce a learning block that optimizes an objective function. Notice that the objective function optimizes the algorithm's learning (e.g., minimizing the cross-entropy) according to the learning problem (e.g., classification), and it is not necessarily related to the optimal solution of the networking problem (e.g., selecting the best modulation scheme to minimize interference). Moreover, depending on which loop is considered, we include different training and inference loops, which inject their outcomes into the network or its digital twin. A digital

Table 5.1: MAPE-K decomposition of different scaling methods

MAPE-K Component	Scaling Method			
	Threshold-based	**Control-based**	**SL-based**	**RL-based**
Monitor	Resource State: CPU utilization Network State: service latency, E2E delay	Network State: service latency, E2E delay	Traffic Demands: Traffic load Resource State: CPU utilization, number of replicas Network State: service latency, E2E delay	Resource State: CPU utilization, number of replicas Network State: service latency, E2E delay
Analyze	Comparison between the monitored variables and the predefined thresholds	Computation of how the control variable needs to be changed as a weighted sum of an error term and a trend	The monitored variables are passed by a time series forecasting algorithm to learn hidden patterns	The monitored variables are averaged to compose the State.
Plan	Predefined actions according to the thresholds		Apply a mathematical formula to translate future patterns into scaling decisions	An agent takes the best action according to the learned strategy and current network state
Execute	An API to the MANO platform to communicate scaling decisions			
Knowledge	External: Human knowledge to define the threshold	Control terms' values	Model of the expected traffic load	Strategy with the actions to be taken according to the network state
Training Loss State/Actions/Rewards	N/A		Cross-Entropy	States: Avg CPU utilization, Avg latency Rewards: Resource utilization tolerance

twin network is a virtual representation of a physical network. Thanks to their access to real-time data, digital twins can accurately model complex systems. This way, multiple decisions can be tested safely without impacting the production network [25].

Regarding this last point, ML algorithms typically have two operation modes: training and testing/inference. Algorithms that have been tested on a digital twin can be readily introduced in production networks. In contrast, the outcomes in training mode cannot be applied directly. Since current network architectures do not naturally integrate these solutions, they are typically trained in non-production networks (e.g., simulators, emulators, and testbeds) and deployed in isolation (e.g., without interacting with other ML algorithms). Consequently, the gap between reality and simulation in the networking domain is big and difficult to close.

In Table 5.1, we decompose the most prominent scaling solutions using this extended version of the MAPE-K. All algorithms monitor similar variables and execute their decisions similarly by exposing an Application Programming Interface (API) to communicate with MANO platforms. The *Monitor* and the *Execute* functions represent the algorithms' inputs and outputs and are common to all scaling methods. However, depending on how the infrastructure exposes the monitoring information and the control parameters, the information these two functions process may vary.

The main differences emerge in the *Analyze* and *Plan* functions, where intelligent and non-intelligent methods differ. For SL-based scalers, the intelligence is in the *Analyze* function, where they identify the common patterns between the incoming traffic load and the scaling decisions. On the contrary, the intelligence of RL-based scalers is in the *Plan* function, where they translate network states into scaling decisions, freely deciding when to increase or decrease the number of NS instances. Notice that a combination of algorithms is also possible by using the same MAPE-K representation. For example, a scaling solution may combine an RL method to estimate the derivative term of a PI controller, which determines the number of replicas needed to meet the SLA [185].

Another difference among the scaling methods is how the *Knowledge* is acquired. In non-intelligent methods, the knowledge is extracted from the network and resides in behavioral rules (e.g., increasing the replicas by two if the threshold is surpassed), typically designed by CSPs, which are external agents. In contrast, in Network Intelligent Functions (NIFs), the knowledge can be automatically incorporated into the network by including the learning function explained above. However, the way the scalers obtain this knowledge is entirely different. While intelligent scalers obtain their knowledge by

training, the control-based tune the control parameters by running brute-force heuristics.

5.4 NI for NFV-scaling using Closed-Control Loops

In NFV-based networks, multiple network functions can be deployed as small software pieces in the form of VNFs. To work properly, these VNFs require a minimum amount of resources, e.g., processing power in terms of CPUs, which indicates the number of jobs per second it can process. Moreover, these VNFs can be deployed in COTS servers with limited CPU capacity, and as a consequence, the number of VNF instances is limited as well. Furthermore, the network receives a workload, which must be served under a given SLA, e.g., maximum latency. To meet this SLA, the number of computing resources must be dynamically and efficiently changed without incurring in under- or over-provisioning.

In this chapter, we assume a very simple yet challenging scenario where a CSP must deploy a service consisting of one containerized NF (e.g., video transcoder). Users or other services in the network can request this service by submitting processing tasks or jobs. To meet SLAs, the CSP resizes its NSs, by changing the number of the replicas of the NFs to fulfill operational and business objectives in a multi-objective setup [163]. We consider horizontal scaling instead of vertical scaling since it can be exploited by distributed applications where a workload is shared among different instances of the same application. While vertical scaling is limited to the capacity of a single server, horizontal scaling leverages the virtually infinite computing capacity in cloud environments, making it easier to deploy several instances of the same application. Additionally, horizontal scaling allows more granular control of NFs in multi-tier applications where each tier is scaled independently.

As mentioned above, the system is assumed to work in a time-slotted fashion. At the beginning of the slot, the workload, expressed in terms of jobs per unit of time, queues in front of a load balancer. If no jobs are in the queue, the workload is equally distributed among the number of active replicas. Notice that a job may incur some latency even when no queue is present since they are sequentially processed at the replica level. Otherwise, the incoming batch of jobs waits in the queue until they can be processed by one of the replicas. Each replica can process a predetermined number of jobs per time slot. Finally, a decision engine determines the number of replicas in the following slot, depending on the accumulated and the expected work to be processed. The process repeats at the next time slot. Figure 5.2 depicts the system described above.

Under these circumstances, scaling can be modeled as a decision-making problem since, in each time step, a decision about the number of replicas should be made, as an MDP. Modeling scaling as a MDP allows us to design the scaling solution as a closed-loop control, which leverages autonomicity and adaptiveness, as explained in the previous section. Then, the information retrieved by the *Monitor* function composes the network state. At time step t, the state s_t is defined as a tuple $\langle v_t, c_t, d_t \rangle$ where v_t represents the number of active replicas, c_t is the mean CPU usage among the active replicas, and d_t is the processing latency from the active replicas, i.e., the peak latency introduced by processing the jobs. Based on this information, the decision-maker decides its actions. This simple model keeps the action space small by only defining three actions. Thus, a_t is a number from the set $\mathcal{A} = \{-1, 0, 1\}$, where -1 means to decrease the number of

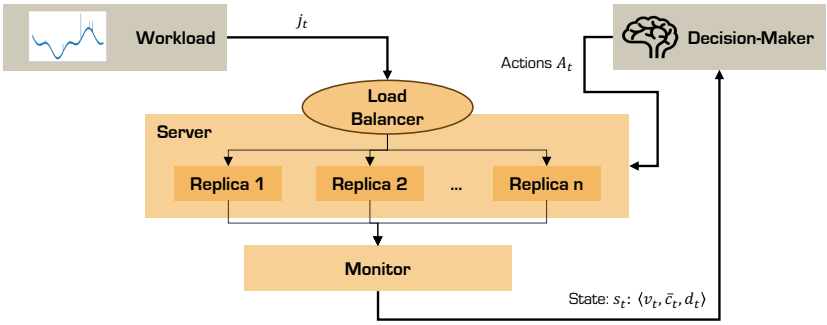


Figure 5.2: Simple model of the scaling problem.

replicas by one, 1 means to increase the number of replicas by one, and 0 will maintain the same number of replicas. Only one action is allowed per time slot.

To complete the framework, the decision-maker receives a reward every time slot if its action leads towards the end goal. Notice that if the reward is only penalizing extremely high latencies, the decision-maker will take the most obvious action: to keep increasing the number of instances, disregarding the economic impact of such a decision. Therefore, one of the critical aspects in designing a solution for an MDP is the RFn definition. Depending on the RFn, different policies can be found.

5.4.1 Scaling agents

5.4.1.1 Deep Reinforcement Learning (DRL)-based Agent

As stated in Section 5.2, SL can be used to solve the scaling problem. However, SL requires a labeled dataset for learning. To build this dataset, an expert must map a combination of the input features (e.g., latency, CPU usage, workload) to a scaling decision. Moreover, there is no guarantee that the trained model will perform well with input features not seen in the training phase, even if the model obtains high accuracy in the inference phase. On the contrary, RL is an online learning approach where the agent does not need a labeled dataset since it learns from interacting with an environment.

To determine the adequate amount of VNF instances to fulfill a maximum latency SLA using DRL, the problem needs to be formulated as an MDP, as done above. The optimal policy is found in RL after many agent interactions with the environment (i.e., in steady-state). Q-Learning is the most used algorithm to find this optimal policy by defining a Q-function for all state-action pairs to measure how good the policy is. Therefore, finding the optimal policy is reduced to finding the action that maximizes this Q-function. Note that the state-action pairs can be stored in a table (i.e., Q-table); however, its size will grow substantially according to the number of states (e.g., a continuous state space), preventing its wide adoption. Instead, DQN uses a NN as a Q-function approximator to overcome the Q-Table's size limitation.

Contrary to most of the RL applications in networking, where the states, actions, and RFn are defined using a networking rationale, in this RFn, we map the scaling problem

Table 5.2: Variables used in this chapter.

Variables	Description
$\mathcal{S}$	State space.
s, s_t	A particular state in step t.
$\mathcal{A}$	Action space.
a, a_t	A particular action in step t.
P	Transition Kernel.
$p(s'\|s, a)$	Transition probability to state s' by taking action a in state s .
r	Immediate reward.
$\pi(a\|s)$	Probability of selecting action a in state s.
ϵ	Tolerance range.
v_t	Number of active replicas in step t.
$\bar{v}$	Mean number of active replicas during a period of time.
c_t	Mean CPU usage of the active replicas in step t.
c_{tgt}	CPU utilization target.
d_t	Processing latency of the active replicas in step t.
$\bar{d}$	Mean latency of the active replicas during a period of time.
d_{tgt}	The target latency. The maximum tolerated value $(1 + \epsilon) \cdot d_{tgt}$ cannot be surpassed.
c_{perf}	Performance cost.
w_{perf}	Importance (weight) of c_{perf} in the total cost.
c_{res}	Resource cost.
w_{res}	Importance (weight) of c_{res} in the total cost.
V'	Normalized number of replicas created by an agent.
w_v	Importance (weight) of V' in networking scoring.
D'	Normalized latency of an agent.
w_d	Importance (weight) of D' in networking scoring.

to known applications of RL. One of the classical environments in OpenAI Gym is *Cart-Pole*,[1] where a pole is attached to a cart moving along a frictionless track. The pole is placed upright on the cart, and the goal is to balance it by applying forces to the left and right sides of the cart. Similarly, in scaling, the agent tries to guarantee a given SLA (e.g., latency) by taking discrete actions (i.e., increase, decrease, or maintain the number of replicas) that resemble the ones taken in *Cart-Pole* by applying forces to the left or the right. At time step t, the state $s^{(t)}$ is defined as:

1. Mean CPU usage among the active VNFs.
2. Mean number of jobs waiting in the queue.
3. Peak (maximum) latency from the active VNFs.
4. The number of active VNFs.

Based on this information, the DQN agent decides if the number of VNF instances must be increased, decreased, or kept the same. The RFn is also defined in a similar way as in the *Cart-Pole* problem. Our agent takes discrete actions to maintain a given continuous variable (e.g., latency) at a certain level. Consequently, the agent is rewarded if the actions are leading towards that goal. More specifically, the RFn at time step t is defined as

$$r_t = \begin{cases} 1 & |d_t - d_{tgt}| < \epsilon \cdot d_{tgt} \quad \vee \\ & |c_t - c_{tgt}| < \epsilon \cdot c_{tgt} \\ 0 & |d_t - d_{tgt}| \geq \epsilon \cdot d_{tgt} \quad \vee \\ & |c_t - c_{tgt}| \geq \epsilon \cdot c_{tgt} \\ -100 & \text{if wrong behavior} \end{cases} \tag{5.1}$$

where d_{tgt} is the target latency as defined by the SLA and ϵ is a range of tolerance (e.g., 20%). The tolerance range should not be too large, such as 70%, as the agent may learn to create more replicas than necessary. On the other hand, a smaller tolerance range could result in the agent not receiving enough reward, potentially hindering its learning process (see Section 6.6.4). If the RFn is only defined based on the perceived latency, the agent will take the most obvious action: to keep increasing the number of instances, disregarding the economic impact of such a decision. To keep the number of instances at an adequate level, we also reward the agent if the current CPU usage, c_t, is within a predefined range. If the CPU usage is low, probably the workload can be served using fewer replicas and vice versa. Moreover, the agent is penalized if it incurs in a wrong behavior, such as creating more replicas than necessary or exceeding the perceived latency beyond an allowed upper bound. Notice that defining a wrong behavior could be specified by the SLA, e.g., do not set more than 20 replicas, but typically, this implies domain knowledge when defining such cases, e.g., approximate the CPU utilization target.

[1] https://www.gymlibrary.dev/environments/classic_control/cart_pole/

5.4.1.2 THD-based Agent

Reactive scalers are widely used in cloud applications due to their simplicity and ease of deployment. They use THD-based rules, which depend on the monitored performance metric (e.g., service latency) to perform the predefined scaling actions. Note that one disadvantage of reactive scalers is their questionable effectiveness under bursty workloads [165]. Based on the above Equation (5.1), a THD-based agent can be defined as follows, in which the tolerance range of CPU usage and peak latency is made up of ϵ and respective cpu_{tgt} and d_{tgt} values:

- if the current CPU usage or the peak latency is above their respective tolerance range, the number of VNF instances is increased,
- if the current CPU usage or the peak latency is below their respective tolerance range, the number of VNF instances is decreased.

As with the RL-based auto-scaler, the number of VNF instances to increase/decrease is limited to one per time step.

5.4.1.3 Proportional–Integral–Derivative (PID)-based Agent

The PID agent uses the current $d^{(t)}$ and previous $d^{(t-1)}$ peak latency to decide how to set the number of VNF instances. In particular, it keeps track of a variable $\delta^{(t)}$ at time step t:

$$\delta^{(t+1)} = \delta^{(t)} + \alpha \left(d^{(t)} - d_{tgt} \right) + \beta \left(d^{(t)} - d^{(t-1)} \right) \tag{5.2}$$

If, at the beginning of time step $t + 1$,

- $\delta^{(t+1)} \geq 1$, then the number of VNF instances is increased by 1 and $\delta^{(t+1)}$ is decreased by 1,
- $\delta^{(t+1)} \leq -1$, then the number of VNF instances is decreased by 1 and $\delta^{(t+1)}$ is increased by 1,
- $-1 < \delta^{(t+1)} < 1$ the number of VNF instances is kept the same.

The second and third terms in eq. (5.2) are the integral and proportional terms, respectively. The former tries to keep the peak delay around the target, while the latter tries to proactively react to trends in the latency evolution. There is no differential term in eq. (5.2). Notice that the PID agent only needs the peak latency (current and previous value) as input. In particular, it does not need CPU usage. A PID agent is typically not designed to learn. Good values for its parameters α and β are often determined based on some runs on training data or on linearizing the system to control.

5.5 Experimental Validation

5.5.1 Simulation Setup

In light of the growing amount of research involving RL algorithms in networking, creating an environment is essential to assessing and comparing the performance of the different algorithms. An environment is a process that interacts with the agent and reacts to the agent's actions by transitioning from one state to another. To address this, we created a dedicated environment to facilitate fair and comprehensive assessments of RL algorithms for scaling. This environment is tailored to the specific challenges and characteristics of the scaling problem, with the primary objectives of providing a common ground for evaluation, promoting collaboration within the research community, and tracking the progress of RL algorithms over time.

In the case of scaling, the environment is the system that hosts the different replicas and allows them to process the workload. We developed *DynamicSim*, a simulator that enables the creation of edge-cloud network scenarios. This simulator is based on Simulation of Discrete Systems of All Scales (Sim-Diasca)[2], a general-purpose, parallel, and distributed discrete-time simulation engine for complex systems written in the Erlang language [186]. Erlang facilitates the implementation of large-scale parallel and distributed applications such as the ones tackled in networking. Erlang is a functional programming language based on the actor model, a powerful model for creating highly concurrent, distributed software.

Each actor in this architecture is considered a processing unit, which can communicate with other actors via asynchronous messages. Each actor stores their respective messages that are pending processing. After processing the message, an actor modifies its state, sends more messages, or makes new actors. Due to the actors' lack of shared state or resources and asynchronous communication, this paradigm significantly reduces blocking waits and race situations, two major issues with concurrent systems. Moreover, the actor model facilitates distributed software development since it makes no difference if two actors are running on the same machine or different ones.

Figure 5.3 shows the architecture of the simulator. Sim-Diasca (lower layer) is in charge of synchronizing time between the actors, evolving the system state, sending and receiving messages to and from the controller, and managing the results. Its built-in support for distributed simulation enables deploying a simulation case over multiple computers. Through the base actor model, own-defined models can be created. Therefore, we establish the middle layer called *DynamicSim*, in which we define an actor model for the VNFs, the server, and the load balancer. The traffic generator and the monitor modules act as an interface between the actors in the lower layer and the high-level functions defined in Python. Finally, we design several user-defined simulation cases in the topmost layer, including the one presented in Section 5.4.

However, to be able to interact with a RL agent, *DynamicSim* should follow the OpenAI Gym [51] interfaces. Such interfaces communicate the states, actions, and rewards, providing an abstraction layer between the agent and the environment. In this manner, the agent is unaware of how the environment goes from one state to another, and conversely,

[2] https://github.com/Olivier-Boudeville-EDF/Sim-Diasca.

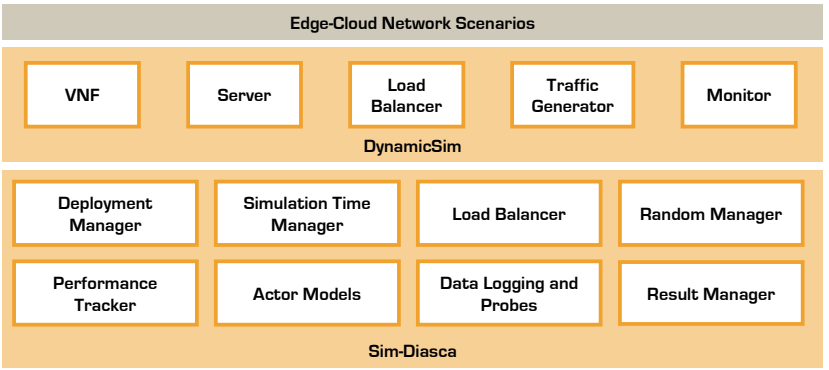


Figure 5.3: Architecture of Sim-Diasca

the environment is unaware of how the agent makes decisions. The interfaces were created using SB3 custom environments, which inherit the methods from Gym Class that provide that abstraction layer. Thus, in each interaction (or step), the agent chooses an action according to a policy and receives the following state from the simulator. The agent is rewarded or penalized based on the RFn depending on the received state. This process is repeated until the agent converges to a policy that maximizes the expected reward in the long term.

In a simulation case, the duration of a time step is user-defined. Within a time step, the actors simulate its functionality, representing the work done over such a long time. After each actor finishes its simulated work, the time manager increases the time step by one, and the simulation goes to the next tick. At the beginning of the simulation, an initial set of actors is generated based on the defined simulation case in Sim-Diasca. This simulation case consists of a server with two replicas and a load balancer between them. The load balancer evenly divides the incoming workload among the created replicas. The system state at step t is defined as $s_t = \langle v_t, c_t, d_t \rangle$, as explained in Section 5.4. Notice that d_t and c_t are real variables, while v_t is an integer variable, which poses the scaling problem into the continuous space. The latency is reported since we need to guarantee that each replica can fulfill the QoS requirement.

Similarly, at step t, the agent takes an action $a_t = -1, 0, 1$, where $a_t = 1$ increases the number of replicas by one, $a_t = -1$ decreases the number of replicas by one, while $a_t = 0$ keeps the number of replicas intact. However, it takes at least one slot to boot or to shut down a replica, depending on the algorithm's decision. No job can be assigned to that particular replica during the boot time. Similarly, the replica resources are released during the shutdown time once its pending jobs are processed.

The communication between the gray and orange modules in Figure 5.3 is based on the publisher-subscriber paradigm implemented in ZeroMQ (0MQ) to transfer structured data based on Google protobuf. The decision-making agent interacts with the simulator (i.e., environment) in regular time ticks. We make the duration of an interval between time ticks the same as the time step mentioned above. In practice, the agent communicates its scaling decision every tick and then waits until the monitor module generates a new report. Once a report is ready, the agent will receive it and evaluate the impact of its decisions.

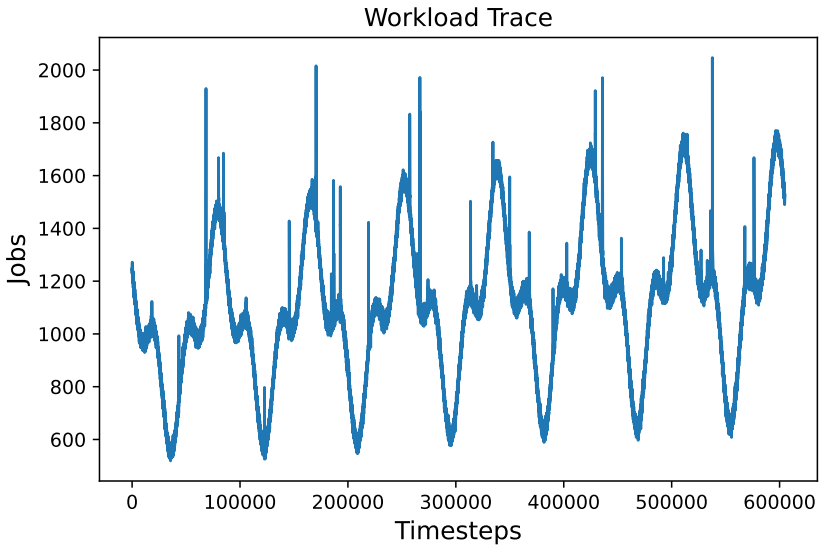


Figure 5.4: Complete workload trace

Table 5.3: Parameters used during the simulation

Parameter	**Value**
VNF Thread Limit	300
Server Processing Capacity	12000
Latency Target (d_{tgt})	20 ms
Cpu Usage Target (cpu_{tgt})	0.75
Tolerance (ϵ)	0.2

Moreover, the traffic generator (workload module in Fig 5.2) follows a known pattern in data centers, as shown in Figure 5.4. Generally, the traffic to a data center is low at night and peaks during working hours. This pattern repeats more or less during the weekday. The traffic is generated using:

$$\begin{aligned} W(t) = \max(0, 300 \cdot (0.9 + 0.1\cos(\pi \cdot T/10)) \cdot \\ (4 + 1.2\sin(2\pi \cdot T) - 0.6\sin(6\pi \cdot T) + \\ 0.02(\sin(503\pi \cdot T) - \sin(709\pi \cdot T)))) \\ + 5N(t) + I(t) \end{aligned} \tag{5.3}$$

where $T = \frac{t}{86400}$, which re-expresses the time t expressed in ticks (i.e., seconds) in T days, the term $\sin(2 \cdot \pi \cdot T)$ introduces a daily pattern and $\sin(6 \cdot \pi \cdot T)$ an 8h pattern. The rest of the terms introduce some randomness so that this pattern does not repeat itself every day. In particular, $N(t)$ is a zero-mean, unit-variance Gaussian random variable and $I(t)$ introduces exponentially decaying impulses on average every 10 000s of average height 200 jobs lasting about 500s.

Every tick, a VNF asks resources to the server accordingly with the jobs they need to process but without exceeding a thread limit. This thread limit is set to 300 jobs, which

represents the capacity of a CPU. Similarly, the server has a capacity of hosting maximum 40 VNFs. Table 5.3 summarizes the remaining parameters used during the simulation.

5.5.2 Results and Discussion

In this section, we show the details of the DQN agent training and the PID agent tuning. We finalize the section by comparing the three auto-scaling approaches.

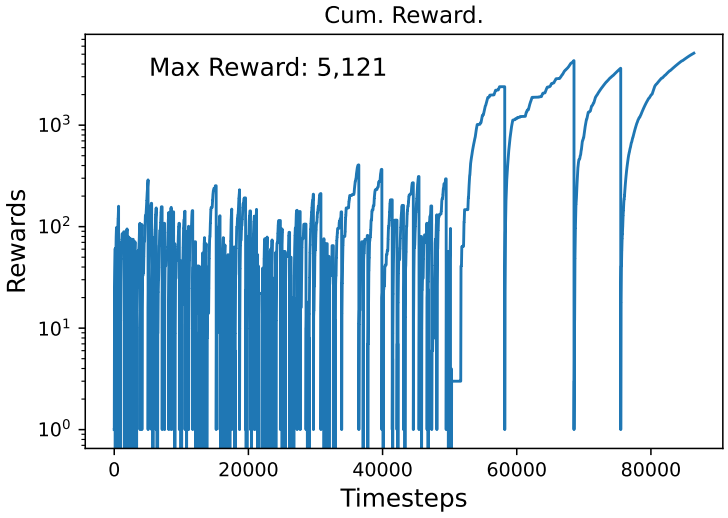
5.5.2.1 Training and Tuning scaling agents

DQN Agent We implemented our DQN agent using SB3 [184], a framework that implements popular RL algorithms in Pytorch[3]. In the definition of the DQN agent, we used the default values given by the SB3 framework. For training our DQN agent, we used episodes. An episode is defined as the number of timesteps until the simulation has to be restarted. Accordingly, we defined two situations where the episode is terminated: when the agent creates more than 20 VNFs or the jobs that are waiting in the queue (overload) are above 200. These two situations represent wrong agent behavior and must be penalized. In such situations, the episode ends, and the simulation is restarted. After the simulation is restarted, the initial scenario is again deployed.

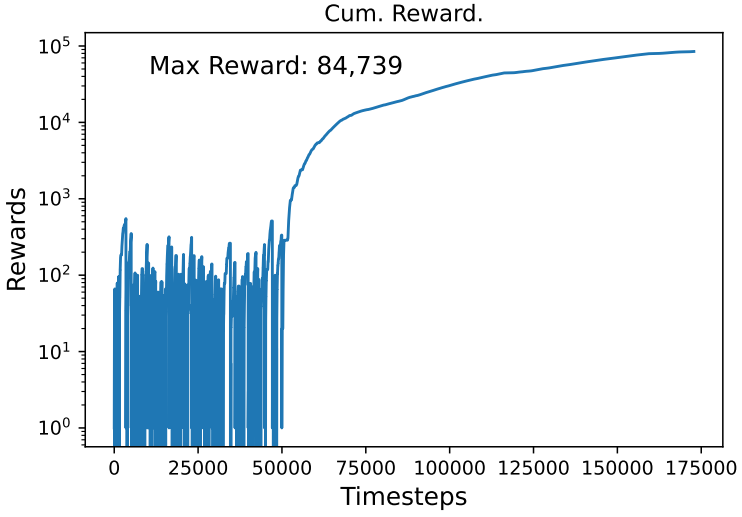
During training, the agent tries to maximize the RFn defined in eq. (5.1). If the peak latency or the CPU usage of the active VNFs are within a tolerance range, the agent is rewarded; otherwise, the agent is not rewarded. If the agent falls into an episode termination case, such as creating more replicas than needed (e.g., 20) or exceeding a specified latency threshold (e.g., 2s), it is heavily penalized, receiving a reward of -100. Typically, the agent is more likely to take actions that produced a reward in the past by taking the actions that led to that situation (exploitation). However, the agent must take random and possibly new actions (exploration) to discover the actions that maximize its reward. Initially, we wanted to experimentally determine how many training steps the agent would require to perform well. Therefore, we trained our DQN agent using different lengths of the trace defined in eq. (5.3). In particular, we used the workload's first (10, 30, 50, 86.4, 172.8 and 259.2K) values as training data. We used the same training data for every training. We noticed that using training lengths shorter than the frequency of the trace (86.4K timesteps, equivalent to one day), the agent cannot converge to a policy that keeps the simulation running.

Figure 5.5 shows the obtained cumulative reward on a logarithmic scale during training. The figure shows the training phase of the agents that obtained better testing results from a set of three runs. As long as the simulation is running, the agent can get a reward if the metrics are within the tolerance range; otherwise, the reward is zero. Therefore, the cumulative reward is continually increasing. However, when the agent falls in the episode termination cases, the agent is hardly penalized, affecting the cumulative reward. As seen from the graphs, the agent needs at least 50K timesteps to keep the simulation running and get higher cumulative rewards. We selected the agent shown in Figure 5.5b for comparison against the baseline approaches. This agent showed faster convergence,

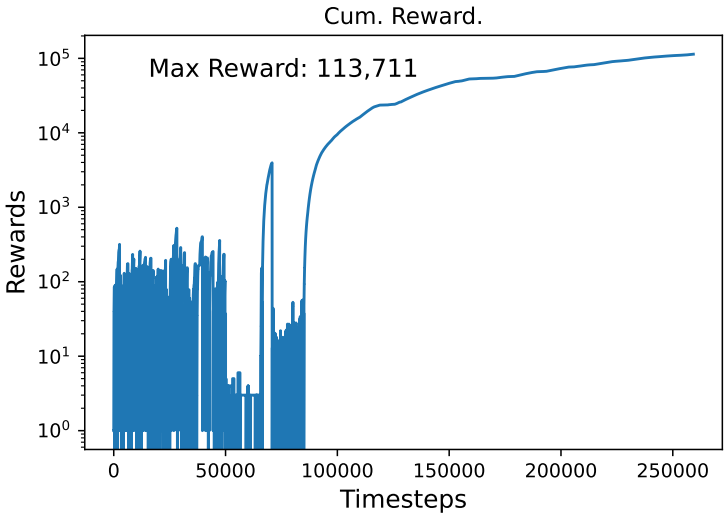
[3] https://pytorch.org/



(a) 86.4K



(b) 172.8K



(c) 259.2K

Figure 5.5: Cumulative rewards from different training lengths of the DQN agent

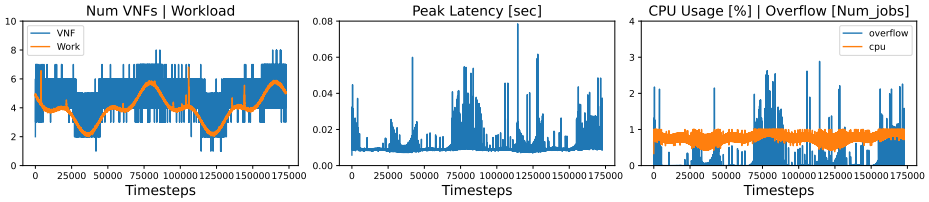
steepest slope in the positive reward area, and the testing phase results were closely similar to the agent in Figure 5.5c, requiring less training time.

PID Agent As stated above, the PID agent tries to keep the peak latency around d_{tgt} = 20ms. The optimal values for its parameters α and β were determined by an exhaustive search. The parameter space (α, β) was sampled by letting α range over the values $\{0.125, 0.25, 0.5, 1, 2, 4, 8, 16\}$ and β over $\{50, 100, 200, 400\}$. Then it was determined for which of all these combinations the latency was the least amount of time above the tolerated upper bound of $(1 + \epsilon)d_{tgt}$, when the PID agent controls the first part of the workload trace, i.e., the training set. It turns out that if the training set spans the first day, the optimal parameters are $(\alpha, \beta) = (16, 200)$, while if the training set spans the first two days, the optimal parameters are $(\alpha, \beta) = (0.25, 200)$. In both these cases, the minimum is broad: relatively small changes in α and β do not alter the number of latency violations drastically so that the choice of α and β is not critical for this context.

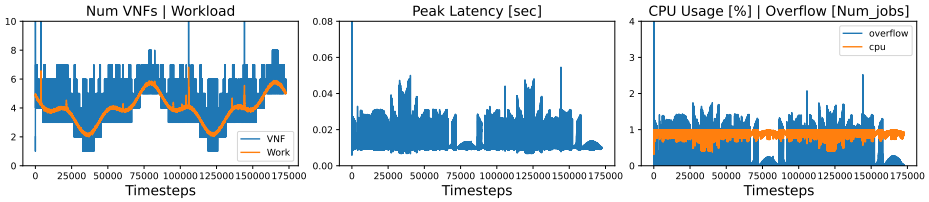
5.5.2.2 Comparison

To test the agents' behavior in unseen workload traces, they were tested using the last 172.8K workload values. It is important to notice that the DQN (and THD-based) agent and the PID agent use different information as input. The former uses the instant peak latency and CPU load, while the latter uses the instant and previous peak latency. Also, the RL agent learns automatically, while the PID agent is manually tuned. Both of these facts mean that care should be taken when comparing the performance of these agents.

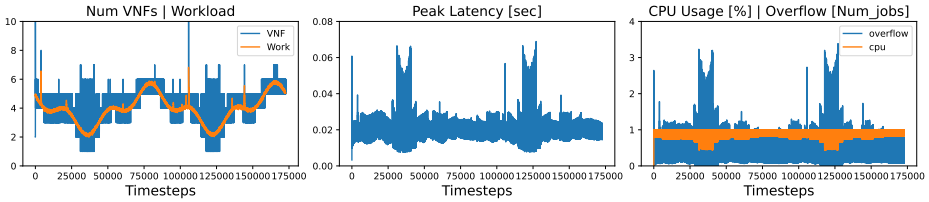
The behavior of each agent is shown in Figure 5.6. On the left of Figs. 5.6a, 5.6b, and 5.6c,



(a) DQN Agent



(b) THD-based Agent



(c) PID Agent

Figure 5.6: Performance of the three proposed agents in terms of the number of created VNFs, peak latency, CPU usage and overflow

Table 5.4: Comparison Results

Metric	Approach	Mean	Std	Min	25%	50%	75%	Max
Number of VNFs	DQN	4.87	0.84	1	5	5	5	8
	THD	4.32	1.25	1	3	4	5	17
	PID	4.06	1.09	1	3	4	5	10
Peak Latency [s]	DQN	0.0095	0.0025	0.0058	0.0088	0.0090	0.0093	0.0785
	THD	0.0153	0.007	0.0058	0.0099	0.0118	0.0195	0.1432
	PID	0.0198	0.0048	0.0033	0.0163	0.0194	0.0228	0.0689

Table 5.5: SLA Violations

Approach	%SLA Violations
DQN	0.69%
THD	12.20%
PID	16.57%

the testing workload trace and the number of created VNFs are shown. Every agent is tested using the same workload trace, which is different from the one used during training or tuning. Generally speaking, the THD-based and the PID agents create less VNFs, on average. However, when facing the low workload valleys (around timesteps 30K and 125K), the PID and the THD-based decrease the number of VNFs to a number that cannot serve the incoming demand, increasing the peak latency and the overflow. To recover from that situation and clear the built-up overflow, both agents are forced to create more VNFs than needed, which is reflected on the maximum amount of created replicas. On the contrary, the DQN agent closely follows the workload trace, despite not having any information regarding it. Around the timesteps mentioned above, the DQN agent creates an amount of VNFs that can serve the incoming demand, keeping the peak latency and the overflow within desirable margins. Nonetheless, the DQN agent does not react well when facing the highest peaks of the workload (around timesteps 75K and 170K). In such situations and for short periods, the agent decreases the number of VNFs below the required, increasing the peak latency and the overflow momentarily, probably anticipating the workload valley.

The middle and right parts of Figs. 5.6a, 5.6b, and 5.6c show the peak latency and the CPU usage and overflow, respectively. All three agents can keep the peak latency within the desired tolerance range, with some temporary exceptions. Table 5.4 gives a quantitative analysis of the behavior by showing the main statistical figures: mean, standard deviation, minimum, maximum, and the most representative quartiles of the peak latency and number of created VNFs. As can be seen, the DQN can maintain a more stable number of created VNFs than the PID and the THD-based agents. However, this is more a secondary effect since all the agents are not designed to optimize the number of replicas. Regarding the peak latency, most of the time, all the agents can keep this metric under the upper bound (24ms). Nonetheless, as shown in Table 5.5, the PID agent violates the upper bound 16.57% of the time while, the THD-based and the DQN are reducing the violations to 12.2% and 0.69%, respectively.

5.5.2.3 Discussion

This section shows the numerical evaluation of three types of agents, namely, DQN, PID, and THD-based. Although they are trained/tuned using different setups (cf. Section 5.5.2.1), all these agents are designed with the same goal of keeping the peak latency at a target of 20ms with a tolerance of 20%. In this sense, we put them on the same table in the above experiment using the identical workload in a simulator engine to quantify their performance in a real system. Note that these results not only reveal the trend of these three agents but also set the foundation for future study on the impacts from different setups beyond instant peak latency, i.e., previous peak latency (used by PID agent) and CPU load (used by both DQN and THD-based agents).

Furthermore, choosing the applicable agent is a task beyond only performance evaluation. It also depends on both business- and operational-related conditions. On the one hand, a multi-tier SLA between stakeholders might show different amounts of marginal penalty among agreed objectives, e.g., a high penalty even when slightly violating the maximum service latency; therefore, the auto-scaler agent may have a higher chance to disregard the number of created VNFs. On the other hand, from the operational point of view, an operator might not have the required hardware to support ML solutions. Therefore,

a DQN agent is ruled out due to its requirement to explore new actions to improve the reward, leading to unpredictable behavior on lower-end hardware.

5.6 Conclusion

More and more, future networking systems are becoming autonomous by using intelligent agents to carry out several MANO operations. One of those operations is scaling. Automating scaling is investigated in this chapter to decide the number of NS replicas required to achieve operational, business, and economic goals for multiple stakeholders. Therefore, it is considered a decision-making and multi-objective problem.

This perspective proposes closed-loop architectures for next-generation networks as they are more likely to adopt data-driven approaches. Using and extending the MAPE-K framework, intelligent (SL- and RL-based) and non-intelligent (threshold- and control-based) scalers can be swiftly integrated as NIFs in future network infrastructures.

In this chapter, we designed and evaluated one autonomous and two automatic scaling agents using known techniques such as heuristics, classic control, and RL. We compared the three agents regarding the peak latency and the amount of created VNFs. Additionally, we discussed the advantages and disadvantages of their implementation.

Chapter 6

Orchestrating Network Intelligence. Challenges and Solutions

The content of this chapter has not been published. An author's version is accessible at:

[187] **P. Soto**, M. Camelo, D. De Vleeschauwer, Y. De Bock, N. Slamnik-Krieštorac, C.-Y. Chang, N. Gaviria, E. Mannens, J. F. Botero, and S. Latré, "Designing, Developing, and Validating Network Intelligence for Scaling in Service-Based Architectures based on Deep Reinforcement Learning," arXiv preprint arXiv:2405.04441, 2024.

Building on the previous chapter's exploration of Network Intelligence (NI) for scaling in service-based architectures through Deep Reinforcement Learning (DRL), this chapter explores the designing, developing, validating, and selecting of such NI solutions. In fact, all the steps mentioned before are part of the responsibilities of the Network Intelligence Stratum (NI Stratum) when managing and orchestrating NI, as explained in Chapter 3. Ideally, future Zero-touch network and Service Management (ZSM) approaches will require a Network Intelligence Orchestrator (NIO) that (i) determines when a Machine Learning (ML) algorithm is ready for deployment, (ii) between two or more algorithms, compares which algorithm performs better for a given task, and specifically for Reinforcement Learning (RL) algorithms, (iii) assess the performance of two or more Reward Functions (RFns). Such challenges are partially solved by aligning with Machine Learning Operations (MLOps) practices. This chapter highlights the challenges introduced by incorporating RL into ML workflows, particularly in model development and operation, noting the existing gap in MLOps frameworks that often focus on Supervised Learning (SL) and overlook RL. Specifically, this chapter aims to solve the following Research Question (RQ):

[RQ_4] **What methodologies can be developed to evaluate and select the most effective Artificial Intelligence (AI)/ML models for networking problems, ensuring optimal performance and automated testing and validation under changing network conditions, enhancing current MLOps frameworks to provide comprehensive lifecycle management of such models?**

6.1 Context and Problem Statement

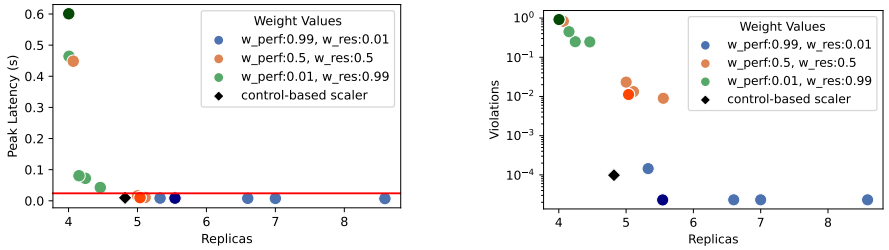
The upcoming Sixth Generation (6G) networks will face challenging and diverse objectives, such as offering Quality of Service (QoS) while guaranteeing Quality of Experience (QoE), infrastructure optimization, and scalable resource utilization. To face these challenges, these networks necessitate advanced automation beyond simplistic rules and heuristics [188]. Recognizing this complexity, AI and ML are acting as main enablers for network automation and, ultimately, for providing the network with self-X (where "X" can stand for healing, configuration, management, optimization, and adaptation) capabilities [189, 13].

By embedding intelligence across all layers and throughout the entire lifecycle of communication services, the telco industry will transition towards an AI-native environment. The primary objective is to enable highly autonomous networks guided by high-level policies and rules, with minimal human intervention. Emphasizing closed-loop operations and leveraging AI and ML techniques, the creation of NI emerges as a pipeline of efficient algorithms for rapid response to network events [79], enabling autonomous network operations, and minimizing human intervention [19].

This transition will offer new opportunities for service provisioning but also introduce technical and business challenges. One notable innovation is the emergence of a NIO that supports the orchestration and management of such AI/ML models [4]. In general, appropriate workflows for NI lifecycle management should be provided [94], which require the alignment with MLOps practices [23]. The existing MLOps frameworks strive to automate and operationalize ML processes, ensuring the delivery of production-ready software. These workflows are designed to be model- and platform-agnostic. However, most MLOps implementations primarily focus on SL, often neglecting comprehensive support for RL algorithms [46].

Therefore, to enable a completely autonomous network operation, the NIO should interpret the high-level policies and rules, e.g., using intent-based management [69], and guarantee that if needed, the NI is ready for composing the Network Service (NS) and posterior deployment. To achieve this, the NIO should trigger the training of the AI/ML models and select the models that best suit the network's operational conditions and requirements. With multiple AI/ML models available to address a networking problem, the challenge lies in determining which model to deploy and identifying the necessary metrics for this selection. Moreover, most of the MLOps implementations predominantly focus on SL, often neglecting comprehensive support for RL algorithms [46]. Incorporating RL introduces additional challenges to the ML workflow in the aspects of algorithmic design, model development, and model operation.

As a motivating example of how the algorithmic design influences the behavior of the RL agent, we found that running the same algorithm with the same RFn and hyperparameters using different seeds can lead to drastically different behavior, depending on the RFn. More specifically, a DRL-based scaling agent is trained and tested in a similar way as in Section 5.5 with a RFn as it will be explained in Section 6.4.2.3. Several cost function weights (w_{perf}, and w_{res}, cf. Section 6.4.2.3) were evaluated to show how the scalers can adapt their behavior based on what they are focused on optimizing. For instance, by optimizing performance over resources, the scaler will more likely pay attention to fulfilling the Service Level Agreement (SLA) while disregarding the number



(a) Peak Latency vs Replicas trade-off

(b) SLA violations vs Replicas trade-off

Figure 6.1: Testing results showing the trade-off between (a) peak latency, and (b) fraction of violations and number of replicas.

of replicas. Moreover, since the RL behavior during training highly depends on the initial weights of the neural network, we trained the scaler several times with different seeds. In each training, the same weight configuration is used.

The results of the evaluation using the testing trace are shown in Figure 6.1. Figure 6.1a shows the behavior of the proposed scalers in terms of the peak latency where the red line identifies the SLA. In this case, the SLA specifies a service's maximum tolerated peak latency (24ms). Similarly, Figure 6.1b shows how many violations are made by the scalers. The green, blue, and orange dots show the different optimization profiles defined by the assigned weights of the cost function. As it can be seen, despite being trained with the same RFn, the different dots (green, blue, or orange) exhibit a scaling behavior that could be centered around a similar working point with few outliers, as in the orange case; or it can be more disperse and difficult to assess, such as the green and blue dots.

On the contrary, once the Proportional–Integral (PI) parameter values are set, its behavior is deterministic. Moreover, for the RFn defined here, the best set of PI parameters is insensitive to the weights. The comparison with the control-theoretic solution showed that the learning behavior of the RL-based scaler heavily depends on the RFn definition, where light variations of the RFn result in different behaviors, while the control-theory based scaler is more deterministic in the scaling decisions. Thus, the RFn must be carefully designed, and its effects on the stability of the RL algorithm must be studied.

Following this motivating example, in this chapter, we follow the MLOps workflow (cf. Figure 6.2) to describe how the NIO should tackle the challenges presented in each of the stages of the NI lifecycle [46]. Properly tackling these challenges, as we intend to do in this chapter, will allow 6G systems to be closer to an autonomous network operation, which will improve the scalability, reliability, and performance of network services leveraged by AI/ML algorithms.

The remainder of this chapter is structured as follows. Section 6.2 lists the challenges in the use of MLOps and its extension to Reinforcement Learning Operations (RLOps) in the networking perspective. Section 6.3 details the characteristics of a benchmark in ML and reviews some efforts in creating a RL benchmarking in networking. The remaining sections go deeper into each of the challenges mentioned in Section 6.2. Section 6.4 details the design of the different RL algorithms as well as their defining RFns and the

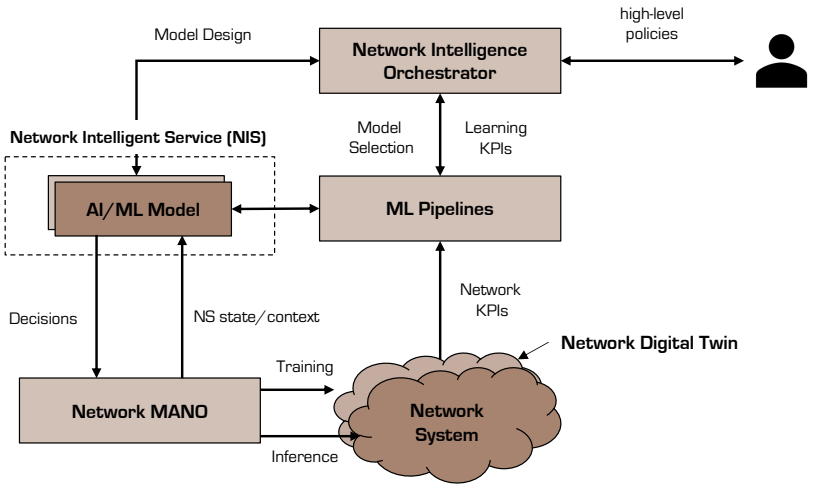


Figure 6.2: Developing and operating NI in AI-native architectures requires interaction among different building blocks; among them, the NI Orchestrator plays a fundamental role in coordinating the remaining elements [4].

environment used for building them. Assuming the availability of multiple RL models for deployment, Section 6.5 proposed a methodology for training, validating and selecting RL algorithms well suited to the scaling problem. Section 6.6 shows the experimental evaluation of the RL algorithms and the associated RFns, including the selection of the best performing agent according to the proposed methodology. Finally, Section 6.7 concludes the chapter.

6.2 Challenges in operationalizing RL-based algorithms

The integration of NI into future network architectures is essential and should be supported by a NI orchestrator for managing intelligent models [4], as proposed in Chapter 3, Section 3.3.2. Effective workflows for NI lifecycle management, aligned with MLOps practices are crucial [23]. Rooted in the “DevOps" concept, the MLOps frameworks embrace a methodology that advocates software automation through Continuous Integration, Delivery, and Deployment (CI/CD), facilitating swift, frequent, and reliable releases [190]. Both DevOps and MLOps align with the infinity model, emphasizing continuous product improvement, making them well-suited for software development, where the emphasis lies in incorporating new updates and features rather than complete product redesign. Figure 6.2 illustrates, on the right, a typical ML workflow.

For developing and operating Network Intelligent Service (NIS), the shown ML workflow should be applied alongside the NIS lifecycle. A key component in an AI-native architecture with autonomous operations is the NIO. This orchestrator is responsible for interpreting the user intentions (e.g., based on intent-based networking [69]), composing a NIS, deploying an appropriate AI/ML model, which guides the NIS behavior, and monitoring that the NIS behavior is as expected. To be able to do that, the NI orchestrator implements (on its own or via a third-party service) an ML workflow.

This workflow starts with the definition of the requirements, typically derived from the problem at hand. For instance, scaling can be solved through SL techniques, which implies the collection, curation, and maintenance of a historical database with historical scaling decisions and values of the Key Performance Indicators (KPIs) collected from the managed entity. However, scaling can also be solved through RL techniques (as it will shown later in this chapter), which implies the preparation of a sandbox [24] in which the RL agent can freely interact with a digital replica [25] of the managed system to train a scaling policy.

Once the ML model is selected, the model training and fine-tuning occur thanks to the MLOps framework, including tuning the model's architecture and hyperparameters (e.g., learning rate, activation functions, and regularization methods). Following model construction, validation is performed by introducing unseen data to enhance generalization. Upon achieving a predefined performance, the NIO deploys the model as a Network Intelligent Function (NIF) inside the NIS. After deployment, the NIO continuously monitors the performance of the NIS. Typical quality metrics of ML models are accuracy, loss value, and average return after an epoch, among others. Nevertheless, the NIO could potentially be supported by metrics gathered from the network to assess the quality of the ML model. Continuous monitoring is vital due to potential data and model drift, ensuring service quality. If the model fails to meet quality standards, refinement is undertaken, restarting the cycle.

The existing MLOps frameworks strive to automate and operationalize ML processes [191], ensuring the delivery of production-ready software. The workflow is intended to be model- and platform-agnostic. However, most of the MLOps implementations predominantly focus on SL, often neglecting comprehensive support for RL algorithms [46, 192]. Incorporating RL introduces additional challenges to the ML workflow, encompassing four aspects.

Firstly, the problem to be addressed must be formulated as a Markov Decision Process (MDP), where the learning goal of a RL agent is to find an optimal policy through the use of a RFn. By interacting with an environment, the RL agent gathers states and rewards, facilitating the iterative approximation or computation of the expected long-term reward, enabling the selection of optimal actions at each step. Unlike the intrinsic "labels" in SL, rewards reflect the anticipated behavior of agents. Consequently, the design of the RFn must be done carefully and sometimes requires specialized considerations.

Training RL algorithms in production networks is challenging due to the trial-and-error nature of the algorithms, leading to possible network vulnerabilities. To ensure the safety of RL algorithm training, an offline approach is recommended [193]. In this offline setting, the RL agent accumulates experiences by exploring actions randomly, allowing it to learn without impacting network operations. To facilitate this learning process, Network Digital Twins (NDTs) are proposed to provide a controlled, reliable, and easily accessible simulation environment [25], enabling RL algorithms to safely explore new actions. Consequently, a parallel branch is required to support this learning type. In the primary branch, decisions from an already trained agent are implemented in the production network, while algorithms still undergoing training can introduce their outcomes in the NDT [4].

A third aspect challenging the full support of RL in MLOps frameworks is the reproducibility issue. It has been found that minor implementation details can consider-

ably impact performance, sometimes surpassing the disparity between various algorithms [194, 195, 196]. Best practices in the RL community include the use of standard agent-environment interfaces [51], RL implementations [184] and full transparency in reporting the settings used when training RL algorithms. Unfortunately, such practices are not fully adopted in the growing body of literature of RL applications in networking [197].

Finally, a key feature of the NIO involves selecting and validating models. In a dynamic environment where various Network Operators (NOs) may develop their own AI/ML/RL models, multiple models could potentially be deployed for a single function. Hence, the orchestrator is crucial in determining which available model should be deployed, preventing conflicts, and optimizing resource utilization. Unfortunately, existing implementations of MLOps frameworks lack this feature, as their workflows are designed for a single model or perform manual model selection based solely on the learning metrics (e.g., loss, accuracy, or reward). However, as we will show in the following sections, RL algorithms should also be validated in terms of the performance of the task they are designed to solve. Moreover, to be able to compare different RL algorithms for scaling, a benchmark scenario is needed. Unfortunately, such a benchmark is not available.

Summarizing, deploying and managing RL-based NI introduces additional challenges to the ML workflow, particularly in algorithmic design, model development, and model operation. First, the networking problem should be framed as a decision problem to facilitate the application of RL techniques. Second, RL depends on a RFn, whose design is crucial and often problematic due to RL's tendency to overfit to the provided rewards [198, 199], leading to unintended outcomes. Additionally, training RL algorithms online is unpractical because exploration-related randomness can destabilize production networks. Finally, including networking evaluation of the algorithm performance during algorithm selection is essential to ensure autonomous network operation.

In the following sections, we go deeper into each of the challenges mentioned in this section. Specifically, Section 6.3 details the characteristics of a benchmark in ML and reviews some efforts in creating a RL benchmarking in networking. But mainly, this chapter focuses on the design and development phases of the workflow shown in Figure 6.2. Section 5.4 already posed the scaling problem as a MDP, setting the requirements for the environment and algorithm design, which are shown in Section 6.4. Section 6.5 focuses on the development part as it explains the difficulties when training and validating RL algorithms. Additionally, it proposes a methodology for performing model selection, an important step before its deployment in production networks.

6.3 Existing Solutions for RL-benchmarking in Networking

In ML, a benchmark refers to a standardized set of tasks, datasets, or performance metrics used to assess and compare the performance of different algorithms or models. The goal of a benchmark is to provide a common ground for evaluating the effectiveness of various approaches in solving specific problems. Benchmarks are widely used in ML to measure algorithmic advancements, facilitate fair comparisons, and track the state-of-the-art in different subfields.

The main elements in a ML benchmark include well-defined tasks, carefully curated datasets, metrics to measure performance, evaluation protocols, and often baseline models for reference. Benchmarks define specific tasks or problems that algorithms are expected to solve. These tasks range from classification and regression to more complex challenges like image recognition and natural language processing. However, such tasks are related to SL approaches. Regarding RL, typical tasks are related to continuous or discrete control problems.

Concerning the datasets, benchmarks often include standardized datasets for training, validation, and testing. These datasets are carefully curated to represent the diversity and complexity of real-world scenarios, ensuring that algorithms are evaluated under realistic conditions. While SL approaches benefit from a wide range of datasets depending on the task [52, 53, 54], RL are evaluated using standard environments, where OpenAI Gym [51] is the most recognized. Three main types of environments are identified. Environments with discrete action spaces are evaluated using Atari video games, including Pong, Breakout, Space Invaders, Seaquest, and more. Moreover, for continuous action spaces, OpenAI Gym integrated the MuJoCo physics engine, where the action input applies torque on the joints of a humanoid robot learning to walk. Less complex environments include the classic control ones, which are stochastic in their initial state and are considered the easiest to solve by a RL algorithm.

Additional elements in a ML benchmark include the metrics used to quantify their performance. Common metrics include accuracy, precision, recall, F1 score, mean squared error, or area under the Receiver Operating Characteristic (ROC) curve, depending on the nature of the problem. Moreover, benchmarks also define the protocols for training and evaluating models. This includes guidelines on splitting the dataset into training and testing sets, whether cross-validation is used, and any specific considerations for model evaluation. Finally, benchmarks often include baseline models or algorithms that provide a reference point for comparison. These baselines help establish a performance threshold against which new algorithms can be measured.

Nevertheless, reproducibility of RL is hard to achieve, as shown in [196], where the authors performed a thorough study by implementing and comparing state-of-the-art DRL algorithms in the benchmarks provided by OpenAI Gym such as Half-Cheetah, Hopper, and Swimmer. They found out that different implementations of the same algorithm with the same hyperparameters can lead to distinct behaviors. Moreover, due to the lack of meaningful metrics and more stringent standardization in experimental reporting, discerning the significance of improvements over the previous state-of-the-art becomes challenging. Due to variance across trials, multiple runs with different random seeds are necessary for reliable comparisons. Proper significance testing, including bootstrapping and power analysis, is vital to validate performance gains. The key to reproducibility lies in thoroughly reporting hyperparameters, implementation details, and experimental setup. Without this crucial information, progress in RL research is significantly hindered.

Automated RL has emerged as a field where various algorithms aim to automate different aspects of the RL pipeline [192]. Initially, hyperparameter optimization algorithms, such as Bayesian optimization [200] and random/grid search [201], are applied to fine-tune RL algorithms. Next, methods like network architecture search [202] are used to optimize the architecture of the Deep Neural Network (DNN) involved in DRL algorithms. Finally,

meta-learning algorithms [203] and reward shaping are employed to learn better reward functions, improving the overall efficiency of RL. However, these techniques typically focus on fine-tuning a single RL model, selecting the best one guided by a single metric: the accumulated reward. As we will demonstrate in Section 6.6, the accumulated reward is not the sole metric for evaluating RL algorithms. For instance, several algorithms trained under the same conditions showed similar accumulated rewards, yet differences in their performance emerged when evaluated in scaling setups.

More focused on methods for continuous control, the authors in [195] examine the general variance of the algorithm's performance depending on the initialization of their hyperparameters in policy gradients for continuous control and provide guidelines on reporting novel results against the baseline methods. By running extensive experiments, the authors found that state-of-the-art RL algorithms frequently exhibit significant variations due to differences in hyperparameter configurations. Moreover, outcomes in diverse continuous control domains do not consistently align, as demonstrated in the experimental results involving the Hopper and Half-Cheetah environments. Half-Cheetah's performance is more susceptible to fluctuations from hyperparameter tuning, whereas Hopper does not exhibit the same sensitivity. According to their analysis, the under-reporting of hyperparameters may lead to unfair baseline comparison in continuous control environments.

In networking, standard implementation of RL algorithms is still in its infancy. An initial step involves establishing an abstraction layer that enables the representation of a network environment as a Gym environment. This layer exposes simulation entities' states and control parameters for the agent's learning objectives. An example of such an abstraction layer is shown in [204], where ns-3 [205], a known network simulator whose results are widely accepted by the networking research community, is used as a network environment. The main components of *ns3-gym* are two communicating processes that exchange information between frameworks using Google Protocol Buffers (Protobuf) [156] and ZeroMQ (0MQ) sockets. As a case study, the authors presented two examples of a cognitive radio transmitter that adapts its channel access patterns accordingly with a periodic interferer.

An extension of the previous work is made in [206], where they proposed a shared memory implementation to replace the Protobuf - 0MQ combination as a more efficient, high-speed way of communication. Moreover, thanks to their implementation, the ns-3 simulator can communicate with popular ML frameworks such as PyTorch and TensorFlow, including Deep Learning (DL) algorithms as well. Like the previous work, the authors showed the viability of *ns3-ai* by predicting the channel quality information in cellular systems to improve the channel estimation and the communications between users and base stations in 5G new radio.

More recently, Bonati et al. [43], presented *OpenRAN Gym*, an Open Radio Access Network (O-RAN) compliant toolbox that combines several software frameworks for Radio Access Network (RAN) data collection and control. AI algorithms are deployed as xApps in the near real-time RAN intelligent controller, trained in Colosseum, the world's largest wireless network emulator [207], and later can be seamlessly transferred to heterogeneous platforms. Utilizing the presented toolbox, the authors illustrated the application of two xApps for managing a large-scale O-RAN deployed on Colosseum. Furthermore, they highlighted the seamless transition of *OpenRAN Gym* solutions and experiments from

the Colosseum to real-world platforms, including the Arena testbed, the POWDER, and COSMOS platforms within the PAWR program.

We just described common benchmarks in RL and reviewed the current body of literature regarding the integration of networking and RL. On the one hand, the works showing RL techniques for scaling are opaque in reporting the work done at fine-tuning their RL agents. They lack a detailed overview of the parameters used in their agents, only show the results of their best-performing agent, or do not specify how many runs of the same algorithm they performed, a practice highly discouraged by RL practitioners since it hampers the reproducibility of such techniques and further development.

On the other hand, there are growing efforts in designing frameworks that allow a common and standard way of training and evaluating RL algorithms for networking. Nevertheless, such efforts focus more on RAN implementations than on cloud-like environments. The ns-3 simulator puts more effort into simulating actual devices but lacks support for service-based network architectures. Several extensions are available but tailored to specific cases, such as Field Programmable Gate Arrays (FPGA) implementations for 5G networks [42] and O-RAN solutions [41], where the service management and orchestration module, in charge of the xApps lifecycle operations (scaling being one of them), is out of the scope of their model. Moreover, the work proposed by *OpenRAN Gym* is interesting since it created a platform to train and evaluate ML and RL models using large-scale emulators. However, since their interfaces do not follow the OpenAI Gym abstractions, it limits the benchmarking of RL algorithms and compatibility with future RL frameworks in other network domains.

With this work, we aim to tackle the deficiencies mentioned above. Concerning the reproducibility issue, we thoroughly describe our training and testing procedures. Moreover, we follow the standard implementations of stable baselines, which already deliver the tuned hyper-parameters as intended in their original versions. Furthermore, we open-source our environment and abstraction layer for scaling in service-based network architectures, similar to the work done in [183]. Though, being a simulator, *DynamicSim* avoids the impracticality of owning or renting a Kubernetes (K8s) cluster, accelerating the learning curve of new RL in networking practitioners. We hope this work will foster research in the applicability of RL in networking and further adoption in real-life environments.

6.4 Model Design: DRL algorithms for scaling

Once the problem has been defined, we can proceed to the next step in the design phase. Since the data for RL approaches is obtained through continuous interaction with an environment, we skip the data preparation step. Then, we can design the model that will solve the problem. This section explores the intricacies of designing DRL algorithms tailored explicitly for efficiently scaling resources. Based on the problem constraints elaborated in Section 5.4, we discuss the nuanced aspects of algorithmic design, elucidating key principles and methodologies essential for achieving proper lifecycle management of this type of NI in service-based architectures.

6.4.1 DRL Algorithms Tailored to the Scaling Problem

Given the properties of the scaling problem, there are two known DRL algorithms that are suitable, namely, a Deep Q-Network (DQN) [176] and Proximal Policy Optimization (PPO) [208] algorithm. DQN is a type of off-policy, value-based algorithm that combines Q-learning with deep neural networks to estimate the value of states or state-action pairs. Being a value-based algorithm, DQN learns the value of each action in a given state to later select the actions that maximize the expected reward over time. Through repeated interactions and learning from the experience replay buffer, the DQN can learn an optimal policy to perform well in complex tasks. Still, it may struggle with high-dimensional state spaces. It has been successfully applied to various tasks, including playing Atari games and controlling robotic systems [209].

On the other hand, PPO is an on-policy, policy-based algorithm based on policy gradient. Policy-based RL directly parameterizes the policy mapping states to actions without explicitly learning a value function, offering greater flexibility and robustness in exploring the action space. However, more samples may be required to converge to an optimal policy than value-based methods. PPO tries to address the limitations of previous policy gradient algorithms such as Trust Region Policy Optimization (TRPO) by striking a balance between making significant updates to the policy (to improve performance) and ensuring that the updates are not too large, which could lead to instability or catastrophic changes. PPO achieves this by updating the policy with a "clipped" objective function.

6.4.2 Reward Functions (RFns)

We designed three RFns to guide the agents' learning. The goal in proposing multiple RFns is twofold. On one side, we want to highlight that the RFn definition is among the most difficult aspects in RL. Here, we have three different RFns for the same problem, and though they seem to be logical and make sense, not all of them yield good results, as shown in Section 6.6. Conversely, the exploratory work performed in this study may help uncover a broader spectrum of potential solutions and can lead to more robust and adaptive learning.

6.4.2.1 Reward Function 1 (RFn1): Inspired from Cart-Pole

This RFn was detailed previously in Section 5.4.1.1, Equation 5.1, but we provide a concise overview here for completeness. More specifically, the RFn at step t is defined as

$$r_t = \begin{cases} 1 & |d_t - d_{tgt}| < \epsilon \cdot d_{tgt} \quad \vee \\ & |c_t - c_{tgt}| < \epsilon \cdot c_{tgt} \\ 0 & |d_t - d_{tgt}| \geq \epsilon \cdot d_{tgt} \quad \vee \\ & |c_t - c_{tgt}| \geq \epsilon \cdot c_{tgt} \\ -100 & \text{if wrong behavior} \end{cases} \tag{6.1}$$

where d_{tgt} is the target latency as defined by the SLA and ϵ is a range of tolerance (e.g.,

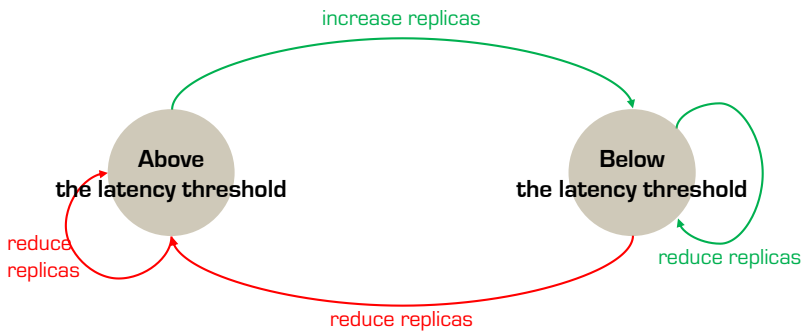


Figure 6.3: Markov Reward Process for auto-scaling

20%). If the RFn is only defined based on the perceived latency, the agent will take the most obvious action: to keep increasing the number of instances, disregarding the economic impact of such a decision. To keep the number of instances at an adequate level, we also reward the agent if the current Central Processing Unit (CPU) usage, c_t, is within a predefined range. If the CPU usage is low, probably the workload can be served using fewer replicas and vice versa. Moreover, the agent is severely penalized if it incurs in a wrong behavior, such as creating more replicas than necessary or exceeding the perceived latency beyond an allowed upper bound. Notice that the CPU utilization target, c_{tgt}, is not normally specified by the SLA and must be approximated.

6.4.2.2 Reward Function 2 (RFn2): A Markov Reward Process

A Markov Reward Process (MRP) is a mathematical framework used to model and study the behavior of a system that involves stochastic transitions between states and yields rewards over time. MRPs are used to formalize problems where an agent interacts with an environment, and the agent receives rewards for being in certain states. This way, MRPs could be directly applied to RL setups. A MRP is defined by mainly three concepts: the states, the transitions, and the rewards. Notice that it is not necessarily the case that the states of the MRP are the same as in the MDP.

Particularly for the scaling problem, we define two states: the process exhibits a latency above the specified threshold, or the process shows a latency below (or at) the defined threshold. The agent dictates the transitions between these two states with its actions. For example, it can be that the process is below the latency threshold, but the agent decides to remove a replica, which takes the process to the state of being above the latency threshold. Then, the algorithm designer can establish what actions led to a good outcome and which did not.

Figure 6.3 shows an example of such MRP. In green are depicted the actions that we consider good because they lead toward the goal we want to achieve, i.e., fulfilling the SLA with the least amount of replicas possible. In contrast, depicted in red are the actions that push us away from that goal. More specifically, for both agents, we only reward the good actions. We noticed that by using positive reinforcement, the agents could show a better performance. Then, the RFn we used was defined as follows.

- If the process was in state *Above*, and the agent's action was to increase the number of replicas, which led to the process being in the *Below* state, the agent receives a reward of 1. In this case, the agent is encouraged to increase the number of replicas only to fulfill the SLA.
- If the process was in state *Below*, and the agent's action was to decrease the number of replicas, which led the process to keep being in the *Below* state, the agent receives a reward of 1. In this case, the agent is encouraged to use resources efficiently when the SLA is fulfilled.
- All the other combinations of states and transitions are not rewarded nor penalized.

6.4.2.3 Reward Function 3 (RFn3): A Multi-Objective Function

This RFn aims to fulfill the agreed SLA with the minimum amount of replicas. Therefore, in every time step, the agent pays an immediate cost depending on how good or bad the action it took is. The cost of taking action when the environment moves from one state to another can be defined as a weighted function, including the following contributions.

- If the agent cannot fulfill the SLA, it incurs a performance cost c_{perf}, with an associated w_{perf}, which is paid every time the perceived latency d exceeds a maximum tolerated latency, $(1+\epsilon)\cdot d_{tgt}$. The cost is zero otherwise.
- If the agent must deploy a new replica, a resource cost c_{res} is paid, with an associated w_{res}; this can be seen as a rental cost in cloud environments or the consumed energy of the replica while it is running.

These two contributions are combined into a weighted function, shown in Equation 6.2, where the respective non-negative weights define an optimization profile, $w_{perf} + w_{res} = 1$. The weights (w_{perf} and w_{res}) multiply an indicator function $\mathbb{1}\{\cdot\}$ that varies between 1 and -1 depending on whether or not a condition is met, as indicated in Equations 6.5 and 6.6. For instance, if the perceived latency exceeds a maximum tolerated value, the indicator function is 1; otherwise, the indicator function is 0. Conversely, if a new replica is instantiated, the indicator function is 1, or -1 if the replica is removed. Finally, the RFn is defined as the negative of the cost function.

$$r = -c_{total} = -(c_{perf} + c_{res}) \tag{6.2}$$

$$c_{perf} = w_{perf} \cdot \mathbb{1}\{perf\} \tag{6.3}$$

$$c_{res} = w_{res} \cdot \mathbb{1}\{res\} \tag{6.4}$$

$$\mathbb{1}\{perf\} = \begin{cases} 1 & d_t > (1+\epsilon)\cdot d_{tgt} \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

$$\mathbb{1}\{res\} = \begin{cases} -1 & a = \text{remove replica} \\ 0 & a = \text{maintain replica} \\ 1 & a = \text{add replica} \end{cases} \tag{6.6}$$

Table 6.1: Optimization profiles used in RFn3.

Optimization Profile	w_{perf}	w_{res}
1	0.5	0.5
2	0.01	0.99
3	0.99	0.01

We used different optimization profiles for this RFn by assigning weights. By giving equal weights, the agent should learn policies that balance the creation of replicas while not exceeding the latency threshold. On the other hand, by assigning more weight to the resource cost, the agent should learn policies that limit the creation of replicas, disregarding the impact on latency violation. Conversely, if more weight is assigned to the performance cost, the agent should learn policies that encourage the creation of replicas to minimize the latency violation. Table 6.1 shows the optimization profile we used in this study.

6.4.3 Environment

The environment was completely described in Section 5.5.1. We briefly detailed it again in this section for completeness.

We developed *DynamicSim*, a simulator that enables the creation of edge-cloud network scenarios. This simulator is based on Simulation of Discrete Systems of All Scales (Sim-Diasca), a general-purpose, parallel, and distributed discrete-time simulation engine for complex systems written in the Erlang language [186]. Figure 6.4 shows the architecture of the simulator. Sim-Diasca is in charge of synchronizing time between the actors, evolving the system state, sending and receiving messages to and from the controller, and managing the results. Through the base actor model, own-defined models can be created. Therefore, *DynamicSim* defines an actor model for the replicas, the server, and the load balancer. The traffic generator and the monitor modules act as an interface between the actors in the lower layer and the high-level functions defined in Python through the sub-pub communication schema developed by 0MQ.

For effective interaction with a RL agent, *DynamicSim* must adhere to OpenAI Gym [51] interfaces, facilitating communication of states, actions, and rewards. This creates an abstraction layer between the agent and its environment, keeping each entity oblivious to the internal workings of the other. Utilizing Stable-Baselines3 (SB3) custom environments, which extend Gym Class methods, this setup allows agents to select actions based on their policy, receive new states from the simulator, and earn rewards determined by the RFn. The cycle continues until the agent optimizes its policy for maximum long-term rewards.

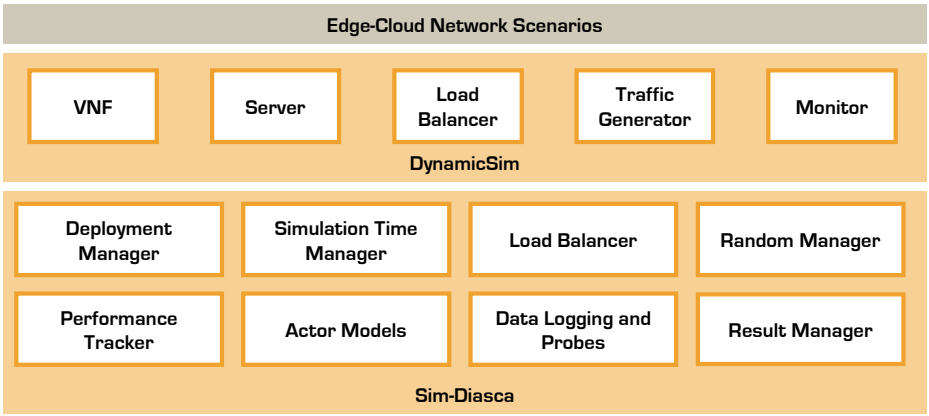


Figure 6.4: Architecture of Sim-Diasca

6.5 Training, Validating, and Selecting DRL Algorithms for Scaling: A Methodology

In the previous sections, we detailed the influence of scaling computing resources in service-based architectures on system requirements, shaping the learning objectives for RL agents (cf. Section 5.4). Building upon these requirements, Section 6.4 delved into such agents' algorithmic and environmental design. Aligned with the infinity model, this section focuses on the development phase. Assuming the availability of multiple RL models for deployment, we incorporate best practices for algorithm comparison and selection, thereby expanding current MLOps frameworks.

6.5.1 Training

Once the appropriate RL algorithms and RFns tailored to the problem have been selected, the next step in their lifecycle is to develop the agent. This is commonly known as training. Following the recommendations from the O-RAN alliance, RL algorithms should be trained offline [193]. Offline training in networking can be performed in a NDT [25], where the outcomes of the untrained agents can be safely injected without compromising the stability of the production network [4]. The environment of Section 6.4.3 can serve as a NDT for the proposed use case.

A major challenge in the RL community is that minor implementation details can considerably impact performance, sometimes surpassing the disparity between various algorithms [194]. Ensuring the reliability of implementations employed as experimental baselines is paramount. Otherwise, comparing novel algorithms to unreliable baselines may result in inflated estimates of performance enhancements. Thus, standard, open-source frameworks that provide reliable implementations of RL algorithms are recommended [184, 210].

Regarding the implementation of the algorithms presented in Section 6.4.1, for this study, we use SB3, a popular framework that provides a set of reliable implementations of RL

algorithms in PyTorch[1]. We use the default architecture and default hyper-parameters given by stable baselines for both algorithms. Both algorithms use two fully connected networks with 64 units per layer. While actor and critic are shared for PPO to reduce computation, DQN has separate feature extractors, one for the actor and one for the critic since the best performance is obtained with this configuration.

Additionally, since RL's reproducibility can be far more difficult than expected [195, 196], given their random initialization of the parameters, among others, Colas et al. [211, 212] suggest evaluating several seeds of DRL algorithms in an offline manner. In this offline evaluation, the algorithm performance after training is independently assessed, usually using a deterministic version of the current policy. We call an *agent* an algorithm - RFn - seed combination.

Consequently, to compare the performance of the different RL algorithms, we follow the guidelines suggested by Colas et al. [211, 212]. In their paper, the authors suggest (i) obtaining the measure of performance of every agent to plot the learning curves of the algorithms, (ii) performing statistical difference testing, (iii) comparing the full learning curves not only comparing the final performances of the two algorithms after t steps in the environment.

Considering those recommendations, we extend the current methodological proposals for lifecycle management of RL applications in service-based architectures to support algorithmic comparison and selection. We summarize this extension into the methodology shown in Figure 6.5.

According to Colas et al. [211], statistical tests are necessary to determine the number of random seeds required to confidently claim a performance difference between two or more RL algorithms. Like Colas et al. [211], we began this analysis using five random seeds, which involved creating and training five distinct agents. Each agent is trained using episodes, where an episode is defined as the number of steps until the simulation must be restarted or a maximum number of steps is reached. Accordingly, we determined two situations where an episode is terminated: when the agent creates more replicas than needed or lets the latency go above an inadmissible threshold. These two situations represent an undesired agent behavior and must be penalized. In such situations, the episode ends, the agent receives -100 of reward, and the simulation is restarted. Typically, the agent receives a reward that oscillates between -1 and 1 (cf. Section 6.4.2). However, by applying a penalty of -100, we aim to send a strong signal to the agent, indicating that this behavior is undesirable and reinforcing that it should be avoided. After the simulation is restarted, the initial scenario is again deployed.

The input data used to train the algorithms is fully described by Equation 5.3 (cf. Section 5.5.1). Since each simulation step represents the events of one actual second, we selected one-hour episodes. The chosen traffic trace follows a repetitive pattern with a one-hour period (see Figure 5.4). This setup ensures that the agents can explore at least one full cycle. Longer training data can be used, but this will also increase the training time. Thus, an episode's maximum number of steps is 3600.

Notice that the episode length may vary during training since, at the beginning of the training, the agent is expected to take random actions, which can easily lead to episode termination due to undesired behavior. After the agent is trained during N episodes, we

[1] `https://pytorch.org/`

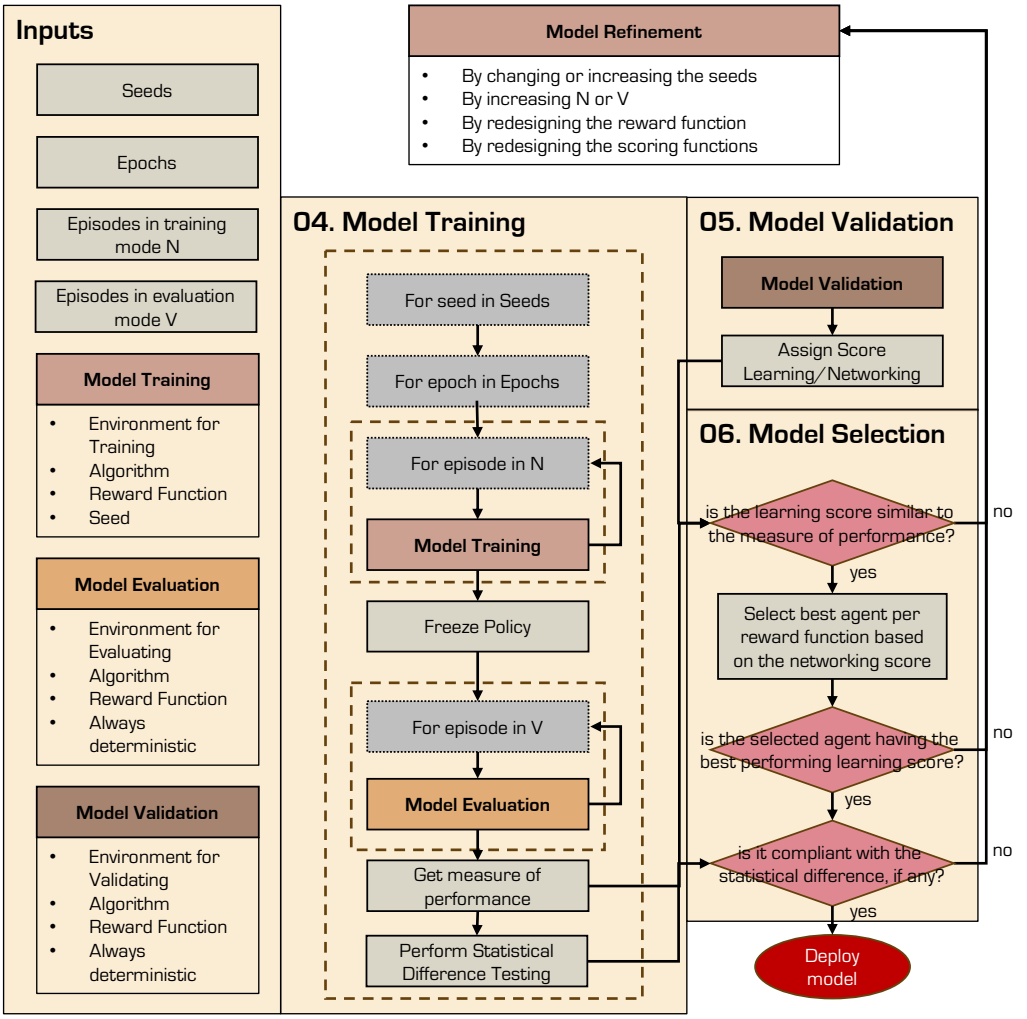


Figure 6.5: Proposed Methodology

Table 6.2: Parameters used in the experimental evaluation.

Steps per episode	3600
Episodes in training mode N	24
Episodes in evaluation mode V	12
Epochs E	10
d_{tgt}	20 ms
c_{tgt}	75%
ϵ	20%

freeze its policy and evaluate it during V episodes. Following the logic of SL approaches, we split the workload trace into two, one for training and another for evaluation, aiming to test the generalization capabilities of the algorithms to unseen data. The training trace uses the workload's first 432×10^3 values (i.e., first 5 days), while the evaluation trace uses the last 172.8×10^3 values (i.e., the last 2 days). Training and evaluating the agent is considered an epoch; we repeat the process during E epochs.

Generally, the values of N, V, and E should be selected by trading off training time and the agent's performance. The agent's performance should be evaluated as soon as possible, which implies selecting a lower training time, i.e., lower N, but ensuring enough training is reflected in improving the agent's behavior. Moreover, a larger V gives a better estimation of the true agent's performance but, at the same time, slows down the training. Following this trade-off, we selected the values for N, V, and E for all RFns and RL algorithm combinations. The values are shown in Table 6.2.

At the end of an evaluation episode, we record the accumulated reward of the episode. Then, the performance per epoch is the average earned reward over the V episodes. To provide a fair comparison of the performance of the algorithms among the different RFns, all the RFns must be in the same range. Unfortunately, that is not true for the proposed RFns. However, a minimum and a maximum achievable reward can be calculated for every agent, depending on the episode duration.

Suppose an agent makes good decisions to avoid falling into episode termination cases. In that case, the maximum reward possible is the duration of a full episode times the maximum reward. A similar analysis can be done with the minimum achievable reward. Table 6.3 shows the range variation in each RFn, where the underscore in RFn 3 indicates the optimization profile, c.f., Table 6.1. Having the minimum and maximum range of the reward, we apply a min-max scaler so the agents' performance falls within the same range.

Once each agent's and run's performance was calculated, we performed statistical difference testing as a main way to compare their performance. In particular, we completed the Welch t-test on the data since it was more robust than other tests to the violation of their assumptions [212].

Table 6.3: Minimum and Maximum reward possible in each of the reward functions.

Reward Function	Min	Max
RFn 1	0	3600
RFn 2	0	3600
RFn 3_1	-3600	1800
RFn 3_2	-3600	3564
RFn 3_3	-3600	36

6.5.2 Validation

Once trained and evaluated, the performance of all the scalers is assessed. This performance can be evaluated from two perspectives: how well the agent learns and how well the agent performs the task at hand. For this purpose, we tested the learned policy in deterministic mode after the E epochs of training by letting the agent run a validation episode of 172.8×10^3 steps. We gathered the accumulated reward the agents received at the end of the validation episode and the main statistical figures of the number of created replicas and the latency. Depending on the obtained values, we score all the agents using Equations 6.7 and 6.8.

$$\text{score learning} = \frac{\text{normalized accumulated reward}}{\text{normalized episode length}} \tag{6.7}$$

Equation 6.7 scores the agents regarding the learning perspective. The episode termination cases should be avoided if the agent is well-trained. If these cases do not occur, the agent maintains the simulation running, and the episode length is the largest it could be, i.e., 172.8×10^3 steps. Then, using a min-max scaler, the normalized length of a testing episode should be 1, and the score should be approximately the same as the normalized reward achieved during training (see the y-axis of Figure 6.6). Suppose the agent falls into the episode termination cases. In that case, the validation episode is finalized before time, where the normalized episode length is lower than one, forcing the score to be higher than one. In this case, the agent can be discarded since it is unacceptable.

$$\text{score networking} = w_v \cdot V' + w_d \cdot D' \tag{6.8}$$

$$V' = 1 - \frac{\bar{v} - min}{max - min} \tag{6.9}$$

$$D' = \frac{\bar{d}}{max} \tag{6.10}$$

Equation 6.8 scores the agents regarding the networking perspective in terms of the number of created replicas and the fulfillment of the SLA expressed in terms of the processing latency. The goal of a scaler is to fulfill the SLA with the minimum number of replicas needed. However, minimizing the number of replicas has a trade-off for increasing the latency. Then, to compare how an agent performs over the remaining

agents, we need also to apply normalization. For the number of replicas, we use min-max normalization as expressed in Equation 6.9, where $\bar{v}$ is the average number of replicas created during the validation episode, min, and max are the minimum and the maximum values of all the created agents. In this case, we applied an inverse normalization since the min-max normalization will return values close to zero when $\bar{v}$ is close to the minimum.

Similarly, Equation 6.10 normalizes the latency. By using only the maximum latency value, Equation 6.10 is able to identify how far this maximum is from its mean value. The closer to one, the distribution is more centered around the mean, while the closer to zero, the distribution presents a right-hand side tail. This type of normalization is handy for detecting extreme peak values in the latency distribution [213], which should be avoided in mission-critical applications, for instance. Notice that, while the min, max in the previous equation should select the minimum (or maximum) between all the created agents, the max in Equation 6.10 corresponds to the maximum latency value of each agent during the validation episode. This value is associated with the latency distribution of the agent itself, and therefore it is irrelevant to compare it against other agents.

Equation 6.10 can also be seen differently. $\bar{d}/max$ can be closer to one when the mean latency is too high, e.g., above d_{tgt}. However, in how the DRL agent is built, the cases where the mean latency is high (closer to the maximum) are almost nonexistent. For those cases to happen, the agent must have created too few replicas, causing the latency to increase and risking falling in an episode termination case, hardly penalizing the agent. A well-trained agent will avoid this situation.

Being V' the normalized value of the replicas created by a given agent and D' the normalized latency of a given agent, Equation 6.8 tries to find a balance between the two main objectives of the scaler. w_v and w_d are weights used to tune which of the two criteria is more important to the Communication Service Provider (CSP). Moreover, the equations are chosen depending on the service type that the Network Function (NF) is providing. For instance, if the application allows the users to experiment some extreme latency issues, then Equation 6.10 could be replaced by another equation, e.g., $\bar{d}/d_{tgt}$ measures how far are the mean latency values from the target. After all the agents have been scored, model selection can start.

6.5.3 Selection

Model selection is a filtering process in which all the agents that were not able to perform in a plausible way during the previous two stages are ruled out. The first step in this process is to discard all the agents whose learning score is not similar to the measure of performance obtained during the training. As explained above, a well-trained agent is able to maintain the running of the validation episode; if this is not the case, the learning score will be higher than the measure of performance, indicating a possible outlier. Such agents should undergo model refinement, by, e.g., changing the seed. We filter first by the learning score since an ill-trained agent is most likely to have a poor performance in the scaling task.

Then, by using the networking score, we select the best scoring agent per RFn. Ultimately, the agent's true performance is given in terms of the networking KPIs and not the reward

the agent achieves during training. The chosen agent must satisfy two criteria: firstly, its learning score should rank among the highest within its respective RFn; secondly, if there exists statistical evidence demonstrating higher performance of one algorithm over another, i.e., DQN over PPO, the highest-performing agent should be trained using the algorithm proven to be more effective. If at least one of the criteria is met, the agent with the highest learning score should be deployed. The agents should undergo model refinement if none of the criteria is met. Notice that in the context of fully autonomous networks, the model refinement stage should be carried out by the NIO using simpler techniques, such as increasing the number of agent-environment interactions or the number of random seeds or using more complex techniques, such as hyperparameter tuning [214] or architectural network search [202] or even a complete redesign of the RFn.

6.6 Experimental Validation

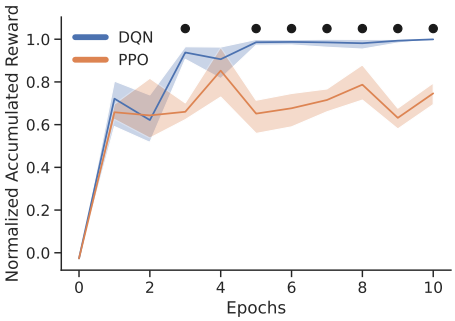
This section presents the experimental evaluation of the two DRL algorithms and the associated RFns. Through a systematic assessment, we aim to elucidate the impact of these algorithmic and RFn choices on the learning dynamics, convergence speed, and overall effectiveness of our proposed RL methodologies. The experiments are designed to provide insightful comparisons, shedding light on the strengths and limitations of each algorithm and offering a comprehensive understanding of their behavior. This empirical analysis validates our contributions and adds to the broader discourse on designing and optimizing RL algorithms for real-world applications.

6.6.1 Training Performance

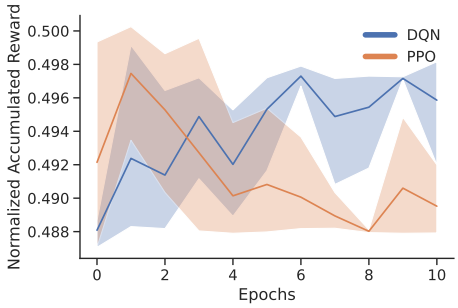
Using the procedure described in Section 6.5.1, we trained and validated the algorithms of Section 6.4.1 using the RFns described in Section 6.4.2. Figure 6.6 shows the learning curves, with 80% error percentile and black dots indicating statistical significance when applicable for all RFns. The agents were also evaluated before training to see how much improvement that learning brings. This is plotted as epoch 0.

As it can be observed from the figure, only RFn1 shows consistent statistical difference across the epochs, where the DQN performs better than the PPO. Unfortunately, we cannot draw any conclusions for the remaining RFns; this may be due to the requirement for additional data. More data can be gathered using two different methods. On the one hand, having more seeds will enable averaging more trials, yielding a more reliable algorithm performance metric [211]. On the other hand, for some cases, e.g., Figures 6.6c, 6.6d, 6.6e, it is noticeable that the algorithms, especially the PPO, have not yet converged to a stable value of the reward. More training epochs can help the algorithm reach convergence in these cases.

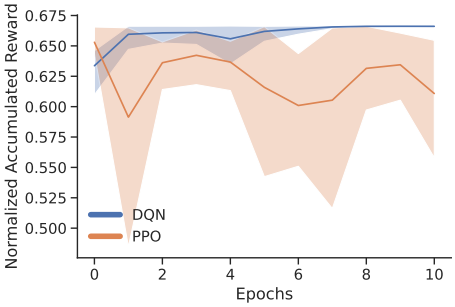
Notice, however, that when we talk about convergence, we do not talk about the algorithm converging to an optimal policy but to a stable reward value, which a sub-optimal policy can produce. Regarding the convergence to an optimal policy, it is known that DRL approaches are sample inefficient [215] and that, for instance, Q-learning algorithms are guaranteed to converge in the limit, when each state-action pair is visited infinitely



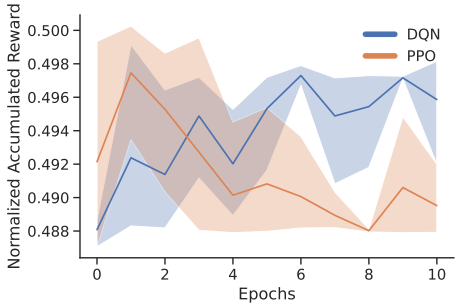
(a) Learning curve of RFn 1



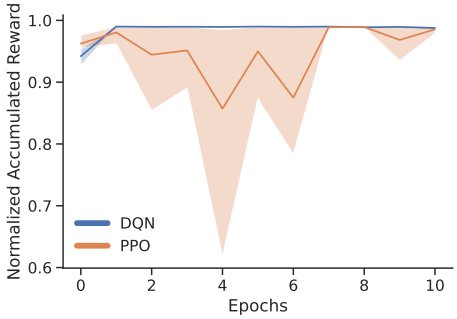
(b) Learning curve of RFn 2



(c) Learning curve of RFn 3_1



(d) Learning curve of RFn 3_2



(e) Learning curve of RFn 3_3

Figure 6.6: Learning curves, with error shades and black dots indicating significant statistical difference when present.

Table 6.4: Learning score of all the individual runs. In red are highlighted the cases where the agents terminated the validation episode before time.

	DQN					PPO				
	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**
RFn1	0.9971	0.9999	0.998	0.9999	0.9999	0.7462	0.7427	0.6765	0.6976	0.7529
RFn2	0.5662	0.5565	0.5521	0.5629	0.5598	0.3777	0.3784	0.3781	0.3784	0.3852
RFn3_1	0.6666	0.6666	0.6666	0.6666	0.6666	246.7	0.6581	1.522	0.6567	0.6588
RFn3_2	0.4975	22.28	0.4975	0.4975	0.4975	1.181	768.0	340.3	594.4	22.18
RFn3_3	43.92	0.9901	0.9901	0.9901	0.9901	0.9901	0.9901	0.984	0.9894	0.989

Table 6.5: Networking score of all the individual runs. The red dash identifies the agents disregarded in the previous stage, while in green are shown the best-performing agents per RFn.

	DQN					PPO				
	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**
RFn1	0.4142	0.4467	0.4166	0.4591	0.4567	0.4525	0.4519	0.4496	0.4444	0.4513
RFn2	0.5219	0.5225	0.5265	0.4472	0.4455	0.4867	0.4968	0.4883	0.5005	0.4931
RFn3_1	0.4499	0.5062	0.4999	0.4774	0.4736	-	0.3886	-	0.4193	0.3894
RFn3_2	0.8695	-	0.7842	0.8863	0.8873	-	-	-	-	-
RFn3_3	-	0.3026	0.2577	0.3462	0.3977	0.4416	0.4630	0.4190	0.4289	0.3844

often [76]. Some strategies have been incorporated in several state-of-the-art algorithms, such as using a ε-greedy policy where $\varepsilon > 0$ or by using a learning rate close to zero. However, in practice, DRL algorithms require millions of samples to achieve good and stable performance depending on the task [208, 176].

In our problem, we trained and evaluated each algorithm during N and V episodes (see Table 6.2), where each episode consists, in the best case, of 3600 samples or interactions. Then, we have $3600 \times V \times E = 432 \times 10^3$ samples or $3600 \times (N+V) \times E = 1296 \times 10^3$ samples, if we consider the training interactions. The number of samples we used is lower than those conventionally used in RL. Nevertheless, both DRL algorithms are able to stabilize when using RFn1 (Figure 6.6a), and to some extent, using RFn2 (Figure 6.6b), while the DQN is also able to stabilize using RFn3_1 (Figure 6.6c) and RFn3_3 (Figure 6.6e).

The value around each agent stabilizes differs from RFn to RFn. This is because of how often they achieve a reward in their trajectory. In RL, a trajectory refers to a sequence of states, actions, and rewards an agent experiences while interacting with an environment over a certain period. Trajectories are fundamental to RL as they are used to learn and update the agent's policy or value function, enabling it to make better decisions based on the cumulative experience gained during these trajectories. For instance, the agents trained using RFn1 will get a reward more often since they are rewarded if they keep the CPU usage within a boundary. This objective is easier to reach than to keep the latency within limits. Regarding RFn2, it is physically challenging for the system to remain in the state *below* while reducing the replicas. For this reason, the agents are not rewarded more often, leading to a lower accumulated reward. A similar reasoning can be followed for RFn3. In the following, we will look closely at each RFn, and via the results from the testing phase, we will analyze how each RFn influences the agent's behavior in the number of created replicas and the latency control.

6.6.2 Validation Performance

As mentioned in Section 6.5.2, we run each agent in a validation episode. The scores of each run are shown in Tables 6.4, and 6.5. Regarding the learning score, we observe in Table 6.4, the cases where this score is above 1, highlighted in red. Such cases, are immediately discarded and are not scored using Equation 6.8. According to the methodology introduced in Section 6.5 (see Figure 6.5), such agents can be replaced by others by changing the seed, for instance. Besides that fact, for most cases, the learning score corresponds to the stabilizing value of the normalized reward in their learning curves, as shown in Figure 6.6. In general, the learning score of the agents per RFn and algorithm is similar, which leads us to think that the algorithms can find an akin policy independently of their weight initialization.

Regarding the networking performance, as depicted in Table 6.5, agents that were eliminated in the previous stage are denoted with a red dash, while those with the highest scores per RFn are highlighted in green. Similar to Table 6.4, scoring is consistent among agents trained with the same RFn. However, our two-step approach, as outlined in our methodology, enables the identification of the most suitable model for deployment.

For instance, suppose we adhere to the conventional method of selecting the agent with the highest accumulated reward. In that scenario, agents 2, 4, and 5 of the DQN utilizing RFn1 could all be considered viable options. Nevertheless, upon closer examination of their scores, it becomes apparent that agent 2 slightly underperforms compared to the others. Furthermore, as illustrated in Table 6.5, PPO agents demonstrate better-than-anticipated performance, contrary to expectations based solely on their learning scores.

Combining both tables reveals that, in certain instances, assumptions based on learning scores align with evidence from networking scores. For example, agents trained with RFn1 consistently outperform PPO agents in both learning and networking. Similar trends are observed in RFn2. In the case of RFn3_1, although both algorithms exhibit similar learning performance, DQN demonstrates clear superiority over PPO in networking. Conversely, in RFn3_3, while all agents exhibit similar learning scores, networking scores indicate better performance by PPO agents for this specific function. In the following section, we look more closely at the behavior of the agents per RFn, including the best-performing ones.

6.6.3 Selection

In the final selection phase, we identify the top-performing agents as follows: (i) DQN *Run* 4 for RFn1, (ii) DQN *Run* 3 for RFn2, (iii) DQN *Run* 2 for RFn3_1, (iv) DQN *Run* 5 for RFn3_2, and (v) PPO *Run* 2 for RFn3_3. According to our methodology, we next verify if these agents meet the deployment criteria.

The learning score criterion ensures the selected agents are among the highest-ranked within their respective RFns. As observed from Tables 6.4 and 6.5, only DQN *Run* 4 with RFn1 and PPO *Run* 2 with RFn3_3 achieve high scores in both learning and networking. Further analysis of the learning curves in Figure 6.6 consistently shows that DQN agents trained with RFn1 outperform PPO agents, validating the selection of DQN *Run* 4 for deployment.

Table 6.6: Main Statistical values for the best-performing agents

	Replicas		Latency	
	RFn1 DQN - Run 4	**RFn3_3 PPO - Run 2**	**RFn1 DQN - Run 4**	**RFn3_3 PPO - Run 2**
mean	**5.0771**	**6.9999**	0.0089	0.0077
std	1.1276	0.0215	0.0004	0.0008
min	2.0000	2.0000	0.0058	0.0058
25%	5.0000	7.0000	0.0087	0.0074
50%	5.0000	7.0000	0.0089	0.0077
75%	6.0000	7.0000	0.0090	0.0081
max	**9.0000**	**7.0000**	**0.0612**	**0.0297**

Table 6.6 supports the observation that DQN generally outperforms PPO. The PPO agent tends to stabilize around 7 replicas, maintaining compliance with the SLA with minimal violations. In contrast, the DQN agent displays more variability in replica creation with slightly increased SLA violations than the PPO counterpart. While the PPO agent might be preferred in scenarios requiring consistent replica creation, the DQN agent is more effective in balancing deployment costs with user QoS. This makes DQN generally more suitable than PPO for this task.

6.6.4 Discussion

Future ZSM approaches will require an intelligence orchestrator [94] that (i) determines when a ML algorithm is ready for deployment, (ii) between two or more algorithms, compares which algorithm performs better for a given task, and specifically for RL algorithms, (iii) assess the performance of two or more RFns. For doing that, a common approach is to compare and benchmark DRL algorithms based on their reward in a well-defined and standard environment. However, this section will present strong evidence that ranking an algorithm based solely on the return might be insufficient. Therefore, the methodology presented in Section 6.5 tries to provide an automatic way of determining if two agents perform the same or differently or to select a better-performing algorithm, facilitating the work of the intelligence orchestrator. We analyze the agents' behavior per RFn to dive into the discussion.

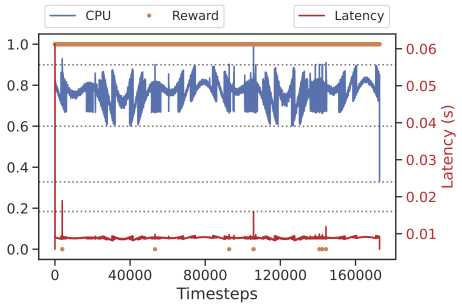
6.6.4.1 Analysis of the behavior of the agents using RFn1

This function rewards the agent if the monitored latency d_t or CPU usage c_t are between predetermined boundaries. These boundaries are set by defining an operating target and a tolerance range. For this RFn, we assume a target latency of 20ms and a CPU usage target of 75% with a tolerance of 20%, as indicated in Table 5.3. Notice that the target latency can be established by the SLA while the CPU target needs to be calculated according to operational needs. Figure 6.6a shows that only RFn1 provides statistical evidence to support the fact that DQN agents are behaving better than the PPO ones.

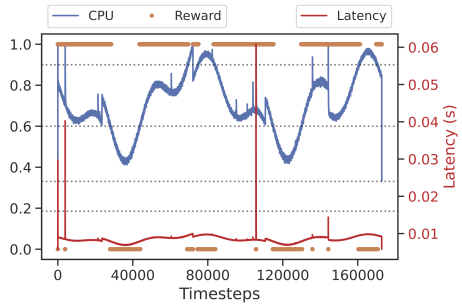
If only the reward is taken into account, Table 6.4 suggests that all DQN agents should

Table 6.7: Main Statistical values of Run 1 of the DQN and the PPO using RFn1.

	Replicas		Latency	
	DQN Run 1	PPO Run 1	DQN Run 1	PPO Run 1
mean	**5.1889**	**5.1234**	0.0088	0.0088
std	1.2358	0.5913	0.0011	0.0012
min	1.0000	2.0000	0.0058	0.0058
25%	4.0000	5.0000	0.0086	0.0085
50%	5.0000	5.0000	0.0087	0.0089
75%	6.0000	5.0000	0.0090	0.0091
max	**9.0000**	**6.0000**	**0.1440**	**0.0657**



(a) Run 4 of DQN



(b) Run 4 of PPO

Figure 6.7: Different agent behavior using RFn1. In blue is the CPU usage; in orange is the achieved reward per step; in red is the latency. The black-dotted line shows the respective thresholds

outperform the PPO ones. However, Table 6.7 proves this is not the case. The networking score shown in Table 6.5 is more accurate in estimating the performance of the agents, where PPO *Run 1* performs even better than DQN *Run 1* by creating fewer replicas in average and having similar latency control with fewer SLA violations.

Despite their lower learning scores, the PPO agents demonstrate performance comparable to DQN agents with higher learning scores. The learning score, based solely on the obtained return, indicates that DQN algorithms are better at optimizing the objective for this RFn, i.e., keep the latency and CPU usage within bounds. In contrast, the networking score shows how the agents will perform when deployed. For instance, Figure 6.7 showcases the top-performing DQN agent (*Run 4*) and PPO agent (*Run 4*), which has one of the lowest learning scores for RFn1. In the graph, the orange line represents the reward achieved by the agent at each step, while the blue and red lines denote the average CPU usage and peak latency of created replicas, respectively. Horizontal black dotted lines indicate CPU and latency bounds. Notably, the DQN agent consistently receives rewards at each step, while the PPO agent struggles, particularly during low or high workload periods. As shown in the Figure, the success of DQN agents lies in their superior control of CPU usage.

6.6.4.2 Analysis of the behavior of the agents using RFn2

We assume the same latency target for this RFn as for RFn1, 20ms. Consequently, the *Above* state is defined as d_t above the latency target, while the *Below* state is defined as d_t below that target. This RFn showcases interesting characteristics of DRL algorithms. First, this RFn is evidence of what in the RL community is known as reward hacking [216]. Reward hacking is the term employed when the agent finds a way to maximize the expected return that is not aligned with the desired behavior. In particular, some of the trajectories of the agents using RFn2 reflect that once an agent is below the latency, it reduces the number of replicas, so it slightly goes above the latency. Being there, the agent increases the number of replicas to receive the reward. This behavior is repeated several times during a trajectory, allowing the agent to receive a reward every 2 or 3 steps with the side effect of occasionally trespassing the latency target.

Examining the agent behavior more closely, the PPO agents adjust replica numbers to transition between states, initially increasing replicas to reach the *Below* state, then reducing them to obtain rewards, before transitioning back to the *Above* state. This cyclic process leads to a gradual increase in achieved rewards over time. Conversely, DQN agents employ a similar strategy but exploit the absence of penalties for creating surplus replicas, remaining in the *Below* state with excess replicas to increase the likelihood of reward acquisition. However, both agent types struggle to meet SLA requirements, with approximately more than 20% of the time spent in violation, indicating the need for a more refined RFn design.

Second, since the agents cannot be rewarded in every step, only a certain combination of actions will produce a reward, which can be considered a sparse RFn. However, finding a suitable policy within a reasonable time is more difficult in a sparse reward setting than in a dense one [217]. The RFn's sparsity is also why the learning score of the DQN and PPO agents is lower than their counterparts using RFn1. Finally, Figure 6.6b, and Tables 6.4 and 6.5, suggest that DQN agents perform subtly better than the PPO ones. Still, this is not the case for every epoch, especially towards the final ones where the DQN agents seem to suffer catastrophic forgetting.

Notice that, from the perspective of the methodology, all PPO agents should be discarded since their learning score (Table 6.4) is higher than the measure of performance (Figure 6.6b). Continuing with the methodology, the best DQN agent (*Run 3*) does not have the best-performing learning score (cf. Table 6.4). Therefore, there is no guarantee that any of the DQN agents will perform consistently when deployed in production networks.

6.6.4.3 Analysis of the behavior of the agents using RFn3

This RFn is a multi-objective function where depending on the optimization profiles (cf. Table 6.1), an agent tries to minimize the associated cost of creating replicas or surpassing a threshold in latency. In the first optimization profile, the agents will try to balance the two objectives; in the second, the agents will optimize replicas creation; in the last optimization profile, the agents will pay more attention to latency control. Therefore, three different behaviors are expected. Having multiple optimization profiles, a network

solution designer can easily switch the agents' behavior so they optimize one objective, i.e., control the latency, or the other, i.e., limit the creation of replicas.

This RFn emphasizes the need for an alternative scoring metric for DRL-based scalers. Notice how, according to Table 6.4, all DQN agents score the same for the different optimization profiles defined by RFn3; the networking score helps in determining which agent is actually performing the best, same as in RFn1. Moreover, focused only in RFn3_3, the networking score helps in determining the best-performing agent even across algorithms, revealing that the PPO agent of *Run 2* performs the best.

Following the methodology reveals that, in RFn3_1, the PPO agents are discarded since their learning score is not similar to the measure of performance (cf. Figure 6.6c). Moreover, the agents trained with RFn3_2 are discarded since we cannot guarantee that their agents are performing consistently, similar to RFn2.

6.6.4.4 Final Remarks

From analyzing in more detail the results obtained, we identified that achieving a stable reward during training is recommended for RL scaling algorithms but not mandatory for an agent to meet key scaling objectives, such as maintaining SLA compliance with minimal replicas. To address this challenge, we proposed a methodology that includes a scoring system based on the reward to quickly identify efficient agents, which generally corresponds to the stabilized reward values seen in training.

However, the scoring system has limitations, particularly in distinguishing between agents that appear similar in learning but differ in deployment suitability. To address this, our methodology also evaluates the performance of scaling tasks, focusing on the number of replicas created and SLA compliance. Despite the lack of a formal evaluation methodology for auto-scaling techniques, our approach aims to fill this gap, particularly in model selection for autonomous network operations.

Finally, we emphasize the critical role of the RFn definition in DRL-based scaling algorithms. While the default RL objective is to maximize expected reward, timely decisions in scaling, such as when to create or delete replicas, can significantly impact long-term performance. This explains why agents that excel in managing replicas and latency may receive lower rewards than those with less effective performance. Using RFns with non-cumulative objectives [218] may yield better results, as different optimization profiles can lead to varying behaviors that better balance the trade-offs between objectives.

6.7 Conclusion

ML algorithms are proposed to transform networks by optimizing areas such as resource management, security, and user experience. As these algorithms become integral to network operations, the need for efficient performance evaluation using relevant KPIs grows. This chapter introduces a methodology for designing, training, and evaluating DRL algorithms, focusing on scaling resources in service-based networks. The scaling problem is modeled as an MDP, where the goal is to determine the number of replicas

in a changing environment. The study tested different RFns to guide agent learning. Following common practices in the RL community, we experimentally evaluated the behavior of the DRL agents. We determined, when possible, the best-performing agent in terms of the replica creation and latency control. It was observed that only RFn1 showed a consistent statistical difference across the training epochs, with the DQN algorithm outperforming PPO.

However, it is important to note that no further conclusions can be drawn for other RFns. We hypothesize several reasons for this. First, no hyperparameter optimization was performed on the algorithm. The hyperparameters used were taken directly from the original versions, where they were fine-tuned for different tasks, which may result in suboptimal performance. On the other hand, this also highlights the generalization capabilities of RL, as the algorithms still perform well without being specifically fine-tuned for this task. Another possible explanation is that the agents struggled to receive rewards, particularly in RFn2. A potential solution to this issue could be introducing intermediate rewards to guide the agent's learning process better.

The chapter also highlights the challenge of fairly comparing scaling algorithms, proposing a methodology that integrates learning and performance metrics like replica creation and SLA compliance. This methodology helps in performing model selection, a crucial step towards completely autonomous network operation, where ideally, the NIO should select the best-performing algorithm for this task. In that sense, our methodology extends current MLOps frameworks by considering multiple models designed with different RFns performing the same task.

Based on the literature analysis, we observe that the applicability of RL techniques in networking leaves ample room for further innovation. Thus, as a contribution to closing that gap, we provided an abstraction layer to integrate two platforms, *OpenAI Gym*, the most used library to train and evaluate RL algorithms, and *DynamicSim*, a discrete-event simulator that enables the creation of edge-cloud network scenarios. With this abstraction layer, we created the playground on which scaling algorithms based on RL can be freely trained, tested, and compared.

Future work involves extending *DynamicSim* to handle more complex scenarios, such as service function chaining and energy consumption optimization, while also addressing the challenges associated with real-world implementations. Additionally, the methodology should be refined in two key areas. First, RL algorithms should be fine-tuned through hyperparameter optimization. Second, the methodology should be expanded to include the comparison and selection of scaling algorithms, including those that are not RL-based. Finally, once these improvements are implemented, the validity of the approach should be re-evaluated.

Chapter 7

Conclusions

This chapter summarizes the key contributions of this dissertation. Based on the problem statements outlined in Section 1.2, we formulated the Research Questions (RQs) posed in Section 1.3, having the hypotheses presented in Section 1.4 as a starting point. The contributions presented in this chapter stem from investigating the mentioned RQs and will provide a means to verify our initial hypotheses. We conclude this chapter by discussing the open challenges that remain following the various contributions of this research and exploring future prospects in the field.

7.1 Main research contributions

The main goal of this dissertation is to provide a set of tools, methodologies, and strategies to pave the way towards the vision of autonomous networks. The previous chapters showed multiple contributions toward this end. First, we defined the Network Intelligence Stratum (NI Stratum), an end-to-end orchestrator for Network Intelligence (NI). Thanks to its design, two NI solutions were developed in the subsequent chapters following well-known closed-loop approaches. Finally, Chapter 6, presented a methodology to design, train, validate, and select Reinforcement Learning (RL) algorithms, improving current Machine Learning Operations (MLOps) frameworks to consider not only the learning-related metrics but also the network-related metrics when developing NI. This section reviews how the contributions of previous chapters addressed the problem statements of Section 1.2, validating the hypotheses presented in Section 1.4. Specifically, four contributions were identified in this dissertation:

[C_1] **A complete architectural design of the NI Stratum, an end-to-end orchestrator for NI.**

In Chapter 3, we presented the complete architectural design and supporting procedures to realize a NI Stratum for next-generation networks. More precisely, the main findings in this contribution are as follows:

- Built upon well-known frameworks for autonomic computing, the NI Stratum allows NI solutions to be decomposed throughout the entire network infrastructure. Using the extended Network Monitor-Analyze-Plan-Execute

over a shared Knowledge (N-MAPE-K) (cf. Chapter 3.3.1), we mapped the NI solutions presented in Chapters 4 and 5, verifying that such frameworks do effectively decompose NI into atomic components, allowing to properly define NI. Thanks to this decomposition, we were able to identify that Supervised Learning (SL) approaches are typically placed in the *Analyze* blocked, while RL are more suitable for the *Plan* block of the N-MAPE-K.

- The set of internal interfaces defined within the NI Stratum facilitate the interaction among its functionalities. Such interfaces represent an adaptation or an extension of current standardized interfaces proposed by major Standard-Defining Organizations (SDOs), e.g., European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) Management and Orchestration (MANO) [64]. Moreover, we analyzed how the external interfaces may be realized when interacting with the Radio Access Network (RAN) and Core network segments. This set of interfaces (cf. Chapter 3.4) was inspired in NFV MANO [64], complementing the additional interfaces needed to provide the appropriated Machine Learning (ML) workflows, and external interfaces that connect the NI Stratum to the different per-domain network controllers, providing a means to increase integration and interoperability with current systems.
- We provided a detailed description of several inter- and intra-Network Intelligence Orchestrator (NIO) procedures that are required to realize the NI Stratum. We not only defined them as functionalities but also provided the required workflows and sequence diagrams that allow us to perform these procedures. Through such procedures, NIO's internal functionalities can effectively cooperate to solve challenges such as conflict resolution, knowledge sharing among Network Intelligent Functions (NIFs) and NIF coordination. Additionally, we explored how the different components work together to create a cohesive system that can effectively orchestrate intelligence across multiple domains.

This contribution and the findings mentioned above provide answer to the Research Question RQ_1 and RQ_2 which were aimed to solve the problem statements P_1 and P_3, respectively. Besides, this contribution confirms hypothesis H_1: Autonomic computing frameworks based on closed-loop control **can effectively decompose NI solutions**, thereby enhancing their coordination and interaction among multiple NIFs. This approach **improves End-to-End (E2E) convergence and global optimization in network management, addressing the current limitations in MANO frameworks** and facilitating the development of autonomous systems.

[C_2] **A NI solution for interference management in wireless networks.**

In Chapter 4, this dissertation introduced two approaches for interference management in wireless networks. The first approach exploited additional information embedded in the network's topology to develop a Graph Neural Network (GNN) model that predicts the performance of a dense wireless deployment, providing an accurate, real-time prediction of the impact of future changes, e.g., selecting the channels to be bonded in a deployment. Secondly, a spectrum management framework enhanced by Artificial Intelligence (AI) for improving coexistence in the unlicensed bands between two widely used technologies, namely, IEEE 802.11 (Wi-Fi) and Long-Term Evolution (LTE). Both data-driven approaches, allow real-

time adaptation to fluctuating conditions, improving spectrum utilization. More precisely, the main findings in this contributions are as follows:

- The intra-technology interference management approach was based on a GNN model that predicts the achieved throughput in highly dense IEEE 802.11 Wireless Local Area Networks (WLANs) using Channel Bonding (CB). The approach proved to be more accurate than recent state-of-the-art Deep Learning (DL) and ML approaches, since the GNN model exploited the additional information embedded in the network's topology. Moreover, we discussed how different features impact the prediction accuracy. We showed how the model can be used in a interference management applications in a Software Defined Networking (SDN) framework, that, based on the available features, a given model can be optimally selected. This interference management application leverages the ability of the data-driven approach to obtain predictions in almost real-time to measure the impact of possible configuration changes, i.e., change in the bonded channels.
- The inter-technology interference management approach proposed a distributed spectrum management framework that (i) implemented a state-of-the-art ML technique that identified several wireless technologies and provided statistics about spectrum usage, (ii) included a rule-based Wi-Fi decision algorithm that adapted to changing conditions and (iii) did not require fundamental changes in current wireless environments and therefore can be deployed in both legacy and SDN-like setups. The approach proved to be effective in real-life deployments by reducing the throughput loss caused by a colliding LTE transmission.
- Based on the N-MAPE-K, we described both use cases, showing how NIFs for interference management in wireless networks can be generically defined, allowing re-utilization and evolution.

This contribution and the findings mentioned above provide answer to the Research Question RQ_3 and RQ_5 which were aimed to solve the problem statements P_2, P_5, P_6 and P_7. Besides, this contribution confirms hypothesis H_2: Developing context-aware, data-driven strategies **allow networks to enhance real-time adaptation to fluctuating conditions** and improve network and computation resource utilization. As such, these models would boost overall network performance, reducing latency and **minimizing interference without compromising users Quality of Service (QoS) and Quality of Experience (QoE)**.

[C_3] **A NI for scaling in service-based network architectures.**

In Chapter 5, this dissertation suggested a scaling method for service-based network architectures based on Deep Reinforcement Learning (DRL). Adapting prior knowledge on well-known RL problems, the proposed scaler was able to reduce the Service Level Agreement (SLA) violations, while maintaining control over the Virtual Network Functions (VNFs)' replicas. Moreover, the chapter showed how prominent scaling methods can easily be mapped to the proposed N-MAPE-K. More precisely, the main findings in this contributions are as follows:

- We designed and evaluated three auto-scaling methods: an RL-, a Proportional–Integral–Derivative (PID)- and a Threshold (THD)-based algorithm that determined the number of VNF instances to guarantee an SLA. Unlike

predictive scalers, the proposed approaches did not require any information about the workload and yet could maintain the number of VNF instances at a level that avoids incurring in over- or under-dimensioning.

- We proposed a methodology to define the RL-based auto-scaler that first maps the auto-scaling problem to a Markov Decision Process (MDP) of a well-known RL problem and then adapt it accordingly instead of defining it from scratch. This approach is contrary to most of the work in the literature where the MDPs are traditionally directly determined by the networking problem with all the difficulties of designing a properly working MDP.
- Based on the N-MAPE-K, we described several scaling methods (e.g., ML-, rule-, and control theory-based) showing how NIFs for NFV scaling can be generically defined, allowing re-utilization and evolution.
- Using a cloud-based scenario, we quantitatively compared the scaling methods showing that, despite several approaches fitting the N-MAPE-K, only data-driven algorithms produce the desired adaptation and automation in network operations.

This contribution and the findings mentioned above provide answer to the Research Question RQ_3 which is aimed to solve the problem statement P_2. Besides, this contribution confirms hypothesis H_2: Developing context-aware, **data-driven strategies allow networks to enhance real-time adaptation to fluctuating conditions** and improve network and computation resource utilization. As such, these models would boost overall network performance, **reducing latency** and minimizing interference **without compromising users QoS and QoE**.

[C_4] **A methodology to design, train, validate, and select RL algorithms, enhancing MLOps frameworks.**

In Chapter 6, this dissertation presented a methodology to design, train, validate, and select DRL algorithms with different Reward Functions (RFns) and their use in networking. We believe this methodology will improve current MLOps frameworks to consider not only the learning-related metrics but also the network-related metrics when developing NI. More precisely, the main findings in this contribution are as follows:

- As a use case, we focused on the scaling of computing resources in service-based network architectures such as Open Radio Access Network (O-RAN) or NFV. This use case requires intelligent scalers since current approaches for scaling are based on Central Processing Unit (CPU) usage. However, under this approach, the correlation between SLA, QoS parameters, scaling decision triggers, and learning metrics are unknown or difficult to model. For instance, Altaf et al. and Gotin et al. [219, 220] show that CPU utilization might not be the best metric for non-CPU intensive applications. We modeled the scaling problem as an MDP and proposed an RL algorithm that determines the number of replicas under a given constraint and constantly changing environment.
- We designed three RFns that are suitable to the problem. Different RFns can guide agents to explore distinct strategies to achieve their objectives. This exploration helps uncover a broader spectrum of potential solutions and can lead to more robust and adaptive learning. In particular, sparse RFns are

challenging for agents to learn effectively since the agents only receive a reward after executing a sequence of actions. In contrast, the other two RFns provide more frequent and informative feedback, facilitating faster learning and convergence. In addition, we explore how the agent balances competing goals. This is particularly important in multi-objective RL, where optimizing one objective may come at the cost of deteriorating another.

- We enhanced the MLOps methodology by incorporating practices for algorithm selection and comparison. In contrast to prior methodologies such as MLOps [190], Reinforcement Learning Operations (RLOps) [46], Q-Model [221], and O-RAN workflows [193], our approach considered multiple RL models introduced above, tailored to solve a specific problem. Furthermore, these diverse RL models were configured with different RFns, introducing an additional challenge as we aimed to optimize the lifecycle management for not just one but multiple algorithms. Moreover, we validated this methodology via extensive experimentation and performance evaluation.
- We proposed a benchmark on which scaling agents can be tested, and their algorithms can be compared. The benchmark is built on top of well-known frameworks in RL research, such as Stable Baselines [184] and OpenAI Gym [51]. We open-sourced this benchmark with the agents as a baseline for further evaluating, comparing, and advancing RL algorithms and models. Furthermore, it can complement well-established gym environments for networking, such as [204].

This contribution and the findings mentioned above answer the Research Question RQ_4, which aims to solve the problem statements P_4 and P_8. Besides, this contribution confirms hypothesis H_3: Enhancing current network management frameworks with **integrated platforms and protocols for lifecycle management of NI solutions**—including support for various ML types (e.g., SL, RL) and modes (e.g., training, testing)—will improve model effectiveness and network performance. Additionally, **establishing benchmark datasets and standardized evaluation metrics for ML models in networking** will facilitate rigorous research, leading to more robust and effective models that better exploit network specificities.

7.2 Open challenges and future perspectives

The main contributions of this dissertation were discussed in the previous section, along with how they addressed the RQs and provided evidence for the hypotheses that were put forth. Notwithstanding the contributions made by this dissertation in this field, challenges and opportunities for further exploration remain. Thus, in this section, we will discuss the limitations of the proposed approaches, potential roadblocks to real-world deployment, and areas where future research can significantly contribute to the evolution of intelligent next-generation networks.

7.2.1 Large-scale Simulations and real-life Deployments

The four technical chapters of this dissertation introduce several ideas for defining, creating, deploying, and orchestrating NI to address various network management challenges. Despite being developed with well-known, open-source frameworks, the solutions were only tested and evaluated in small-scale deployments or simulations. Consequently, a critical next step is rigorously testing and evaluating these solutions in real-life deployments or large-scale simulations. This challenge is urgent, as it will determine the practicality and scalability of the proposed solutions in real-world scenarios.

For instance, Chapter 3 presented the design of the NI Stratum. However, its implementation is left as future work. Deploying and managing a NI Stratum is a complex process that involves the integration and coordination of multiple frameworks, not only including network MANO and MLOps frameworks, as shown in this paper, but also data management frameworks [222]. Despite being data an easy-to-obtain resource in networks, the variety and the velocity of the network-related data are higher than in typical ML applications. Currently, there are many frameworks for managing data, some of which are case-specific. Thus, developers and data engineers utilize a diverse array of data frameworks. While offering flexibility, this diversity reflects the challenges in creating a cohesive data pipeline architecture that efficiently bridges different computational environments and aligns with the diverse requirements of telecommunication networks and their users.

In Chapter 4, this dissertation introduced two approaches for interference management in wireless networks. Developing production-ready NI models is cumbersome, comprising the optimization of complex networking tasks and hyperparameter tuning, all while adhering to network constraints such as throughput and latency. This complexity hinders the efficient and error-free deployment and management of NI and calls for proficiency not only in ML but also in network design and programmable hardware. Regarding the intra-technology interference management approach based on GNNs, the fact that the model is assessed solely with synthetic data is a limiting factor. To complete the study, it is worth including large-scale simulations with legacy WLANs that can only transmit over one unbonded channel, more dynamic scenarios that reveal the spatiotemporal dependence of the link quality with the achieved throughput, the impact of non-Wi-Fi interference on the models, and future Orthogonal Frequency Division Multiplexing (OFDM) operation.

On the other hand, the inter-technology interference management approach focused only on Wi-Fi, but the presented results can be extended to other wireless technologies, including different variations of Wi-Fi, such as IEEE 802.11b/g/ac, by re-training the Technology Recognition (TR) module to recognize new technologies. Thanks to the separation of the Wi-Fi controller with the Access Point (AP), this contribution can be easily incorporated in much more powerful wireless management frameworks so that handovers can not only be performed at frequency level (using Channel Switch Announcement (CSA)) but also in space (moving clients among physical APs). Additionally, internal statistics, typically collected by such frameworks, can improve the quality of the decisions. Finally, our interest in this chapter was mainly towards LTE's interference and how Wi-Fi can adapt, but given that new wireless technologies are emerging, we can surely exploit TR's capabilities to create a technology-agnostic spectrum management framework.

Finally, Chapter 5 presented two scaling methods for service-based network architectures based on DRL. Despite the promising results obtained by the DRL-based scaler, one of the open challenges considers using more complex scaling scenarios where the same agent performs vertical and horizontal scaling. From the RL point of view, this scenario extends the scaler's action set, possibly resulting in more convergence time and increased complexity. Similarly, Chapter 6 presented a methodology to design, train, validate, and select DRL algorithms with different RFns and their use in networking. However, based on the literature analysis, we observe that the applicability of RL techniques in networking is not mature enough to be deployed in production networks. An interesting idea to speed up the maturity of such kind of algorithms is to combine the abstraction layer of Santos et al. [183] and the evaluation methodology of Zalokostas et al. [223] as a way to implement RL algorithms for scaling in real-life deployments. However, to avoid extra costs, the algorithm could first be trained in our benchmark to improve the quality of the algorithm's decisions.

7.2.2 Advanced Machine Learning based Techniques for Network Management

Chapters 4 and 5 introduced different solutions utilizing state-of-the-art ML-based techniques such as GNNs, Convolutional Neural Networks (CNNs) trained in a semi-supervised way, and DRL algorithms. Nonetheless, as network complexity escalates, so must the solutions' complexity. An example of this is the NI Stratum, which involves detecting and resolving conflicts, especially when multiple NI operate within different network domains. Effective policy interpretation, configuration, and enforcement are necessary to manage these conflicts. This requires a robust mechanism for interpreting NI decisions and outcomes.

It is undoubtedly that a first approximation to autonomous networks is autonomic networking [224], where human guidance and intervention for networks are still required. Operators and network engineers should understand the decisions and outcomes of the NI models. In that sense, eXplainable Artificial Intelligence (XAI) [225] is the broad term that comprises a set of methods that allow human users to understand the decision-making process of ML systems, providing clear insights into how the ML model arrives at a particular decision. The goal of XAI is to make AI systems transparent, interpretable, and trustworthy. This entails explaining precisely how an AI model arrives at a given conclusion, the factors that influenced the result, and the degree of confidence the model has in its forecasts.

Leveraging XAI techniques can significantly enhance trust in black-box NI models by offering insights into their inner workings and demystifying their outcomes. As networks become more autonomous, explaining decisions made by NI is crucial, especially in scenarios with critical consequences. XAI ensures that AI-driven decisions within NI are fair, unbiased, and compliant with regulatory standards. By illuminating the complex decision-making processes of DRL algorithms, XAI makes their policies more transparent and interpretable for humans, thereby enhancing trust and reliability in AI systems in network management. Additionally, understanding DRL policies can lead to better RFn design, as identifying pitfalls allows for targeted improvements [226].

In autonomous networks, decisions must be made quickly and efficiently, minimizing

human intervention. Causal ML [227] focuses on understanding the cause-and-effect relationships in data, going beyond mere correlations. This is crucial for identifying the root causes of network issues such as congestion, failures, or security breaches. XAI complements this by providing interpretable models and explanations for the decisions based on these causal relationships. This combination is important in autonomous networks where self-healing capabilities are required.

A flexible architecture, like the one proposed in Chapter 3, is desired to support both ML operation modes (i.e., training and testing) in production networks. When the NI solution is under training, its outcomes and decisions cannot be directly applied to production networks since it would compromise their stability and reliability. But after achieving its desired performance during training, the NI can be deployed in the production network. However, since the possibility of experiencing a shift in the data distribution exists, the outcomes of an already trained NI can become obsolete. Therefore, new kinds of algorithms that are transferable to other (related) domains must be developed, for example, by training in simulators and transferring that knowledge to real-world networks using techniques such as transfer learning [228].

Moreover, future networks' dynamic and constantly evolving nature introduces variability that can lead to data drift [103], impacting the ML model performance and potentially degrading the managed network's performance. Two approaches are often recommended to address data drift: first, enhancing ML models' generalization through zero-shot learning techniques [229] or hierarchical reinforcement learning [230], allowing models to handle unseen classes during testing. Second, incorporating data drift detectors [231] into the lifecycle management of the ML models. These strategies support the goals of zero-touch management, enabling networks to become fully autonomous.

Finally, grasping the hype on generative AI, Large Language Models (LLMs) are proposed [232] for the telco industry as an enabler of intent-based networking. In this paradigm, high-level descriptions of desired network outcomes, or intents, are provided to the LLM. These models can be used to create a pipeline that breaks down intents into a sequence of policies, enabling automated execution through a closed control loop [233]. Furthermore, LLMs can automatically generate, verify, and implement network configurations based on high-level intents expressed in natural language. These models could be implemented as agents in the NI Stratum and invoked through the NIO procedures.

7.2.3 Network Digital Twins

As already stated, when the NI solution is under training, its outcomes and decisions cannot be directly applied to production networks since it would compromise their stability and reliability. Thus, one solution contemplated by the flexible architecture, like the one proposed in Chapter 3, is to inject the decisions of the NI under training in the network digital replica. Developing accurate network digital twins is also paramount for future research in NI. These abstractions help to bridge the gap between ML model training and deployment in autonomous networks.

During training, the NI model parameters are fine-tuned to suit the particular challenges the model is intended to solve. To mitigate the risk of introducing instability to the network due to premature decisions, the outputs from the training phase should be

first tested within a digital twin of the network. Such a digital twin acts as a safe environment [234], allowing potential errors or inefficiencies from an incomplete training process to be identified and rectified without any adverse impact on the actual network operations, as suggested by Almasan et al. [25]. The challenge lies in creating a digital twin that precisely mirrors the complexities and dynamics of the target network, which remains an open issue in the field.

Developing accurate network digital twins involves leveraging ML techniques such as GNNs for modeling complex relationships within networks. Furthermore, accurately predicting and modeling network behavior depends on maintaining real-time data integration and continuous learning through telemetry and self-learning mechanisms [235]. Developing a thorough and trustworthy network model requires putting into practice a methodology that addresses every stage of the digital twin's lifespan, from data collection to bidirectional connections between digital and physical components [236]. Moreover, the accuracy and efficiency of digital twin building depend on tackling the temporal misalignment of the multidimensional data and eliminating redundant data acquisition associated with the physical network [237]. All these methods can be combined to create network digital twins that precisely estimate performance metrics, optimize network management, and support decision-making processes effectively.

7.2.4 Distributed Intelligence

The inherently distributed nature of telecommunication networks, now a heterogeneous amalgam of diverse devices, technologies, and stakeholders, makes it impossible to have a one-size-fits-all solution. This complexity necessitates the collaboration of multiple targeted solutions to approach an optimal outcome. A common challenge across many studies is the heterogeneity of devices and communications, requiring sophisticated, distributed systems to work together effectively.

Federated Learning [238, 239] is an approach that shows promise in meeting the demands of varied network settings. It allows collective model training while protecting data privacy at each local node. To improve system efficiency and stability, however, important issues must be resolved, such as efficiently exchanging knowledge between various models, resolving possible conflicts that could cause the system to become unstable, and synchronizing shared knowledge to reduce communication overhead.

Bibliography

[1] IBM, "An architectural blueprint for autonomic computing," white paper, IBM, Jun, 2006. xix, 2, 3, 6, 20, 45

[2] S. Sangwoo, "The Success of AI Depends on the Speed of Iteration: An MLOps Strategy for AI Models in Manufacturing." Accessed: 2024-05-30. xix, 4

[3] M. K. Bahare, A. Gavras, M. Gramaglia, J. Cosmas, X. Li, O. Bulakci, A. Rahman, A. Kostopoulos, A. Mesodiakaki, D. Tsolkas, M. Ericson, M. Boldi, M. Uusitalo, M. Ghoraishi, and P. Rugeland, "The 6G Architecture Landscape - European perspective," feb 2023. xx, 48

[4] M. Camelo, M. Gramaglia, P. Soto, L. Fuentes, J. Ballesteros, A. Bazco-Nogueras, G. Garcia-Aviles, S. Latré, A. Garcia-Saavedra, and M. Fiore, "Daemon: A network intelligence plane for 6g networks," in *IEEE Globecom Workshops (GC Wkshps)*, pp. 1341–1346, IEEE, 2022. xxi, 40, 43, 48, 132, 134, 135, 144

[5] 3GPP, "5G System Overview." `https://www.3gpp.org/technologies/5g-system-overview`. Accessed: 2024-05-28. 1

[6] W. Saad, M. Bennis, and M. Chen, "A vision of 6g wireless systems: Applications, trends, technologies, and open research problems," *IEEE Network*, vol. 34, no. 3, pp. 134–142, 2020. 1, 38

[7] S. S. Mwanje and C. Mannweiler, "Towards cognitive autonomous networks in 5g," in *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*, pp. 1–8, IEEE, 2018. 2

[8] H. Jiang, T. Wang, and S. Wang, "Multi-scale hierarchical resource management for wireless network virtualization," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 919–928, 2018. 2

[9] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643–655, 2019. 2

[10] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao, "Dynamic service function chain embedding for nfv-enabled iot: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 507–519, 2019. 2

[11] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018. 2, 19

[12] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, 2018. 2, 20

[13] E. Coronado, R. Behravesh, T. Subramanya, A. Fernàndez-Fernàndez, M. S. Siddiqui, X. Costa-Pérez, and R. Riggio, "Zero touch management: A survey of network automation solutions for 5g and 6g networks," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2535–2578, 2022. 2, 20, 132

[14] ITU-T, "ITU-T Y.3172-series - Architectural framework for machine learning in future networks including IMT-2020," recommendation, ITU-T, 2019. 2, 38, 39, 43

[15] TM FORUM, "Autonomous Networks: Empowering digital transformation for smart societies and industries," white paper, TM FORUM, 2020-10. 2, 5, 43

[16] ITU-T, "Autonomous networks – Architecture framework," recommendation, ITU, 2023. 2, 40, 43

[17] ETSI, "Autonomous Networks, supporting tomorrow's ICT business," white paper, ETSI, 2020-10. 2, 5

[18] ETSI, "Improved operator experience through Experiential Networked Intelligence (ENI)," White Paper No. 22, ETSI, Oct, 2017. 2, 6

[19] ETSI, "Zero-touch network and Service Management (ZSM); Reference Architecture," group specification, ETSI, 2019-08. 2, 6, 38, 104, 108, 132

[20] J. R. Boyd, "The Essence of Winning and Losing." Accessed: 2024-05-29. 3, 6

[21] J. Strassner, N. Agoulmine, and E. Lehtihet, "Focale: A novel autonomic networking architecture," 2006. 3, 6

[22] M. El Rajab, L. Yang, and A. Shami, "Zero-touch networks: Towards next-generation network automation," *Computer Networks*, vol. 243, p. 110294, 2024. 3

[23] T. Zhang, M. Hemmatpour, S. Mishra, L. Linguaglossa, D. Zhang, C. S. Chen, M. Mellia, and A. Aghasaryan, "Operationalizing ai in future networks: A bird's eye view from the system perspective," *arXiv preprint arXiv:2303.04073*, 2023. 3, 132, 134

[24] F. Wilhelmi, M. Carrascosa, C. Cano, A. Jonsson, V. Ram, and B. Bellalta, "Usage of Network Simulators in Machine-Learning-Assisted 5G/6G Networks," *IEEE Wireless Communications*, vol. 28, no. 1, pp. 160–166, 2021. 4, 8, 83, 84, 85, 135

[25] P. Almasan, M. Ferriol-Galmes, J. Paillisse, J. Suarez-Varela, D. Perino, D. Lopez, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong, V. Ram, S. Xiao, X. Shi, X. Cheng, A. Cabellos-Aparicio, and P. Barlet-Ros, "Network digital twin: Context, enabling technologies and opportunities," *IEEE Communications Magazine*, pp. 1–13, 2022. 4, 6, 45, 115, 135, 144, 167

[26] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015. 4

[27] S. Hämäläinen, H. Sanneck, and C. Sartori, *LTE self-organising networks (SON): network management automation for operational efficiency*. John Wiley & Sons, 2012. 6

[28] A. Banerjee, S. S. Mwanje, and G. Carle, "Toward control and coordination in cognitive autonomous networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 49–60, 2021. 6

[29] A. Collet, A. Banchs, and M. Fiore, "Lossleap: Learning to predict for intent-based networking," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pp. 2138–2147, IEEE, 2022. 6

[30] M. Camelo, P. Soto, and S. Latré, "A General Approach for Traffic Classification in Wireless Networks Using Deep Learning," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 5044–5063, 2022. 6

[31] Z. Chen, K. He, J. Li, and Y. Geng, "Seq2Img: A sequence-to-image based approach towards IP traffic classification using convolutional neural networks," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1271–1276, 2017. 6

[32] F. Wilhelmi, D. Góez, P. Soto, R. Vallés, M. Alfaifi, A. Algunayah, J. Martín-Pérez, L. Girletti, R. Mohan, K. V. Ramnan, *et al.*, "Machine learning for performance prediction of channel bonding in next-generation ieee 802.11 wlans," *ITU Journal on Future and Evolving Technologies*, vol. 2, no. 4, pp. 67–79, 2021. 6

[33] N. Bhushan, J. Li, D. Malladi, R. Gilmore, D. Brenner, A. Damnjanovic, R. T. Sukhavasi, C. Patel, and S. Geirhofer, "Network densification: the dominant theme for wireless evolution into 5g," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 82–89, 2014. 7

[34] Qualcomm, "Extending LTE Advanced to unlicensed spectrum," White-paper, Qualcomm, 2013. 7, 72

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017. 8

[36] M. Katzef, A. C. Cullen, T. Alpcan, C. Leckie, and J. Kopacz, "Wireless network simulation to create machine learning benchmark data," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pp. 6378–6383, IEEE, 2022. 8

[37] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010. 8

[38] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 project goals," in *Proceedings of the 2006 Workshop on ns-3*, pp. 13–es, 2006. 8

[39] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian conference on communications and computing (COLCOM)*, pp. 1–6, Ieee, 2014. 8

[40] J. Liu, Y. Shen, M. Simsek, B. Kantarci, H. T. Mouftah, M. Bagheri, and P. Djukic, "A new realistic benchmark for advanced persistent threats in network traffic," *IEEE Networking Letters*, vol. 4, no. 3, pp. 162–166, 2022. 8

[41] W. Garey, T. Ropitault, R. Rouil, E. Black, and W. Gao, "O-RAN with Machine Learning in ns-3," in *Proceedings of the 2023 Workshop on ns-3*, pp. 60–68, 2023. 8, 139

[42] M. Miozzo, N. Bartzoudis, M. Requena, O. Font-Bach, P. Harbanau, D. López-Bueno, M. Payaró, and J. Mangues, "SDR and NFV extensions in the ns-3 LTE module for 5G rapid prototyping," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, IEEE, 2018. 8, 139

[43] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "OpenRAN Gym: AI/ML development, data collection, and testing for O-RAN on PAWR platforms," *Computer Networks*, vol. 220, p. 109502, 2023. 8, 138

[44] S. Singh, J. Thaliath, I. F. Siddiqui, A. Jain, S. Yoon, M. Attique, and N. M. F. Qureshi, "Machine learning as a service for beyond 5g networks," in *2022 IEEE Globecom Workshops (GC Wkshps)*, pp. 455–460, IEEE, 2022. 10

[45] B. Arzani, K. Hsieh, and H. Chen, "Interpretable feedback for automl and a proposal for domain-customized automl for networking," in *Proceedings of the 20th ACM Workshop on Hot Topics in Networks*, HotNets '21, (New York, NY, USA), p. 53–60, Association for Computing Machinery, 2021. 10

[46] P. Li, J. Thomas, X. Wang, A. Khalil, A. Ahmad, R. Inacio, S. Kapoor, A. Parekh, A. Doufexi, A. Shojaeifard, *et al.*, "RLOps: Development life-cycle of reinforcement learning aided open RAN," *IEEE Access*, vol. 10, pp. 113808–113826, 2022. 10, 132, 133, 135, 163

[47] F. N. Khan, "Machine learning-enabled intelligent fiber-optic communications: major obstacles and the way forward," *IEEE Communications Magazine*, vol. 61, no. 4, pp. 122–128, 2022. 11

[48] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016. 11

[49] Databricks, "MLFlow," 2024. Accessed: 2024-01-29. 12, 49

[50] Google, "Kubeflow," 2024. Accessed: 2024-01-08. 12, 49

[51] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016. 12, 113, 121, 136, 137, 143, 163

[52] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009. 12, 137

[53] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler, "Long range arena: A benchmark for efficient transformers," *arXiv preprint arXiv:2011.04006*, 2020. 12, 137

[54] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018. 12, 137

[55] M. S. Gast, *802.11 Wireless Networks: The Definitive Guide*. O'Reilly Media, Inc., 2005. Second Edition. 20

[56] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher Speed Physical Layer (PHY) Extension in the 2.4 GHz band," standard, IEEE, 2000. 20, 22

[57] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6GHz," standard, IEEE, 2013. 20, 22

[58] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: Enhancements for High Efficiency WLAN," draft, IEEE, 2021. 20, 23, 72, 76

[59] ITU-R, "Radio Regulations," regulation, ITU-R, 2020. 20

[60] M. N. Sadiku, *Elements of electromagnetics*. Oxford University Press, USA, 2007. 21

[61] S. Barrachina-Munoz, F. Wilhelmi, and B. Bellalta, "Dynamic channel bonding in spatially distributed high-density WLANs," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 821–835, 2019. 23, 74, 77, 85

[62] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014. 25

[63] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015. 25

[64] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration," specification, ETSI, 2014. 25, 26, 27, 39, 44, 47, 51, 59, 64, 65, 160

[65] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management.," in *NSDI*, pp. 335–348, 2009. 25

[66] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014. 25

[67] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008. 26

[68] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE communications magazine*, vol. 53, no. 2, pp. 90–97, 2015. 26

[69] K. Mehmood, K. Kralevska, and D. Palma, "Intent-driven autonomous network and service management in future cellular networks: A structured literature review," *Computer Networks*, vol. 220, p. 109477, 2023. 27, 132, 134

[70] P. Goyal, R. Mikkilineni, and M. Ganti, "Fcaps in the business services fabric model," in *2009 18th IEEE international workshops on enabling technologies: infrastructures for collaborative enterprises*, pp. 45–51, IEEE, 2009. 27

[71] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. 28

[72] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018. 28, 35, 36

[73] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006. 28

[74] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008. 28, 79

[75] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016. 29

[76] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992. 36, 152

[77] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. 36

[78] P. Soto, M. Camelo, G. García-Avilés, E. Municio, M. Gramaglia, E. Kosmatos, N. Slamnik-Kriještorac, D. De Vleeschauwer, A. Bazco-Nogueras, L. Fuentes, J. Ballesteros, A. Lutu, L. Cominardi, I. Paez, S. Alcalá-Marín, L. E. Chatzieleftheriou, A. García-Saavedra, and M. Fiore, "Designing the network intelligence stratum for 6g networks," *Computer Networks*, vol. 254, p. 110780, 2024. 37

[79] M. Camelo, L. Cominardi, M. Gramaglia, M. Fiore, A. Garcia-Saavedra, L. Fuentes, D. De Vleeschauwer, P. Soto-Arenas, N. Slamnik-Krijestorac, J. Ballesteros, *et al.*, "Requirements and Specifications for the Orchestration of Network Intelligence in 6G," in *IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–9, IEEE, 2022. 38, 39, 43, 45, 132

[80] Y. Wang, R. Forbes, U. Elzur, J. Strassner, A. Gamelas, H. Wang, S. Liu, L. Pesando, X. Yuan, and S. Cai, "From design to practice: ETSI ENI reference architecture and instantiation for network management and orchestration using artificial intelligence," *IEEE Communications Standards Magazine*, vol. 4, no. 3, pp. 38–45, 2020. 38, 108

[81] 3GPP, "Architecture enhancements for 5G System (5GS) to support network data analytics services," Technical Specification (TS) 23.288, 3rd Generation Partnership Project (3GPP), December 2021. Version 17.3.0. 38, 47

[82] M. Gramaglia, M. Camelo, L. Fuentes, J. Ballesteros, G. Baldoni, L. Cominardi, A. Garcia-Saavedra, and M. Fiore, "Network intelligence for virtualized ran orchestration: The daemon approach," in *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pp. 482–487, IEEE, 2022. 39, 42

[83] TM FORUM, "Autonomous Networks Technical Architecture," Document IG1230, TM FORUM, 2022-12. 40, 43

[84] I. Paez, L. Cominardi, M. Camelo, P. Soto-Arenas, N. Slamnik-Krijestorac, M. Fiore, M. Gramaglia, A. Banchs, M. Molina, A. Hernandez, D. de Vleeschauwer, C.-Y. Chang, A. Garcia-Saavedra, A. Lutu, L. Fuentes, J. Ballesteros, A. Skalidi, A. Kostopoulos, and G. Iosifidis, "DAEMON Deliverable 2.1: Initial report on requirements analysis and state-of-the-art frameworks and toolsets," June 2021. 40

[85] M. Iovene, L. Jonsson, R. Dinand, M. D'Angelo, G. Hall, M. Erol-Kantarci, and J. Manocha, "Defining AI native: A key enabler for advanced intelligent telecom networks," Tech. Rep. BCSS-23:000056 Uen, Ericsson, Feb. 2023. 40, 43

[86] P. Li, Y. Xing, and W. Li, "Distributed AI-native Architecture for 6G Networks," in *2022 International Conference on Information Processing and Network Provisioning (ICIPNP)*, pp. 57–62, IEEE, 2022. 41, 43

[87] D. Rossi and L. Zhang, "Network artificial intelligence, fast and slow," in *Proceedings of the 1st International Workshop on Native Network Intelligence*, pp. 14–20, 2022. 41, 42, 43

[88] F. Brito, J. C. Cisneros, N. Linder, R. Riggio, E. Coronado, J. Palomares, J. Adzic, J. Renart, A. Lindgren, M. Rosa, *et al.*, "A network architecture for scalable end-to-end management of reusable AI-based applications," in *2023 14th International Conference on Network of the Future (NoF)*, pp. 98–102, IEEE, 2023. 41, 42, 43

[89] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Orchestrating Network Intelligence in the Open RAN," *IEEE Transactions on Mobile Computing*, 2023. 41, 43

[90] R. Bassoli, M. Ericson, H. Flinck, H. Harkous, B. M. Khorsandi, P. Pöyhönen, *et al.*, "Deliverable D5.2: Analysis of 6G architectural enablers' applicability and initial technological solutions," Oct. 2022. accessed: 2024-06-19. 41

[91] O. Akgul, B. Arar, M. D. Angelis, S. Barmpounakis, J. van de Beek, G. Bernini, *et al.*, "Deliverable D3.3 Initial analysis of architectural enablers and framework," Apr. 2024. accessed: 2024-06-19. 42

[92] S. Stańczak, Z. Utkovski, *et al.*, "Toward 6G: Key Directions and Research Questions," 2022. accessed: 2024-06-19. 42

[93] T. Taleb, R. L. Aguiar, I. G. B. Yahia, B. Chatras, G. Christensen, U. Chunduri, *et al.*, "White Paper on 6G Networking. 6G Research Visions," 2020. accessed: 2024-06-19. 42

[94] L. E. Chatzieleftheriou, M. Gramaglia, M. Camelo, A. Garcia-Saavedra, E. Kosmatos, M. Gucciardo, P. Soto, G. Iosifidis, L. Fuentes, G. Garcia-Aviles, *et al.*, "Orchestration procedures for the network intelligence stratum in 6g networks," in *European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pp. 347–352, IEEE, 2023. 43, 49, 51, 132, 154

[95] B. M. Khorsandi, M. Uusitalo, M.-H. Hamon, B. Richerzhagen, G. D'Aria, A. Gati, *et al.*, "Deliverable D1.4: Hexa-X architecture for B5G/6G networks – final release," July 2023. accessed: 2024-06-19. 43

[96] A. Garcia-Saavedra and X. Costa-Perez, "O-RAN: Disrupting the virtualized RAN ecosystem," *IEEE Communications Standards Magazine*, vol. 5, no. 4, pp. 96–103, 2021. 47

[97] G. Garcia-Aviles, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, P. Serrano, and A. Banchs, "Nuberu: Reliable RAN virtualization in shared platforms," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pp. 749–761, 2021. 49

[98] A. Garcia-Saavedra, J. X. Salvat, D. D. Vleeschauwer, C.-Y. Chang, L. Fuentes, D.-J. Munoz, A. Lutu, M. Gucciardo, M. Fiore, L. E. Chatzieleftheriou, N. Slamnik-Kriještorac, P. Soto, M. Camelo, M. Gramaglia, G. Garcia-Aviles, E. Municio, and N. Mhaisen, "DAEMON Deliverable 3.2: Refined design of real- time control and VNF intelligence mechanisms," Nov. 2022. 49

[99] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016. 49

[100] L. Fuentes, M. Amor, Ángel Cañete, M. Fiore, S. Alcalá, S. Barmpounakis, I. Chondroulis, I. Belikaidis, E. Kosmatos, A. G. Saavedra, J. X. Salvat, M. Camelo, P. Soto, A. Lutu, G. Iosifidis, D. D. Vleeschauwer, C.-Y. Chang, and A. Pentelas, "DAEMON Deliverable 4.2: Refined design of intelligent orchestration and management mechanisms," Jan. 2023. 49

[101] O-RAN Alliance, "O-RAN Working Group 2 AI/ML workflow description and requirements," technical report, O-RAN Alliance, Oct, 2020. 51, 55

[102] M. Polese, L. Bonati, S. D'oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Communications Surveys & Tutorials*, 2023. 55

[103] D. M. Manias, A. Chouman, and A. Shami, "Model drift in dynamic networks," *IEEE Communications Magazine*, 2023. 62, 166

[104] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, "Age of information: An introduction and survey," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 5, pp. 1183–1210, 2021. 62

[105] P. B. del Prever, S. D'Oro, L. Bonati, M. Polese, M. Tsampazi, H. Lehmann, and T. Melodia, "Pacifista: Conflict evaluation and management in open ran," *arXiv preprint arXiv:2405.04395*, 2024. 66

[106] ETSI, "Experiential Networked Intelligence (ENI); System Architecture ," specification, ETSI, Dec, 2021. 67

[107] P. Soto, M. Camelo, J. Fontaine, M. Girmay, A. Shahid, V. Maglogiannis, E. De Poorter, I. Moerman, J. F. Botero, and S. Latré, "Augmented Wi-Fi: An AI-based Wi-Fi management framework for Wi-Fi/LTE coexistence," in *IEEE International Conference on Network and Service Management (CNSM)*, pp. 1–9, IEEE, 2020. 71

[108] P. Soto, M. Camelo, K. Mets, F. Wilhelmi, D. Góez, L. A. Fletscher, N. Gaviria, P. Hellinckx, J. F. Botero, and S. Latré, "ATARI: A graph convolutional neural network approach for performance prediction in next-generation WLANs," *Sensors*, vol. 21, no. 13, p. 4321, 2021. 71

[109] M. Camelo, A. Shahid, J. Fontaine, F. A. P. de Figueiredo, E. De Poorter, I. Moerman, and S. Latre, "A semi-supervised learning approach towards automatic wireless technology recognition," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–10, IEEE, 2019. 71, 94, 95, 97, 105

[110] Cisco, "Cisco Annual Internet Report (2018–2023) White Paper," 2020. 72

[111] Deek, Lara and Garcia-Villegas, Eduard and Belding, Elizabeth and Lee, Sung-Ju and Almeroth, Kevin, "Intelligent Channel Bonding in 802.11 n WLANs," *IEEE Transactions on Mobile Computing*, vol. 13, no. 6, pp. 1242–1255, 2013. 72, 73

[112] LTE-U Forum, "LTE-U Technical Report Coexistence Study for LTE-U SDL V1.0," tech. rep., LTE-U Forum, 2015. 73

[113] 3-GPP, "Evolved Universal Terrestrial Radio Access(EUTRA); Physical Layer Procedures (Release 13) V13.4.0," Standard TS 36.213, 3-GPP, 2016. 73

[114] MulteFire Alliance, "The multefire specifications." 73

[115] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," standard, IEEE, March 2012. 73, 96

[116] Y. Daldoul, D.-E. Meddour, and A. Ksentini, "IEEE 802.11 ac: Effect of channel bonding on spectrum utilization in dense environments," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2017. 74

[117] A. Faridi, B. Bellalta, and A. Checco, "Analysis of dynamic channel bonding in dense networks of WLANs," *IEEE Transactions on Mobile Computing*, vol. 16, no. 8, pp. 2118–2131, 2016. 74

[118] M. A. Khan, R. Hamila, N. A. Al-Emadi, S. Kiranyaz, and M. Gabbouj, "Real-time throughput prediction for cognitive Wi-Fi networks," *Journal of Network and Computer Applications*, vol. 150, p. 102499, 2020. 74, 75, 83

[119] J. Herzen, H. Lundgren, and N. Hegde, "Learning Wi-Fi performance," in *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 118–126, IEEE, 2015. 74, 75, 83

[120] Y. Luo and K.-W. Chin, "Learning to Bond in Dense WLANs With Random Traffic Demands," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 11868–11879, 2020. 74, 75

[121] T. Nihtilä, V. Tykhomyrov, O. Alanen, M. A. Uusitalo, A. Sorri, M. Moisio, S. Iraji, R. Ratasuk, and N. Mangalvedhe, "System performance of lte and ieee 802.11 coexisting on a shared frequency band," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1038–1043, IEEE, 2013. 75, 76

[122] A. Babaei, J. Andreoli-Fang, Y. Pang, and B. Hamzeh, "On the impact of lte-u on wi-fi performance," *International Journal of Wireless Information Networks*, vol. 22, no. 4, pp. 336–344, 2015. 75, 76, 93

[123] J. Jeon, H. Niu, Q. C. Li, A. Papathanassiou, and G. Wu, "Lte in the unlicensed spectrum: Evaluating coexistence mechanisms," in *2014 IEEE Globecom Workshops (GC Wkshps)*, pp. 740–745, IEEE, 2014. 75, 76

[124] S. Sagari, S. Baysting, D. Saha, I. Seskar, W. Trappe, and D. Raychaudhuri, "Coordinated dynamic spectrum management of lte-u and wi-fi networks," in *2015 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 209–220, IEEE, 2015. 75, 76

[125] C. Capretti, F. Gringoli, N. Facchi, and P. Patras, "Lte/wi-fi co-existence under scrutiny: an empirical study," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, pp. 33–40, 2016. 75, 76

[126] P. M. de Santana, V. A. de Sousa, F. M. Abinader, and J. M. d. C. Neto, "Dm-csat: A lte-u/wi-fi coexistence solution based on reinforcement learning," *Telecommunication Systems*, vol. 71, no. 4, pp. 615–626, 2019. 75, 76

[127] V. Maglogiannis, D. Naudts, A. Shahid, and I. Moerman, "A q-learning scheme for fair coexistence between lte and wi-fi in unlicensed spectrum," *IEEE Access*, vol. 6, pp. 27278–27293, 2018. 75, 76

[128] V. Maglogiannis, A. Shahid, D. Naudts, E. De Poorter, and I. Moerman, "Enhancing the coexistence of lte and wi-fi in unlicensed spectrum through convolutional neural networks," *IEEE Access*, vol. 7, pp. 28464–28477, 2019. 76

[129] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019. 76

[130] A. M. Voicu, L. Simić, and M. Petrova, "Survey of spectrum sharing for inter-technology coexistence," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1112–1144, 2018. 76

[131] B. Bellalta, "IEEE 802.11 ax: High-efficiency WLANs," *IEEE Wireless Communications*, vol. 23, no. 1, pp. 38–46, 2016. 76

[132] D. López-Pérez, A. Garcia-Rodriguez, L. Galati-Giordano, M. Kasslin, and K. Doppler, "IEEE 802.11 be extremely high throughput: The next generation of Wi-Fi technology beyond 802.11 ax," *IEEE Communications Magazine*, vol. 57, no. 9, pp. 113–119, 2019. 76

[133] ITU-R, "IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond," recommendation, ITU, 2015. 77

[134] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 79, 90

[135] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016. 80

[136] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018. 80

[137] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN," in *Proceedings of the 2019 ACM Symposium on SDN Research*, pp. 140–151, 2019. 80

[138] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016. 80

[139] F. Wilhelmi, "[ITU-T AI Challenge] Input/Output of project "Improving the capacity of IEEE 802.11 WLANs through Machine Learning," Jun 2020. Accessed: 2020-12-06. 83

[140] S. Barrachina-Muñoz and F. Wilhelmi, "Komondor: An IEEE 802.11ax Simulator." `https://github.com/wn-upf/Komondor`, 2019. Commit: d330ed9. 83, 85

[141] S. Barrachina-Munoz, F. Wilhelmi, I. Selinis, and B. Bellalta, "Komondor: A wireless network simulator for next-generation high-density WLANs," in *2019 Wireless Days (WD)*, pp. 1–8, IEEE, 2019. 83

[142] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008. 83

[143] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE Journal on selected areas in communications*, vol. 18, no. 3, pp. 535–547, 2000. 83

[144] B. Bellalta, A. Zocca, C. Cano, A. Checco, J. Barcelo, and A. Vinel, "Throughput analysis in CSMA/CA networks using continuous time Markov networks: A tutorial," *Wireless Networking for Moving Objects*, pp. 115–133, 2014. 83

[145] S. Barrachina-Muñoz, B. Bellalta, and E. Knightly, "Wi-Fi All-Channel Analyzer," in *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*, pp. 72–79, 2020. 84

[146] IEEE, "TGax Simulation Scenarios. Doc.: IEEE 802.11-14/0980r16." `https://mentor.ieee.org/802.11/dcn/14/11-14-0980-16-00ax-simulation-scenarios.docx`, 2015. Online; accessed 15 June 2021. 84

[147] T. Adame, M. Carrascosa, and B. Bellalta, "The tmb path loss model for 5 ghz indoor wifi scenarios: On the empirical relationship between rssi, mcs, and spatial streams," in *2019 Wireless Days (WD)*, pp. 1–8, IEEE, 2019. 85

[148] Deek, Lara and Garcia-Villegas, Eduard and Belding, Elizabeth and Lee, Sung-Ju and Almeroth, Kevin, "The impact of channel bonding on 802.11 n network management," in *Proceedings of the 7th Conference on emerging Networking Experiments and Technologies*, pp. 1–12, 2011. 88

[149] F. M. Abinader, E. P. Almeida, F. S. Chaves, A. M. Cavalcante, R. D. Vieira, R. C. Paiva, A. M. Sobrinho, S. Choudhury, E. Tuomaala, K. Doppler, *et al.*, "Enabling the coexistence of lte and wi-fi in unlicensed bands," *IEEE Communications Magazine*, vol. 52, no. 11, pp. 54–61, 2014. 93

[150] Q.-D. Ho, D. Tweed, and T. Le-Ngoc, *Long Term Evolution in unlicensed bands*. Springer, 2017. 93

[151] M. Camelo, R. Mennes, A. Shahid, J. Struye, C. Donato, I. Jabandžić, S. Giannoulis, F. Mahfoudhi, P. Maddala, I. Seskar, I. Moerman, and S. Latré, "An ai-based incumbent protection system for collaborative intelligent radio networks," *IEEE Wireless Communications Magazine*, vol. 27, no. 5, 2020. 93

[152] A. Shahid, J. Fontaine, M. Camelo, J. Haxhibeqiri, M. Saelens, Z. Khan, I. Moerman, and E. De Poorter, "A convolutional neural network approach for classification of lpwan technologies: Sigfox, lora and ieee 802.15. 4g," in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 1–8, IEEE, 2019. 93

[153] N. Golmie, R. E. Van Dyck, A. Soltanian, A. Tonnerre, and O. Rebala, "Interference evaluation of bluetooth and ieee 802.11 b systems," *Wireless Networks*, vol. 9, no. 3, pp. 201–211, 2003. 93

[154] M. Girmay, V. Maglogiannis, D. Naudts, J. Fontaine, A. Shahid, E. De Poorter, and I. Moerman, "Adaptive cnn-based private lte solution for fair coexistence with wi-fi in unlicensed spectrum," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 346–351, IEEE, 2020. 97, 98

[155] ZeroMQ, "ZeroMQ." Accessed: 2024-03-28. 98

[156] Google, "Protocol Buffers." Accessed: 2024-03-28. 98, 138

[157] IETF, "Intent-Based Networking - Concepts and Overview," draft, IETF, 2019. 104

[158] K. Hornik, M. Stinchcombe, H. White, *et al.*, "Multilayer feedforward networks are universal approximators.," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. 104

[159] P. Soto, D. De Vleeschauwer, M. Camelo, Y. De Bock, K. De Schepper, C.-Y. Chang, P. Hellinckx, J. F. Botero, and S. Latré, "Towards autonomous VNF auto-scaling using deep reinforcement learning," in *Eighth International Conference on Software Defined Systems (SDS)*, pp. 01–08, IEEE, 2021. 107

[160] P. Soto, M. Camelo, D. De Vleeschauwer, Y. De Bock, C.-Y. Chang, J. F. Botero, and S. Latré, "Network Intelligence for NFV Scaling in Closed-Loop Architectures," *IEEE Communications Magazine*, vol. 61, no. 6, pp. 66–72, 2023. 107

[161] M. S. Bonfim, K. L. Dias, and S. F. Fernandes, "Integrated NFV/SDN architectures: A systematic literature review," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–39, 2019. 108

[162] ETSI, "Open Source MANO (OSM)." Accessed: 2021-08-23. 108

[163] O. Adamuz-Hinojosa, J. Ordonez-Lucena, P. Ameigeiras, J. J. Ramos-Munoz, D. Lopez, and J. Folgueira, "Automated network service scaling in nfv: Concepts, mechanisms and scaling workflow," *IEEE Communications Magazine*, vol. 56, no. 7, pp. 162–169, 2018. 109, 116

[164] T. L. Duc, R. G. Leiva, P. Casari, and P.-O. Östberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–39, 2019. 109, 110

[165] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of grid computing*, vol. 12, no. 4, pp. 559–592, 2014. 110, 120

[166] T. Chen, R. Bahsoon, and X. Yao, "A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems," *arXiv preprint arXiv:1609.03590*, 2016. 110

[167] S. Dutta, T. Taleb, and A. Ksentini, "Qoe-aware elasticity support in cloud-native 5g systems," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2016. 110

[168] D. De Vleeschauwer, J. Baranda, J. Mangues-Bafalluy, C. F. Chiasserini, M. Malinverno, C. Puligheddu, L. Magoula, J. Martín-Pérez, S. Barmpounakis, K. Kondepu, *et al.*, "5growth data-driven ai-based scaling," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pp. 383–388, IEEE, 2021. 110

[169] J. Martín-Pérez, K. Kondepu, D. De Vleeschauwer, V. Reddy, C. Guimarães, A. Sgambelluri, L. Valcarenghi, C. Papagianni, and C. J. Bernardos, "Dimensioning of v2x services in 5g networks through forecast-based scaling," *arXiv preprint arXiv:2105.12527*, 2021. 111

[170] S. Lange, H.-G. Kim, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong, "Machine learning-based prediction of vnf deployment decisions in dynamic networks," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–6, IEEE, 2019. 111

[171] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling vnfs using machine learning to improve qos and reduce cost," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018. 111

[172] T. Subramanya and R. Riggio, "Centralized and federated learning for predictive vnf autoscaling in multi-domain 5g networks and beyond," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 63–78, 2021. 111

[173] T. Subramanya and R. Riggio, "Machine learning-driven scaling and placement of virtual network functions at the network edges," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, pp. 414–422, IEEE, 2019. 111

[174] H. Sami, H. Otrok, J. Bentahar, and A. Mourad, "Ai-based resource provisioning of ioe services in 6g: A deep reinforcement learning approach," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3527–3540, 2021. 111

[175] Y. Garí, D. A. Monge, E. Pacini, C. Mateos, and C. G. Garino, "Reinforcement learning-based application autoscaling in the cloud: A survey," *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104288, 2021. 111

[176] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015. 111, 140, 152

[177] D. Lee, J.-H. Yoo, and J. W.-K. Hong, "Deep q-networks based auto-scaling for service function chaining," in *2020 16th International Conference on Network and Service Management (CNSM)*, pp. 1–9, IEEE, 2020. 111

[178] C. H. T. Arteaga, F. Risso, and O. M. C. Rendon, "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)*, pp. 1–7, IEEE, 2017. 112

[179] P. Gabriela, D. Lee, N. Van Tu, and J. W.-K. Hong, "Machine learning-based auto-scaler for video conferencing systems," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pp. 142–150, IEEE, 2021. 112

[180] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 329–338, IEEE, 2019. 112

[181] L. He, L. Li, and Y. Liu, "Towards chain-aware scaling detection in nfv with reinforcement learning," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pp. 1–10, IEEE, 2021. 112

[182] A. A. Khaleq and I. Ra, "Intelligent autoscaling of microservices in the cloud for real-time applications," *IEEE Access*, vol. 9, pp. 35464–35476, 2021. 113

[183] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "gym-hpa: Efficient auto-scaling via reinforcement learning for complex microservice-based applications in kubernetes," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, 2023. 113, 139, 165

[184] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 12348–12355, 2021. 113, 124, 136, 144, 163

[185] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 497–511, 2012. 115

[186] T. Song, D. Kaleshi, R. Zhou, O. Boudeville, J.-X. Ma, A. Pelletier, and I. Haddadi, "Performance evaluation of integrated smart energy solutions through large-scale simulations," in *2011 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 37–42, IEEE, 2011. 121, 143

[187] P. Soto, M. Camelo, D. De Vleeschauwer, Y. De Bock, N. Slamnik-Krieštorac, C.-Y. Chang, N. Gaviria, E. Mannens, J. F. Botero, and S. Latré, "Designing, Developing, and Validating Network Intelligence for Scaling in Service-Based Architectures based on Deep Reinforcement Learning," *arXiv preprint arXiv:2405.04441*, 2024. 131

[188] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6g networks: Use cases and technologies," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, 2020. 132

[189] T. Taleb, C. Benzaïd, R. A. Addad, and K. Samdanis, "Ai/ml for beyond 5g systems: Concepts, technology enablers & solutions," *Computer Networks*, vol. 237, p. 110044, 2023. 132

[190] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," *IEEE Access*, 2023. 134, 163

[191] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-based systems*, vol. 212, p. 106622, 2021. 135

[192] J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, *et al.*, "Automated reinforcement learning (autorl): A survey and open problems," *Journal of Artificial Intelligence Research*, vol. 74, pp. 517–568, 2022. 135, 137

[193] O-RAN WG2, "O-RAN AI/ML workflow description and requirements - v1.02," technical report, O-RAN, 2020. 135, 144, 163

[194] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in Deep RL: A case study on PPO and TRPO," in *nternational Conference on Learning Representations*, pp. 1–14, 2020. 136, 144

[195] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *arXiv preprint arXiv:1708.04133*, 2017. 136, 138, 145

[196] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32(1), pp. 3207–3214, 2018. 136, 137, 145

[197] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019. 136

[198] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *arXiv preprint arXiv:1705.04304*, 2017. 136

[199] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," *arXiv preprint arXiv:1704.03073*, 2017. 136

[200] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012. 137

[201] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019. 137

[202] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019. 137, 150

[203] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 9, pp. 5149–5169, 2021. 138

[204] P. Gawłowicz and A. Zubow, "Ns-3 meets openai gym: The playground for machine learning in networking research," in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 113–120, 2019. 138, 163

[205] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*, pp. 15–34, Springer, 2010. 138

[206] H. Yin, P. Liu, K. Liu, L. Cao, L. Zhang, Y. Gao, and X. Hei, "ns3-ai: Fostering artificial intelligence algorithms for networking research," in *Proceedings of the 2020 Workshop on ns-3*, pp. 57–64, 2020. 138

[207] L. Bonati, P. Johari, M. Polese, S. D'Oro, S. Mohanti, M. Tehrani-Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder, *et al.*, "Colosseum: Large-scale wireless experimentation through hardware-in-the-loop network emulation," in *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 105–113, IEEE, 2021. 138

[208] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. 140, 152

[209] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. 140

[210] J. Achiam, "Spinning up in Deep Reinforcement Learning." `https://spinningup.openai.com/`, 2018. Accessed: 2024-01-03. 144

[211] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "How many random seeds? statistical power analysis in deep reinforcement learning experiments," *arXiv preprint arXiv:1806.08295*, 2018. 145, 150

[212] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms," *arXiv preprint arXiv:1904.06979*, 2019. 145, 147

[213] P. Rochet and I. Serra, "The mean/max statistic in extreme value analysis," *arXiv preprint arXiv:1606.08974*, 2016. 149

[214] H. S. Jomaa, J. Grabocka, and L. Schmidt-Thieme, "Hyp-rl: Hyperparameter optimization by reinforcement learning," *arXiv preprint arXiv:1906.11527*, 2019. 150

[215] Y. Yu, "Towards Sample Efficient Reinforcement Learning.," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 5739–5743, 2018. 150

[216] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016. 156

[217] J. Eschmann, "Reward function design in Reinforcement Learning," *Reinforcement Learning Algorithms: Analysis and Applications*, pp. 25–33, 2021. 156

[218] W. Cui and W. Yu, "Reinforcement Learning with Non-Cumulative Objective," *IEEE Transactions on Machine Learning in Communications and Networking*, 2023. 157

[219] U. Altaf, G. Jayaputera, J. Li, D. Marques, D. Meggyesy, S. Sarwar, S. Sharma, W. Voorsluys, R. Sinnott, A. Novak, *et al.*, "Auto-scaling a defence application across the cloud using docker and kubernetes," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pp. 327–334, IEEE, 2018. 162

[220] M. Gotin, F. Lösch, R. Heinrich, and R. Reussner, "Investigating performance metrics for scaling microservices in cloudiot-environments," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pp. 157–167, 2018. 162

[221] P. Kurrek, F. Zoghlami, M. Jocas, M. Stoelen, and V. Salehi, "Q-model: An artificial intelligence based methodology for the development of autonomous robots," *Journal of Computing and Information Science in Engineering*, vol. 20, no. 6, p. 061006, 2020. 163

[222] E. Zeydan and J. Mangues-Bafalluy, "Recent advances in data engineering for networking," *IEEE Access*, vol. 10, pp. 34449–34496, 2022. 164

[223] V. Zalokostas-Diplas, N. Makris, V. Passas, and T. Korakis, "Experimental evaluation of ml models for dynamic vnf autoscaling," in *2022 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 157–162, IEEE, 2022. 165

[224] S. Schmid, M. Sifalakis, and D. Hutchison, "Towards autonomic networks," in *Autonomic Networking: First International IFIP TC6 Conference, AN 2006, Paris, France, September 27-29, 2006. Proceedings*, pp. 1–11, Springer, 2006. 165

[225] B. Brik, H. Chergui, L. Zanzi, F. Devoti, A. Ksentini, M. S. Siddiqui, X. Costa-Pérez, and C. Verikoukis, "A survey on explainable AI for 6G O-RAN: Architecture, use cases, challenges and research directions," *arXiv preprint arXiv:2307.00319*, 2023. 165

[226] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez, "Explainability in deep reinforcement learning," *Knowledge-Based Systems*, vol. 214, p. 106685, 2021. 165

[227] J. Kaddour, A. Lynch, Q. Liu, M. J. Kusner, and R. Silva, "Causal machine learning: A survey and open problems," *arXiv preprint arXiv:2206.15475*, 2022. 166

[228] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, pp. 1–40, 2016. 166

[229] F. Pourpanah, M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, X.-Z. Wang, and Q. J. Wu, "A review of generalized zero-shot learning methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 4, pp. 4051–4070, 2022. 166

[230] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–35, 2021. 166

[231] S. Ackerman, O. Raz, M. Zalmanovici, and A. Zlotnick, "Automatically detecting data drift in machine learning classifiers," *arXiv preprint arXiv:2111.05672*, 2021. 166

[232] A. Maatouk, N. Piovesan, F. Ayed, A. De Domenico, and M. Debbah, "Large language models for telecom: Forthcoming impact on the industry," *IEEE Communications Magazine*, 2024. 166

[233] TM FORUM, "Intent Management," Document TMF921, TM FORUM, 2024-04. 166

[234] ITU-T, "Architectural framework for machine learning sandbox in future networks, including IMT-2020," recommendation, ITU, 2022. 167

[235] M. Ferriol-Galmés, J. Suárez-Varela, J. Paillissé, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Building a digital twin for network optimization using graph neural networks," *Computer Networks*, vol. 217, p. 109329, 2022. 167

[236] M. S. Rodrigo, D. Rivera, J. I. Moreno, M. Álvarez-Campana, and D. R. López, "Digital Twins for 5G Networks: A modeling and deployment methodology," *IEEE Access*, 2023. 167

[237] P. Jia, X. Wang, and X. Shen, "Accurate and efficient digital twin construction using concurrent end-to-end synchronization and multi-attribute data resampling," *IEEE Internet of Things Journal*, vol. 10, no. 6, pp. 4857–4870, 2022. 167

[238] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021. 167

[239] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106854, 2020. 167