Designing the Network Intelligence Stratum for 6G Networks

Paola Soto[a,b], Miguel Camelo[a], Gines Garcia-Aviles[c], Esteban Municio[c], Marco Gramaglia[d], Evangelos Kosmatos[e], Nina Slamnik-Kriještorac[a], Danny De Vleeschauwer[f], Antonio Bazco-Nogueras[g], Lidia Fuentes[h], Joaquin Ballesteros[h], Andra Lutu[i], Luca Cominardi[j], Ivan Paez[j], Sergi Alcalá-Marín[g,d], Livia Elena Chatzieleftheriou[g], Andres Garcia-Saavedra[k], Marco Fiore[g]

[a] *University of Antwerp - imec, IDLab, Antwerp, Belgium*
[b] *Universidad de Antioquia, Medellín, Colombia*
[c] *i2CAT, Barcelona, Spain*
[d] *University Carlos III, Madrid, Spain*
[e] *WINGS ICT Solutions, Athens, Greece*
[f] *Nokia Bell Labs, Antwerp, Belgium*
[g] *IMDEA Networks Institute, Madrid, Spain*
[h] *Universidad de Málaga, Malaga, Spain*
[i] *Telefonica Investigacion y Desarrollo SA, Madrid, Spain*
[j] *ZettaScale Technology, Saint Aubin, France*
[k] *NEC Laboratories Europe GmbH, Madrid, Spain*

Abstract

As network complexity escalates, there is an increasing need for more sophisticated methods to manage and operate these networks, focusing on enhancing efficiency, reliability, and security. A wide range of Artificial Intelligence (AI)/Machine Learning (ML) models are being developed in response. These models are pivotal in automating decision-making, conducting predictive analyses, managing networks proactively, enhancing security, and optimizing network performance. They are foundational in shaping the future of networks, collectively forming what is known as Network Intelligence (NI). 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. This paper introduces an in-depth architectural design for a Network Intelligence Stratum (NI Stratum). This stratum is supported by a novel end-to-end NI orchestrator that supports closed-loop NI 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 and demonstrate a reference implementation of the NI Stratum, focusing on its compatibility and integration with current network systems and open-source platforms such as Kubernetes and Kubeflow, as well as on its validation on real-world environments. The paper also outlines major challenges and open issues in deploying and managing NI.

Keywords:
Network Intelligence, 6G Stratum, AI-native network architecture, Network Intelligence Orchestration

1. Introduction

To meet the stringent demands imposed by upcoming services such as multisensory eXtended Reality (XR) applications and connected robotics [1], 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 autonomously 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 intelligent networking, referred to as NI. More specifically, an instance of NI is described as a sequence of efficient AI/ML algorithms that rapidly identify or predict new requests or fluctuations in network activities [2]. 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 real-

Email address: paola.soto-arenas@uantwerpen.be (Paola Soto)

ization 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 [3].

Ongoing initiatives led by major SDOs to incorporate NI into next-generation network architectures invariably revolve around the concept of "closed-loop AI" [4]. 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. This hinders the practical adoption of NI within 6G networks and calls for original enhancements to the network architectural paradigm. In particular, it is fundamental that a native integration of NI algorithms into the overall mobile network architecture is considered to fully support automation capabilities in future generations of communication systems.

This paper presents the complete architectural design and supporting procedures to realize a NI Stratum for 6G networks. Specifically, the contributions of this paper are fourfold:

- A complete architectural design of a NI Stratum, including the Network Intelligence Orchestration (NIO) and the tools to define and design NI in a unified manner. This paper integrates the findings and results of our preliminary work [2, 5, 6, 7], presenting them in a condensed and cohesive format. In addition, compared to state-of-the-art works (see Table 1), our NI Stratum tackles simultaneously multiple challenges of NI such as how to properly define NI, how to manage their life-cycle, and how to coordinate them via well-defined procedures.
- We define a set of internal interfaces to facilitate the interaction among functionalities in the NI Stratum. Moreover, we analyze how the external interfaces may be realized when interacting with the Radio Access Network (RAN) and Core network segments. To the best knowledge of the authors, this is the first work that provides such interfaces.
- We provide a detailed description of several inter- and intra-NIO procedures that are required to realize the NI Stratum. Compared to our previous work [7], we not only define them as functionalities but also provide the required workflows and sequence diagrams that allow us to perform these procedures.
- We implement and demonstrate [8] some of the capabilities of the NI Stratum via two use cases. While the orchestration of Network Intelligence Service (NIS) with support of ML pipelines was part of our previous work [7], we include a new implementation where we show the capabilities of the NI Stratum to orchestrate two Network Intelligence Function (NIF) and realize a NIS in the context of service orchestration at the edge.

The remainder of this paper is organized as follows. Section 2 reviews and summarizes the research and standardization efforts in defining an AI-native architecture. Section 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 4 presents internal and external interfaces needed to perform such orchestration. Using the functional blocks and the interfaces defined in the previous sections, Section 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, Section 6 shows a reference implementation of the proposed NI Stratum. Section 7 concludes the paper and gives an overview of the future challenges.

[9], [10, 11, 12, 13, 14, 15, 16, 17, 18]

2. Research and standardization efforts towards the integration of NI in mobile networks

The vision of Beyond 5G (B5G) and 6G networks largely builds on conflict-free and synergic operation among various NI algorithms across network schedulers, controllers, and orchestrators [2, 6]. This section reviews current efforts from SDOs and academia that aim to facilitate joint and end-to-end NI operation.

Current network management frameworks proposed by major SDOs present deficiencies when integrating and managing NI approaches. Our examination, detailed in [5] and in Section 5 and Appendix B of [19], reveals that standards and platforms from entities like ETSI, O-RAN, and 3GPP, including implementations such as Open Source MANO (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) and only early consideration for methodologies defining and representing NI models (e.g., ETSI-Experiential Networked Intelligence (ENI)). Table 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, scalable, and autonomous network operations. By examining prior works, this paper 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 [20]. 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 ML Operations (MLOps) functions for comprehensive end-to-end model lifecycle management within an AI-native architecture.

In [21], 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. [22] 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 network intelligence. 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 [23]. 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 [24]. Specifically, the authors showed that the intelligence orchestration problem is NP-hard and proposed three complexity reduction techniques. Deployed as a rApp in the non-Real-Time (RT) RAN Intelligent Controller (RIC), *OrchestRAN* empowers Network Operators (NOs) to define high-level control and inference objectives. *OrchestRAN* autonomously 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.

2.1. Differences with previous works

In the preceding sections, we examined the ongoing endeavors of prominent standardization bodies and academic institutions in formulating an AI-native architecture. This subsection aims to underscore the distinctions between our work and these existing initiatives. Specifically, we analyze crucial aspects currently overlooked by present management frameworks and briefly elaborate on how our research addresses these gaps. Table 1 summarizes the main differences between our work and the reviewed literature.

One key differentiator between the proposed architecture and the related work is how we define NI. Most works assume the NI instance as a single block, not a composition of multiple blocks. Thanks to the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) control model, we can decompose the NI in atomic elements that can be deployed across the whole 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 NISs, and helps identify conflicts as demonstrated in the Radio Access Network virtualization (vRAN) use case presented in [6]. A similar approach is presented in [22], where their DIKW pyramid resembles the MAPE-K but lacks the concept of multiple closed-loop control.

Regarding the NI lifecycle management, Brito et al. [23] provided appropriate workflows for NIF/NIS instantiation and replacement. In this paper, we go a step further by providing the necessary workflows for the complete NI lifecycle, from its creation until its termination. Although Li et al. [21] 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.

A common denominator among the research work is the importance of data. In [20], the pervasive nature of the NI is intricately linked to a distributed data infrastructure. The ability to

Table 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
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 [20]	No	Partially	Partially	Partially	No
Li et al. [21]	No	Partially	Partially	Partially	Partially
Rossi et al. [22]	Partially	No	Partially	Partially	No
Brito et al. [23]	No	Yes	Yes	Yes	No
D'oro et al. – OrchestRAN [24]	No	Partially	Partially	No	Yes
Network Intelligence Stratum (Our Work)	**Yes [2, 5]**	**Yes (This work)**	**Yes [5, 7] (Extended in this work)**	**Yes [2]**	**Yes (This work)**

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 [22]. Despite its importance, the studied works do not present a decentralized and unified data management framework for the NI instance. On the contrary, in [2], 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.

The work presented in this paper summarizes our previous work regarding the methodology to define NI, the importance of data, and the mechanisms to coordinate NI instances. Additionally, it extends our previous work by providing the interfaces that allow the stratum to provide communication between their functionalities (internals) and with external entities. Moreover, we provide a set of detailed procedure workflows for managing the NI lifecycle, including detailed procedures to resolve conflicts among multiple and colliding NI and knowledge sharing. Moreover, we extend the reference implementation by considering the necessary extensions to support conflict resolution capabilities in top open-source platforms such as Kubernetes [25], Kubeflow [26], and Zenoh [27].

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.

3.1. 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. 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 an NI instance implements decision-making functionality for deployment in a controller, Network Function Virtualization (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 1. A NIF may engage in

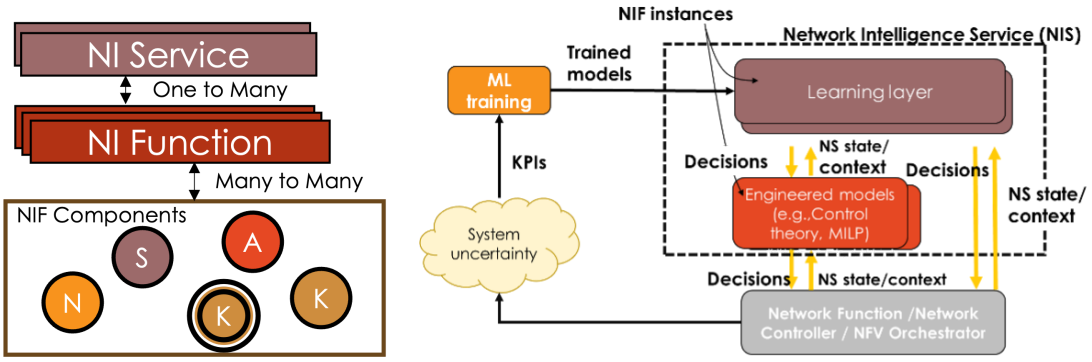

Figure 1: The high-level hierarchical taxonomy of NI algorithms. An NIF corresponds to an individual NI instance that assists a specific functionality; for example, it could capture the implementation of a capacity forecasting task, assisting an NI edge orchestration functionality.

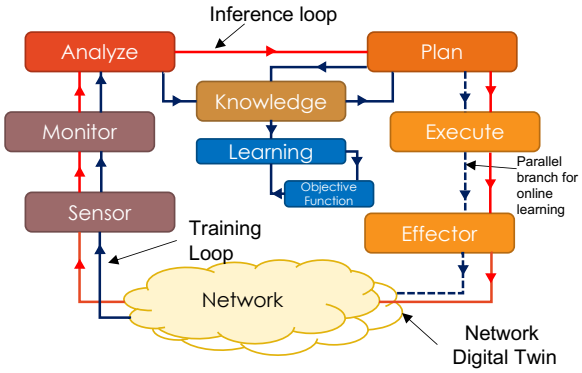

Figure 2: Extended N-MAPE-K abstractions for NI algorithms.

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 close-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 [28], yet cannot fully support the specifications of NIFs internals. Therefore, we introduce an extended Network Monitor-Analyze-Plan-Execute over a shared Knowledge (N-MAPE-K) [2] model tailored to the NI environment, which augments the legacy MAPE-K with original training and closed control loops that a NIF may implement, as shown in Figure 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.

Mapping NI algorithm components into the N-MAPE-K representation allows highlighting the following fundamental classes of atomic NIF Components (NIF-Cs).

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

3.2. *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 Management and Orchestration (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; 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*, *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

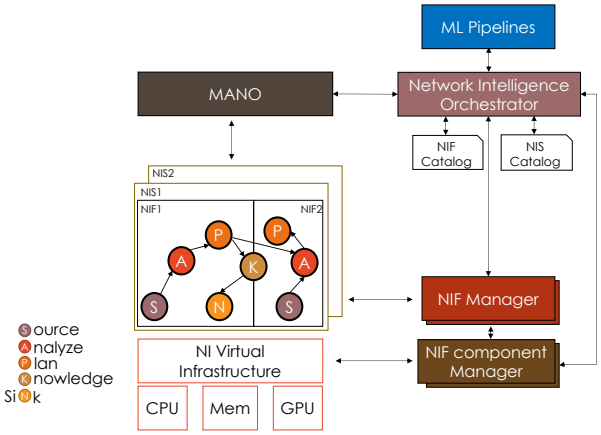

Figure 3: Architecture of the Network Intelligence Stratum.

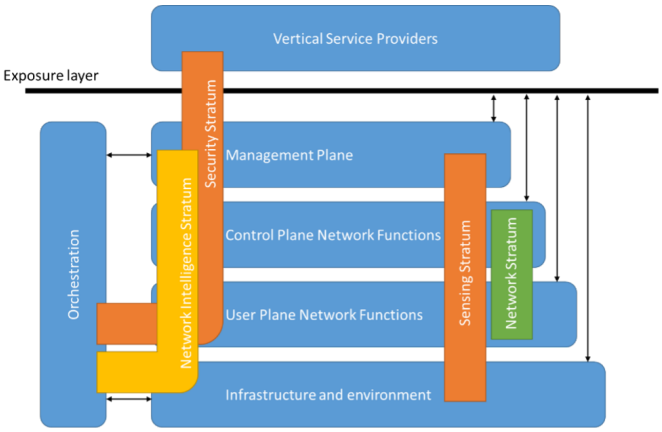

Figure 4: The 5GPPP Architectural WG framework [31].

endpoints must be configured. 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 Network Data Analytics Function (NWDAF) [29], 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 [30].

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 Orchestration: 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 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 interfaces with an external platform to construct ML pipelines and execute such operations, exemplified by an MLOps framework.

The proposed NI Stratum moves our previous network intelligence plane design [5] 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 4 and reported in the whitepaper [31], 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. 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 [7], 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 [32], [33, 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.

Conflict resolution capabilities are crucial for efficiently reusing and combining elements across NIFs to build NISs. By repre-

senting 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 accross domains: for instance, in Section 4.2 of [34], 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 [26] and MLflow [35]. However, in an NI-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 NI-Native architectures. In response to challenges in deploying multiple NIFs concurrently, the NIO incorporates several key functionalities, which are summarized in Figure 5 and listed as follows.

- *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 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 NFV Orchestrator (NFV-O), Virtual Network Function Manager (VNFM), and Virtual Infrastructure Manager (VIM) in frameworks like ETSI NFV MANO.

The architectural design presented in this section is complemented next in the following ways: (i) in Section 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 5, by designing the set of procedures that address the needs and challenges introduced in [7] and that motivate the functionalities mentioned above.

4. Network Intelligence Stratum Interfaces

As shown in Figure 5, the NI Stratum is a composition of different functional blocks that aims for the native integration of

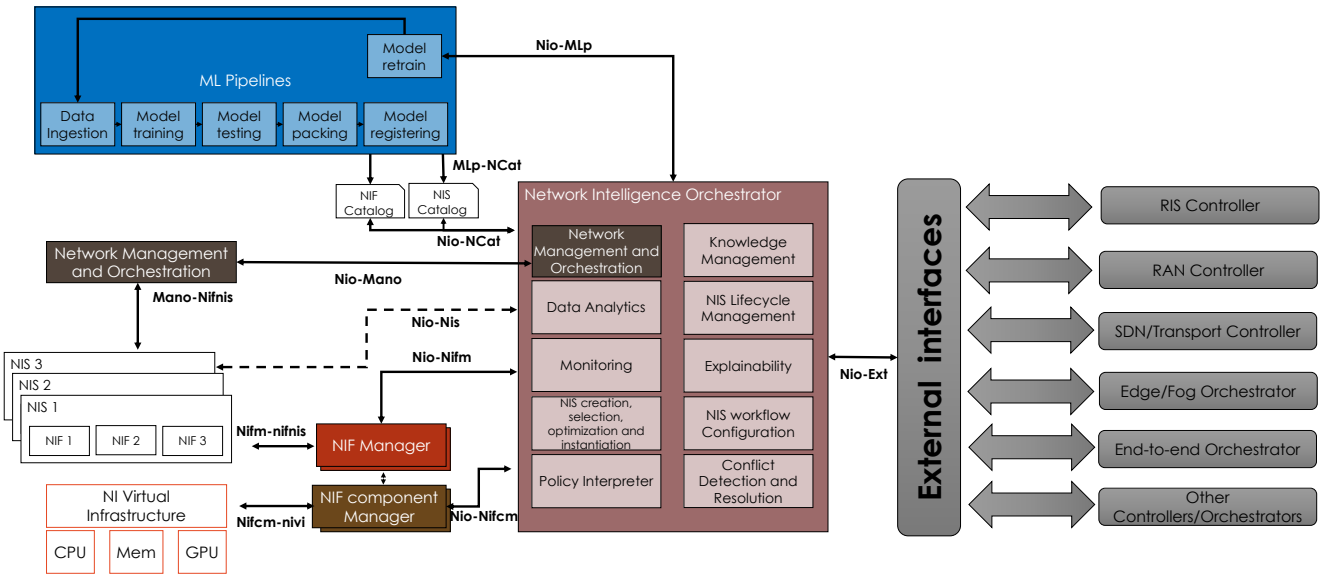

Figure 5: The NI Stratum and the functional blocks of the NIO and ML pipelines, with the internal and external interfaces of the Stratum.

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 [36] and O-RAN [37], 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 what is expected from them.

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 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. It promotes efficient utilization of network resources, optimized network service delivery, and enhanced scalability and flexibility of virtualized NIF. 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 monitoring 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.

- **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 (GPU, FPGA, 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 Service Level Agreements (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, 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. 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 advance 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 YAML and JSON where available (e.g., via ETSI or 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.

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.

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 [38], 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 [37]:

- **Offline Learning**: In O-RAN, even for Reinforcement Learning (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 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:

- **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 (Non-Real-Time Radio Intelligent Controller).
- **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).

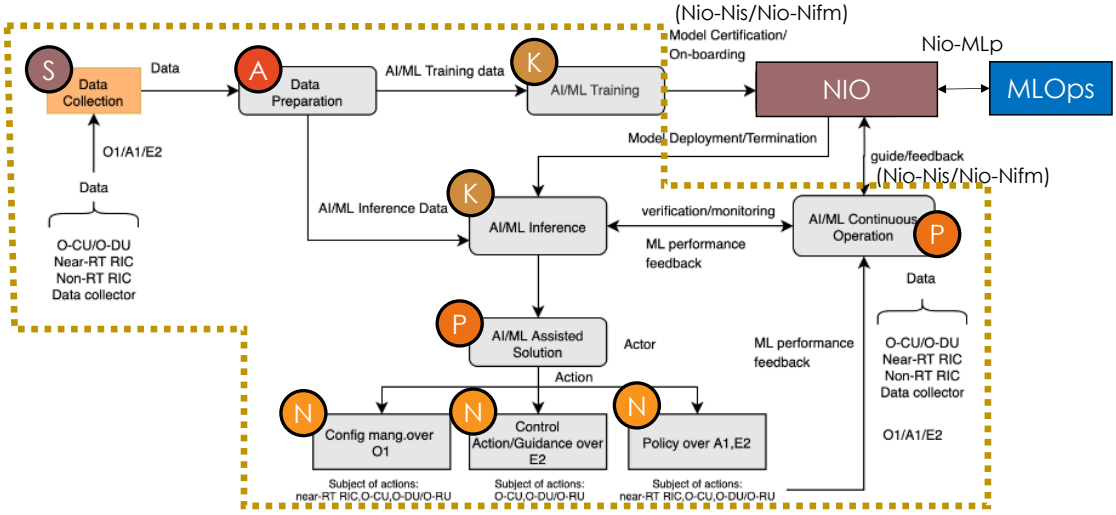

Figure 6: Integration of the NI Stratum and O-RAN AI/ML Lifecycle Procedures and Interface Frameworks

- **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.

4.2.2. 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 7 presents the interconnections among various components. The framework is divided into three domains and shows where the NI Stratum takes a role. The first domain, referred to as 5GC, is where the NWDAF resides. Within this domain, 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 domain encompasses Operations, Administration, and Maintenance (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 domain 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,

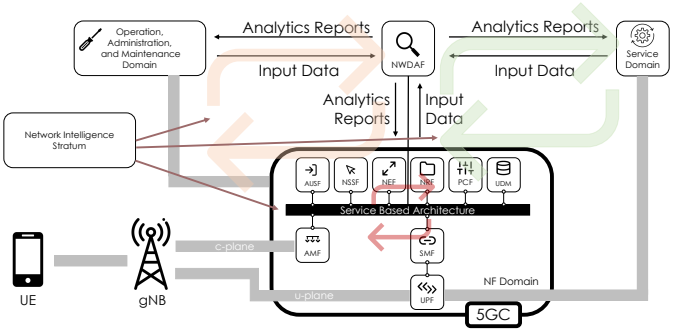

Figure 7: The architectural framework proposed by the 5GPPP Arch WG [31].

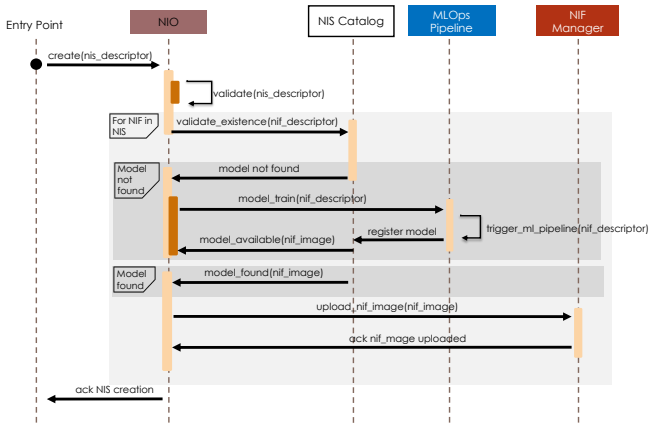

Figure 8: NIS creation process flow.

the MDAF is an 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 domain encompasses the service domain, 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 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 is 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.

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. 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.

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 [36] is the referent architectural framework to look up to. 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.

5.1.1. NI Creation

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 NIF Descriptor (NIFD)/ NIS 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 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.
- 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.

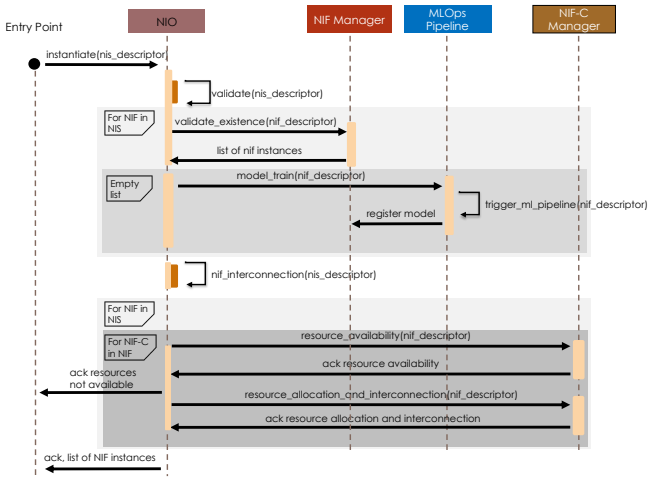

Figure 9: NIS instantiation process flow.

5.1.2. *Instantiation or Deployment*

Figure 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 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

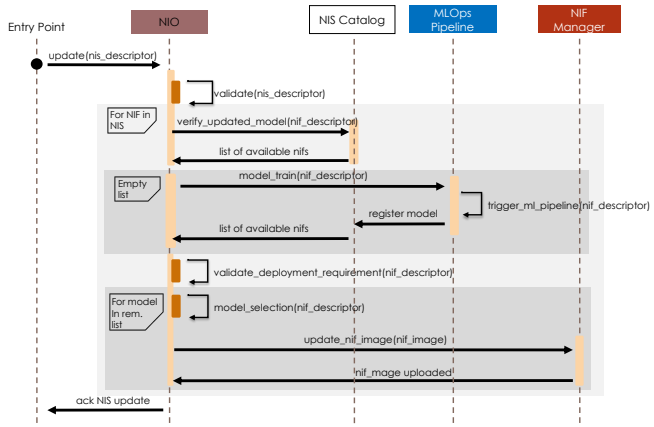

Figure 10: NIS update process flow.

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.

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 inherently 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. 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 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 [36], requiring an extra step to update the NIS/NIF, which was shown above.

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 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. 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.

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 interactions with any NI-related block and the MANO block can perform it. The implementation of such procedures is shown in [36].

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.

5.2.1. Conflict Resolution

We introduced two specific conflict cases in Section 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 that will 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 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

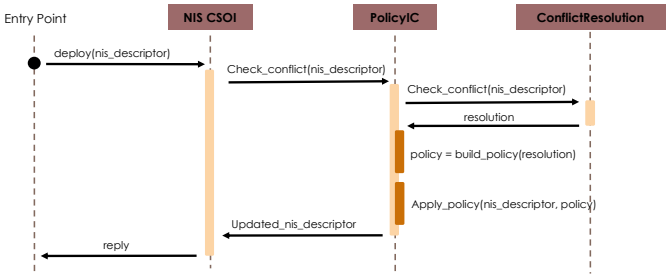

Figure 11: Conflict Resolution process flow.

different monitoring granularity in a Source shared with other NIS, or requires controlling a NF 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 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. 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 has any conflict with the existing NISs. 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). 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.

5.2.2. Knowledge Sharing

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 NISs, via the interaction with the Model Explainability block and derive new policies that represent the shared knowledge among NISs, by interacting with the PolicyIC block.

Figure 12 shows the main steps for the case of NIS Knowledge Sharing. This procedure includes checking the parame-

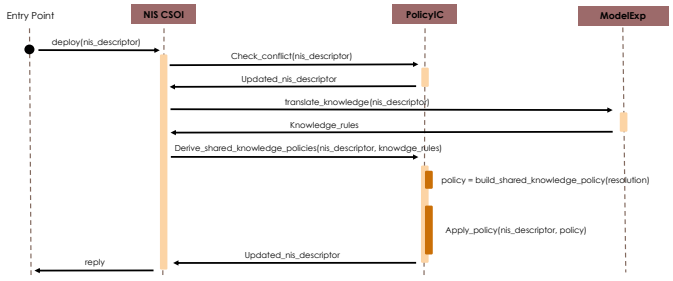

Figure 12: Knowledge Sharing process flow.

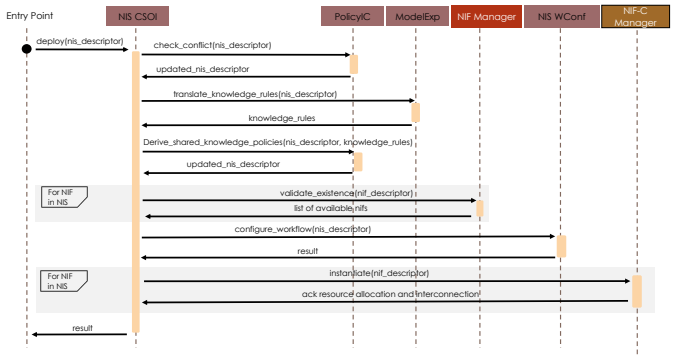

Figure 13: Intra NIO instantiation and deployment process flow.

ters 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 an external domain, the NIS CSOI will first translate such knowledge in the Model Explainability block before building and applying the shared knowledge policies:

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 in 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.

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 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 5.1 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 2 summarizes the functionalities proposed in Section 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. However, further procedures can be defined based on other use cases, e.g., orches-

Table 2: Summary of the procedures proposed to address the challenges described in Section 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

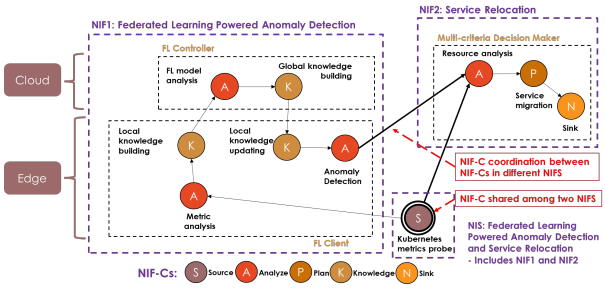

Figure 14: A federated learning-powered anomaly detection and service relocation NIS.

tration 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.

6. Network Intelligence Stratum Implementation

This section shows a reference implementation of the proposed NI Stratum. The first subsection shows how two NIFs can be described and orchestrated to form a NIS. Then, assuming that both NIFs are acting upon the same network element (i.e., reallocating and scaling the same network service), we show how selected functionalities of the NI Stratum can be implemented.

6.1. Combining NIFs to build a NIS

One of the key features of the NI Stratum is to allow the creation, management, and deployment of NISs. In this section, we showcase a key functionality of such Stratum: combining two NIFs to create a NIS. The first NIF utilizes a federated learning-powered anomaly detection algorithm, enabling anomaly detection at the edge (NIF1 in Figure 14). This means that the AI model for detecting anomalies is trained locally on individual devices, while a central controller is responsible for retaining a global model and, therefore, enhancing the local AI model. This way, data privacy is preserved while the approach still benefits from a collaborative learning process. The second NIF employs a service relocation algorithm that collects real-time monitoring information from the edge [39], such as CPU load, memory load, used storage, and end-to-end latency measured at the client side. Leveraging a multi-criteria decision-making function, this algorithm dynamically relocates services based on real-time data (NIF2 in Figure 14), ensuring optimal resource utilization and service performance. It is important to note that by service relocation, we mean relocation of stateless services, a procedure that deploys the correct service instance on the selected edge and terminates all previous service instances running on the other edges to save the edge resources.

Following the proposed framework based on the N-MAPE-K to define NIS, we effectively combined the federated learning-powered anomaly detection with service relocation capabilities to achieve the NIS. Figure 14 shows the N-MAPE-K based diagram of the resulting NIS. By integrating the federated learning approach, the NIS ensures that anomalies are detected securely and efficiently across the network's edge devices. The NIS also leverages the monitoring information collected from the edge to make informed decisions about service relocation, maximizing the network's overall performance and responsiveness.

Before implementation, the NIS needs to be created, which is a procedure that includes validating the existence of constituting NIFs, and if one of them is not found, the proper models are created and trained, resulting in NIF images that are necessary for NIS implementation (Figure 16). The communication among the NIS is implemented using the Eclipse Zenoh data communication framework [27], which provides a reliable and scalable solution for exchanging data between devices and components within the network. Additionally, the implementation utilizes Kubernetes [25] to realize the NIF Manager and the NIF-C Manager, as shown in Figure 15. Kubernetes helps man-

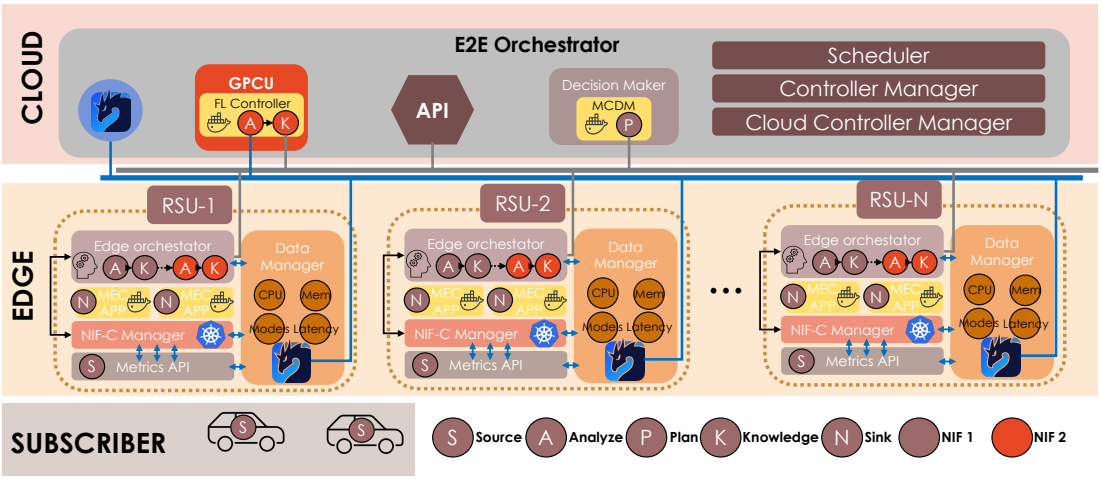

Figure 15: Cloud-to-edge deployment of the proposed NIS and its different components.

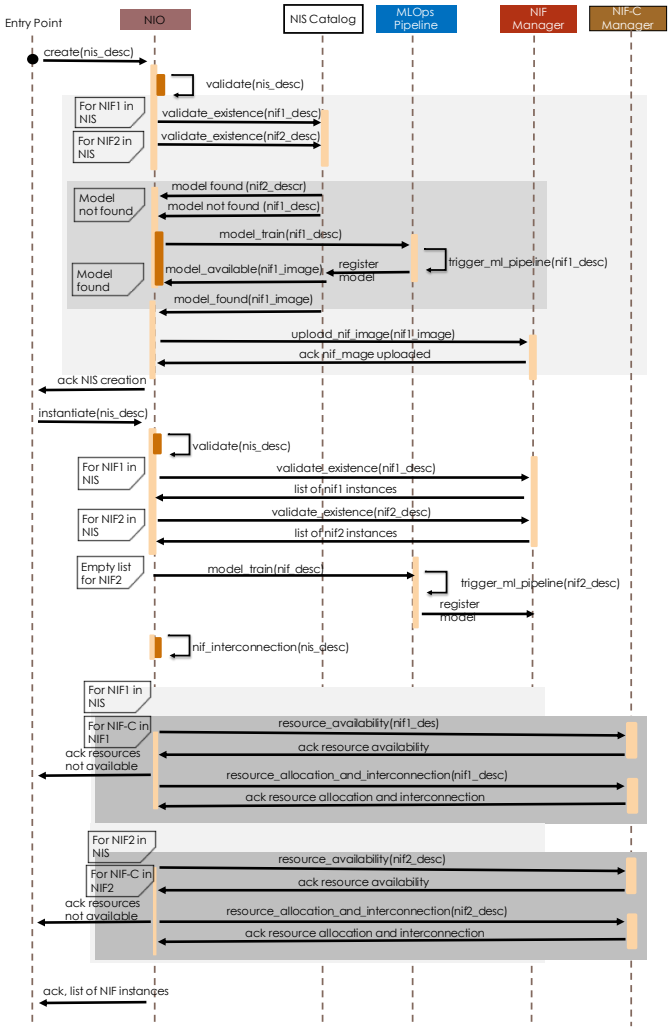

Figure 16: NI Procedures for the example NIS.

age the deployment, scaling, and orchestration of the NIF-Cs, ensuring smooth operation and efficient service relocation.

The Smart Highway testbed [40] located on top of the E313 highway in Belgium served as the edge environment for testing and validating the effectiveness of the proposed NIS, providing a real-world scenario to assess its performance and potential benefits for vehicular services. Figure 15 shows a high-level representation of the deployment and how the components were deployed at the cloud (centralized server) and edge (roadside units at the Smart Highway). As shown in Figure 15, Zenoh-based data managers are used for collecting metrics related to edge performance (CPU, memory, storage) and service performance (end-to-end latency). The edge performance metrics are used in the anomaly detection process (NIF1), while both edge metrics and service metrics are used in the service relocation process (NIF2). In the latter case, the service performance is measured as end-to-end latency collected from the client. The client is implemented on the test vehicle, which measures latency in communication with services deployed at the network edges in real-time. Once the final decision on the service relocation is made, i.e., when NIS decides to which network service the test vehicle should connect, the subscriber running in the vehicle receives all necessary information for connecting to the service.

The outcome of the deployment of this NIS is an edge-aware service relocation, which makes efficient decisions on when and where the service should be relocated to ensure minimum end-to-end latency at every moment, taking into account not only the edge performance but also its stability in terms of possible anomalies. Such intelligent services pave the way towards more reliable and high-performing deployments of vertical services in future 6G ecosystems. Nevertheless, the two NIFs presented in this section could produce conflicting decisions, which might severely affect the overall NIS performance. Thus, it is essential to ensure proper conflict resolution, as explained in Section 6.2.

6.2. *Managing a conflicting NIS*

The conflict resolution procedure is one of the NI Stratum procedures with paramount importance for the stability and smooth operation of the network. The architectural details of the pro-

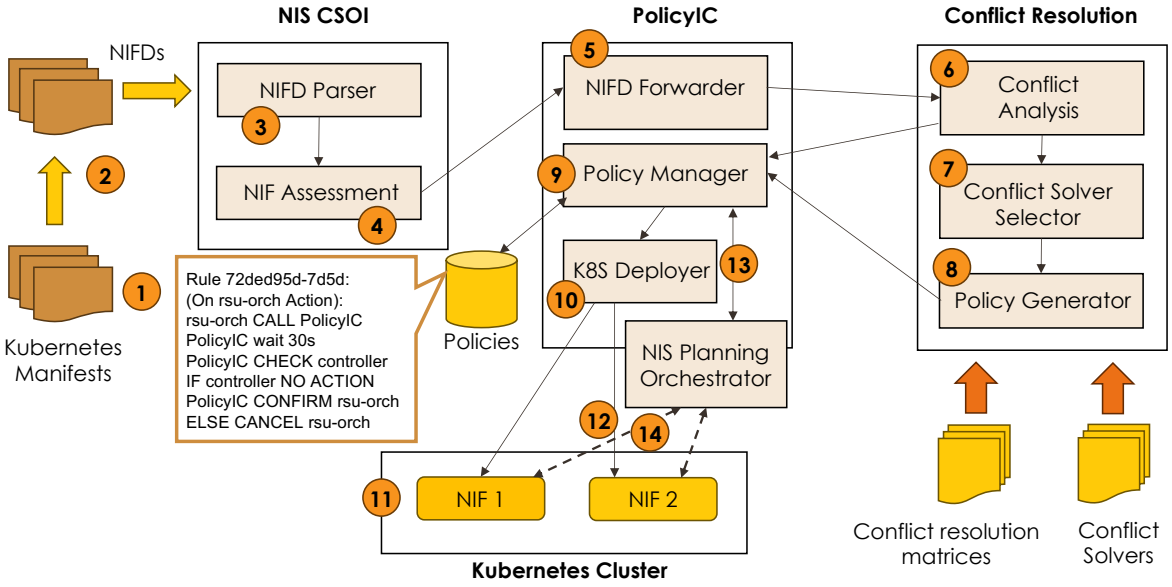

Figure 17: Conflict resolution process.

cedure are described in detail in Section 5.2.1, while in this section, we develop and demonstrate how to enforce this procedure when conflicts emerge between two different NIFs algorithms acting on the same network functions but configuring different values for the target parameters. During the prototyping, Kubernetes was used as the main deployment infrastructure, while the following phases of the conflict resolution process were addressed: (i) NIFD creation and annotation following the N-MAPE-K taxonomy; (ii) initial assessment based on N-MAPE-K types; (iii) conflict identification and; (iv) conflict resolution. The process is illustrated in Figure 17 and explained in the current section.

To demonstrate the conflict resolution capabilities of the NI Stratum, two NIFs were created and deployed. The first NIF includes a federated learning-powered anomaly detection algorithm (identical to the NIF1 presented in Section 6.1) extended with a basic service resource management capability (e.g., scale in/out on resource under/ overutilization). The second NIF remains identical to the NIF2, as explained in the previous section. The configuration actions of the above NIFs (scale in/out and service relocation) cause conflicts because they may act upon the same service, driving the network to unstable conditions if executed in a similar time window. Therefore, in the prototype, we showcase the identification of the conflict while appropriate policy rules are generated before deployment to avoid unstable conditions by orchestrating the configuration actions.

Initially, the NIFDs of both NIFs are created. Since each NIF-C is deployed as Pod in Kubernetes, we create the NIFDs directly from the Kubernetes manifest files. In this direction, the NIFDs are generated by extending such files in two ways: (i) labels are used to discriminate between different N-MAPE-K types (Sensor, Monitor, Analyse, Plan, Execute, Effector); (ii) annotations are used to specify the configuration capabilities of

```
template:
  metadata:
    annotations:
      kompose.cmd: kompose convert
      kompose.version: 1.28.0 (c4137012e)
      daemon.nmapek.plan.target.spec.containers.resources.requests.memory: "true"
      daemon.nmapek.plan.target.spec.containers.resources.limits.memory: "true"
      daemon.nmapek.plan.target.spec.containers.resources.requests.cpu: "true"
      daemon.nmapek.plan.target.spec.containers.resources.limits.cpu: "true"
    creationTimestamp: null
    labels:
      io.kompose.network/f1-dbscan-testing-default: "true"
      io.kompose.service: controller
      daemon.nmapek.type.knowledge: "true"
      daemon.nmapek.type.plan: "true"
  spec:
    containers:
      - env:
```

Figure 18: Descriptor of a NIF (NIFD) that can perform service scale out.

```
template:
  metadata:
    annotations:
      kompose.cmd: kompose convert
      kompose.version: 1.28.0 (c4137012e)
      daemon.nmapek.plan.target.spec.containers.nodeSelector: "true"
    creationTimestamp: null
    labels:
      io.kompose.network/f1-dbscan-testing-default: "true"
      io.kompose.service: rsu-orch
      daemon.nmapek.type.plan: "true"
  spec:
    containers:
      - env:
```

Figure 19: Descriptor of a NIF (NIFD) that can perform service relocation.

the NIF in a self-descriptive way. This process is illustrated in steps (1) and (2) in Figure 17.

The NIFDs of the two aforementioned NIFs are illustrated in Figure 18 and Figure 19 respectively. The first NIFD (Figure 18) is of type "Knowledge" due to the federated learning-powered anomaly detection capabilities and "Plan" since it can perform service scale out. In detail, the possible configuration targets follow the prefix "daemon.nmapek.plan.target" followed by the resource they can configure. This structure is already used in the Kubernetes manifest files (e.g., "spec. containers.resources.requests.memory"). In the same manner, Figure 19 illustrates the NIFD of the second NIF. This NIF is of type "Plan" since it can perform service relocation, affecting the "spec.containers.nodeSelector" part of a Kubernetes manifest file.

Then, the CSOI module is responsible for parsing the provided NIFDs to identify the N-MAPE-K types included in the NIF and the possible configuration actions that the NIFs can generate (step 3 in Figure 17). In addition, an initial assessment is performed to quickly identify if a conflict process is required (step 4). The CSOI communicates with the PolicyIC module, which forwards only the relevant parameters to the Conflict Resolution module. The Conflict Resolution module executes a thorough analysis to identify any possible conflict (step 6). The analysis is based on a set of conflict resolution matrices and the list of already deployed NISs.

In this direction, a conflict may be identified between an existing NIS and a new NIS to be deployed. In case of conflicts, the conflict solver selector is triggered. This module selects the appropriate Conflict Solver from a set of available Conflict Solvers (step 7). The Conflict Solver is a software component that can resolve a conflict during the deployment and operation phases. This means that it may update the existing capabilities of a NIS or generate Policies as a set of rules (step 8) to handle conflicting situations during the operation of conflicting NIFs. An example of a rule is also illustrated in Figure 17, where NIF2 will wait 30 seconds before relocating the service if NIF1 already scaled it. In the case of new Policies, the Policy Manager is responsible for adding them to the Policies database and becoming responsible for keeping them updated during the whole lifecycle. After all checks are made and any Policies are generated and activated, the new NIS is deployed by the K8S Deployer (steps 10 and 11).

During operation, all the certified NIF coexist conflict-free in the same Kubernetes Cluster (step 11). In the case of a conflicting NIF, the candidate conflicting NIF communicates with the PolicyIC (and NIS Planning Orchestrator) before any conflicting action is realized (step 12). Then, the NIS Planning Orchestrator is responsible for retrieving the related Policies from the Policy Manager and applying their rules. After the related Policies are executed, the PolicyIC may or may not give the green light to the NIF to complete the requested configuration (step 14).

7. Conclusion

This paper presents the architectural design of the NI Stratum, an evolution of a NI plane, 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 our previous work on defining NI, highlighting the importance of data and the mechanisms required for coordinating NI instances. We have extended this prior work by introducing 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 the paper is an important step forward in the field, addressing key challenges in NI coordination and conflict resolution.

The research presented in the manuscript not only extends theoretical and methodological aspects of NI but also focuses on practical applicability. The paper expands the scope of the reference implementation to include necessary modifications that support conflict resolution capabilities in leading open-source platforms such as Kubernetes, Kubeflow, and Zenoh. This expansion is crucial for broadening the potential impact and utility of the research, making it more relevant and applicable in real-world scenarios.

7.1. Open Challenges

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 [41]. Moreover, 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.

An ongoing challenge in the deployment of the NI Stratum involves the detection and resolution of conflicts, especially when multiple NI operate within different network domains. The need for effective policy interpretation, configuration, and enforcement to manage these conflicts is crucial. This requires a robust mechanism for interpreting NI decisions and outcomes. Besides improving trustworthiness in black-box NI models, Explainable AI (XAI) [42] can offer insights into the inner workings of NI models, thereby demystifying their outcomes. Moreover, as networks become increasingly autonomous, explaining decisions by the NI becomes imperative, especially in scenarios where the decisions may have critical consequences. XAI provides a framework for ensuring that AI-driven decisions within NI are fair, unbiased, and compliant with regulatory standards.

Although out of scope in this paper, developing accurate network digital twins is also a paramount direction 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, 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. [43]. 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.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 101017109 "DAEMON".

References

[1] Saad, W., Bennis, M. and Chen, M., A vision of 6g wireless systems: Applications, trends, technologies, and open research problems, IEEE Network 34 (3) (2020) 134–142. doi:10.1109/MNET.001.1900287.

[2] Camelo, M. et al, Requirements and Specifications for the Orchestration of Network Intelligence in 6G, in: 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), IEEE, 2022, pp. 1–9.

[3] ETSI, Zero-touch network and Service Management (ZSM): Means of Automation, Group report, ETSI (2020-05).

[4] Wang, Y. et al, From design to practice: ETSI ENI reference architecture and instantiation for network management and orchestration using artificial intelligence, IEEE Communications Standards Magazine 4 (3) (2020) 38–45.

[5] Camelo, M. et al, DAEMON: A Network Intelligence Plane for 6G Networks, in: 2022 IEEE Globecom Workshops (GC Wkshps), IEEE, 2022, pp. 1341–1346.

[6] Gramaglia, M. et al, Network intelligence for virtualized ran orchestration: The daemon approach, in: 2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), IEEE, 2022, pp. 482–487.

[7] Chatzieleftheriou, L.E. et al, Orchestration Procedures for the Network Intelligence Stratum in 6G Networks, in: 2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), IEEE, 2023, pp. 347–352.

[8] h2020, D., Daemon nip presented live at eucnc 2023 (Sep. 2023).
URL https://www.youtube.com/watch?v=-7qOyYUBKf0

[9] Manias, D.M., Chouman, A. and Shami, A., Model drift in dynamic networks, IEEE Communications Magazine (2023).

[10] Bassoli, R. et al, Deliverable D5.2: Analysis of 6G architectural enablers' applicability and initial technological solutions, accessed: 2024-06-19 (Oct. 2022).
URL https://hexa-x.eu/deliverables/

[11] Khorsandi, B.M. et al, Deliverable D1.4: Hexa-X architecture for B5G/6G networks – final release, accessed: 2024-06-19 (Jul. 2023).
URL https://hexa-x.eu/deliverables/

[12] Akgul, O. et al, Deliverable D3.3 Initial analysis of architectural enablers and framework, accessed: 2024-06-19 (Apr. 2024).
URL https://hexa-x-ii.eu/results/

[13] Stańczak, S., Utkovski, Z. et al, Toward 6G: Key Directions and Research Questions, accessed: 2024-06-19 (2022).
URL https://6g-ric.de/6g-ric/#position-paper

[14] Taleb, T. et al, White Paper on 6G Networking. 6G Research Visions, accessed: 2024-06-19 (2020).
URL http://urn.fi/ urn:isbn:9789526226842

[15] Yates, R.D. et al, Age of information: An introduction and survey, IEEE Journal on Selected Areas in Communications 39 (5) (2021) 1183–1210.

[16] Ahmad, R. et al, Zero-day attack detection: a systematic literature review, Artificial Intelligence Review 56 (10) (2023) 10733–10811.

[17] Benzaïd, C. and Taleb, T., Ai for beyond 5g networks: A cyber-security defense or offense enabler?, IEEE network 34 (6) (2020) 140–147.

[18] Naeem, F. et al, Security and privacy for reconfigurable intelligent surface in 6g: A review of prospective applications and challenges, IEEE Open Journal of the Communications Society (2023).

[19] Paez, I. et al, DAEMON Deliverable 2.1: Initial report on requirements analysis and state-of-the-art frameworks and toolsets (Jun. 2021). doi:10.5281/zenodo.5060979.
URL https://doi.org/10.5281/zenodo.5060979

[20] Iovene, M. et al, Defining AI native: A key enabler for advanced intelligent telecom networks, Tech. Rep. BCSS-23:000056 Uen, Ericsson (Feb. 2023).

[21] Li, P., Xing, Y. and Li, W., Distributed AI-native Architecture for 6G Networks, in: 2022 International Conference on Information Processing and Network Provisioning (ICIPNP), IEEE, 2022, pp. 57–62.

[22] Rossi, D. and Zhang, L., Network artificial intelligence, fast and slow, in: Proceedings of the 1st International Workshop on Native Network Intelligence, 2022, pp. 14–20.

[23] Brito, F. 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), IEEE, 2023, pp. 98–102.

[24] D'Oro, S. et al, OrchestRAN: Orchestrating Network Intelligence in the Open RAN, IEEE Transactions on Mobile Computing (2023).

[25] Kubernetes, accessed: 2024-01-08 (2024).
URL https://kubernetes.io/

[26] Kubeflow, accessed: 2024-01-08 (2024).
URL https://www.kubeflow.org/

[27] Eclipse Zenoh, accessed: 2024-01-08 (2024).
URL https://zenoh.io/

[28] IBM, An architectural blueprint for autonomic computing, White paper, IBM (Jun, 2006).

[29] 3GPP, Architecture enhancements for 5G System (5GS) to support network data analytics services, Technical Specification (TS) 23.288, 3rd Generation Partnership Project (3GPP), version 17.3.0 (December 2021).
URL https://www.3gpp.org/DynaReport/23288.htm

[30] Garcia-Saavedra, A. and Costa-Perez, X., O-RAN: Disrupting the virtualized RAN ecosystem, IEEE Communications Standards Magazine 5 (4) (2021) 96–103.

[31] Bahare, M.K. et al, The 6G Architecture Landscape - European perspective (feb 2023). doi:10.5281/zenodo.7313232.

[32] Garcia-Aviles, G. et al, Nuberu: Reliable RAN virtualization in shared platforms, in: Proceedings of the 27th Annual International Conference on Mobile Computing and Networking, 2021, pp. 749–761.

[33] Garcia-Saavedra, A. et al, DAEMON Deliverable 3.2: Refined design of real- time control and VNF intelligence mechanisms (Nov. 2022). doi:10.5281/zenodo.7525876.
URL https://doi.org/10.5281/zenodo.7525876

[34] Fuentes, L. et al, DAEMON Deliverable 4.2: Refined design of intelligent orchestration and management mechanisms (Jan. 2023). doi:10.5281/zenodo.7544155.
URL https://doi.org/10.5281/zenodo.7544155

[35] MLFlow, accessed: 2024-01-29 (2024).
URL https://mlflow.org/

[36] ETSI, Network Functions Virtualisation (NFV); Management and Orchestration, Specification, ETSI (2014).
URL https://www.etsi.org

[37] O-RAN Alliance, O-RAN Working Group 2 AI/ML workflow description and requirements, Technical report, O-RAN Alliance (Oct, 2020).

[38] Polese, M. et al, Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges, IEEE Communications Surveys & Tutorials (2023).

[39] Slamnik-Krieštorac, N. et al, An ml-driven framework for edge orchestration in a vehicular nfv mano environment, in: 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), IEEE, 2023, pp. 728–733.
[40] Smart Highway Testbed, accessed: 2024-01-08 (2024).
URL https://www.fed4fire.eu/testbeds/smart-highway/
[41] Zeydan, E. and Mangues-Bafalluy, J., Recent advances in data engineering for networking, IEEE Access 10 (2022) 34449–34496.
[42] Brik, B. et al, A survey on explainable ai for 6g o-ran: Architecture, use cases, challenges and research directions, arXiv preprint arXiv:2307.00319 (2023).
[43] Almasan, P. et al, Network digital twin: Context, enabling technologies, and opportunities, IEEE Communications Magazine 60 (11) (2022) 22–27.