University of Calabria

Department of Computer Engineering, Modeling,
Electronics and Systems - DIMES

Doctoral Programme in Information and Communication Technologies - XXXVII Edition

Doctoral Dissertation

In-Network Computing and Net4AI Paradigms

For Future 6G Programmable Networks

Supervisor
Prof. Antonio Iera
University of Calabria

Candidate
Dr. Mattia Giovanni Spina

Co-supervisor
Prof. Floriano De Rango
University of Calabria

Candidate ID
237453

Supervisor of the Doctoral Programme
Prof. Giancarlo Fortino
University of Calabria

"This thesis is dedicated to my family, whose presence and support have been fundamental throughout this long journey. Their strength, wise advice, and unconditional love have helped me overcome every challenge. Without them, none of this would have been possible.

I would also like to express my deepest gratitude to my supervisors and Mauro, whose guidance and invaluable insights have been essential to the completion of this work. Their expertise, encouragement, and constant support have been crucial in shaping both my research and my personal growth throughout my PhD journey."

"The irrationality of a thing is not an argument against its existence, rather a condition of it."
— Friedrich Nietzsche

Abstract

The evolution toward 6G networks presents unprecedented opportunities for integrating Artificial Intelligence (AI) within network architectures, transforming traditional communication systems into intelligent, adaptable infrastructures. This thesis explores two key research directions that advance this vision: *in-network security* and *network support to distributed AI*.

The first line of research addresses the need for secure-by-design principles by proposing a distributed, in-network security framework. Leveraging programmable networking devices, this framework embeds anomaly detection models directly within the data plane, creating a pervasive security fabric capable of identifying and mitigating malicious activities in real-time. This approach emphasizes the critical role of distributed intelligence in achieving robust, resilient networks.

The second line of research utilizes In-Network Computing (INC) and Split-AI techniques to distribute AI-relevant computational tasks across various network elements. By dynamically offloading parts of complex AI models, e.g. Neural Network (NN), to computation-enabled user plane entities, this approach reduces latency, conserves energy, and minimizes processing demands on User Equipment (UE). Through the development of an Intelligent User Plane (IUP), this thesis demonstrates how next-generation networks can natively support distributed AI applications, enabling real-time, resource-efficient processing closer to end-users.

Together, these contributions underscore the potential of AI-enhanced 6G networks to deliver secure, scalable, and intelligent infrastructure. By addressing critical requirements for both security and AI support, this work provides a comprehensive foundation for the design and deployment of resilient, AI-native network systems in the 6G era.

Contents

List of Figures

List of Tables

List of Acronyms

AF Application Function

AI Artificial Intelligence

AI4Net AI for Networks

AN Access Node

APSPC All-Pairs Shortest Path Coloring

ASIC Application-Specific Integrated Circuit

BRKGA Biased Random-Key Genetic Algorithm

CCCE Communication and Compute Control Entity

CC Flow Communication and Compute Flow

CP Control Plane

CS Compute Service

CUPE Computation-enabled User Plane Entity

DL Deep Learning

DNN Deep Neural Network

DPPL Data Plane Programming Language

DT Decision Tree

EC Edge Computing

FPGA Field Programmable Gate Array

HLS High-Level Synthesis

IDS Intrusion Detection System

INC In-Network Computing

INCaS-AI **INC-a**ssisted **S**plit-**AI**

IUP Intelligent User Plane

IoT Internet of Things

M/A Match&Action

ML Machine Learning

Net4AI Network for AI

NBI SouthBound Interface

NIC Network Interface Card

NIDS Network Intrusion Detection System

NF Network Function

NN Neural Network

NPU Network Processing Unit

OF OpenFlow

P4 Programming Protocol-independent Packet Processors

PDP Programmable Data Plane

PDU Packet Data Unit

PISA Protocol Independent Switch Architecture

RF Random Forest

RMT Reconfigurable Match Tables

SBI SouthBound Interface

SDN Software-Defined Networking

SL Strong Learner

SMF Session Management Function

TNA Tofino Native Architecture

UE User Equipment

UP User Plane

UPF User Plane Function

URLLC Ultra-Reliable Low-Latency Communication

VNF Virtual Network Function

WL Weak Learner

Chapter 1 Introduction

The rapid advancement of communication technologies has created unprecedented demands on network infrastructures. As global data traffic surges and applications become increasingly data-intensive and latency-sensitive, traditional network architectures are being challenged to meet these evolving requirements. This pressure has exposed a fundamental issue known as *network ossification*, wherein the rigid and inflexible nature of legacy networks limits their ability to adapt to new services and innovations. To address this, the development and adoption of programmable network paradigms have accelerated, particularly SDN and Programmable Data Plane (PDP), which together offer unprecedented levels of flexibility, control, and adaptability within the network.

SDN introduces a paradigm shift by decoupling network control from the underlying hardware, allowing centralized and programmable management of network resources. This decoupling not only simplifies network management but also enables dynamic and context-aware reconfiguration of network functions in response to varying traffic demands. Complementing this, programmable data planes empower network devices to be more than mere data forwarding elements; they can now perform complex packet processing operations in real-time, directly within the network path. This capability facilitates on-demand, customizable processing at the data plane level, moving towards an agile network capable of supporting the sophisticated demands of modern applications.

An emerging paradigm that capitalizes on these capabilities is In Network Computing (INC). By embedding computational resources directly within network devices, INC allows for data processing to occur "on the fly" as it traverses the network, minimizing latency and optimizing resource utilization. This concept is particularly relevant in the context of AI and Machine Learning (ML), where data processing and analysis are often distributed across multiple nodes. Processing data closer to

its source within the network enables timely responses and reduces the bottleneck created by centralized processing hubs, a crucial consideration for AI applications that demand low-latency and high-throughput environments.

Two closely related paradigms have emerged around the intersection of AI and networking: AI4Net and Net4AI. AI4Net refers to the use of AI techniques to enhance network management and operations. Through applications such as traffic prediction, anomaly detection, resource optimization, and fault management, AI4Net has significantly advanced network automation, enabling more efficient and resilient infrastructures. In essence, AI4Net focuses on using AI to optimize the network itself, a widely explored and impactful approach that has transformed operational efficiency and adaptability across many networked systems.

In contrast, Net4AI represents a more recent evolution, enabled by the increasing computational and programmability capabilities within network devices. Rather than applying AI to improve network functions, Net4AI envisions the network as a platform that actively supports and enhances AI workloads. This paradigm shift is driven by the notion of embedding AI capabilities within the network itself, transforming it from a mere data transport medium into a fully integrated AI-native platform. By utilizing in-network computing, programmable data planes, and advanced data handling mechanisms, Net4AI facilitates distributed AI workloads with minimized latency and seamless data flows, empowering a new generation of AI-driven applications. This concept has become essential in the context of emerging 6G networks, where the seamless support of intelligent applications – from autonomous vehicles to immersive virtual environments – is anticipated as a core functionality.

Building on these advancements in networking, the thesis aims to explore the potential of AI for Network (AI4Net) and Network for AI (Net4AI) paradigms as foundational elements for future 6G networks. By investigating these two complementary approaches, this work seeks to delineate their unique characteristics, capabilities, and synergies, contributing to a deeper understanding of how AI-driven techniques can reshape network functionality and performance. In this direction, this thesis will be organized into two main research lines.

Firstly, to address the concept of built-in security and avoid the security challenges that plagued previous generations of networks, this research delves into the design of a distributed, pervasive, in-network defense mechanism. The proposed approach involves an In-Network Distribution of Learning Functions for establish-

ing a secure-by-design programmable data plane within next-generation networks [5, 6]. Leveraging the programmability of modern networking devices, pre-trained ML models are directly embedded within these devices to perform anomaly detection, identifying malicious traffic patterns as they cross the network.

Current research literature witnesses the effort of addressing this particular task. However, the effort is mainly focused on finding the best way to encode complex AI models in order to fit networking devices and meet the limitations of the existing PDP programming language (e.g. Programming Protocol-independent Packet Processors (P4)). Moved by the conviction that addressing the problem by solely focusing on an "embedding"-relevant issue is limited, a change of perspective is proposed in order to address it by leveraging the network itself. More specifically, the proposed approach relies on the joint combination of distributed and split AI techniques, INC, and Virtual Network Function (VNF) chaining to decompose complex AI models into their core functional blocks, which are lighter and more manageable. In this way, by enabling a cooperative approach among the AI-augmented PDPs that execute these function blocks, the computational and memory demands on individual devices are reduced, allowing them to preserve their primary function of packet forwarding while simultaneously enhancing security. To achieve this vision a networking architecture is designed to support this paradigm, and it consists of three main planes: the **Artificial Intelligence Plane (AIP)**, **the Control & Orchestration Plane (C&OP)**, and the **AI-enhanced Programmable Data Plane (AIePDP)**.

In this architecture, the network itself becomes the first line of defense against existing and future threats to network communications. By deploying a cooperative network-wide approach of AI-enhanced programmable devices, it is naturally established an *in-network security fabric* – a pervasive, adaptive system capable of detecting malicious behavior as part of the network's core functions. This secure-by-design approach thus equips next-generation networks with a robust, scalable defense mechanism embedded within their foundational infrastructure.

The second line of research presented in this thesis focuses on integrating INC with Split-AI in the architecture of 6G networks, elaborating on the concept of an IUP introduced and presented in [2, 4] to support AI-driven applications. INC enables computational tasks to be processed directly within network elements, such as Access Nodes (ANs) and User Plane Functions (UPFs), reducing latency and

energy consumption by allowing data to be processed in transit. As a progression from traditional mobile edge computing, the INC-enabled IUP facilitates task offloading to these enhanced user plane entities within 6G systems, particularly benefiting latency-sensitive applications such as augmented reality (AR). This approach allows complex processing tasks to occur closer to the user, thereby reducing the demands on end devices and improving response times. To achieve this, the architecture introduces CUPEs, which augment traditional user plane devices with advanced processing capabilities for AI tasks, thus marking a shift from the purely communication-focused design of prior network generations.

The research further develops the Net4AI paradigm, which leverages INC to support complex AI workflows by distributing NN layers across various network nodes. In this context, the concept of **INC-a**ssisted **S**plit-**AI** (INCaS-AI) [2, 4] is explored, where portions of a neural network are dynamically allocated across CUPEs, enabling distributed AI computation at multiple points within the IUP. This distributed approach is adaptive and resource-efficient, reducing latency, conserving energy, and optimizing resource allocation for AI applications that demand high responsiveness and efficiency. Critical requirements for implementing Split-AI in this manner include intelligent task distribution, dynamic network configuration, and sophisticated signaling mechanisms to ensure optimal placement and reallocation of NN layers as network conditions evolve. In this direction, the main challenges as well as the main functional and structural modifications to both the Control Plane and the User Plane of the next generation of networks are thoroughly explored in order to natively accommodate such AI-relevant tasks directly in the network. This is crucial for the envisioned tight integration between AI and networking as a main pillar for the upcoming 6G Networks.

Building on these foundational elements, the research presents a proof-of-concept model that validates the performance of the INCaS-AI approach within a simulated 6G environment [3]. This model examines the impact of different NN partitioning strategies on key performance indicators (KPIs), including inference latency, network traffic load, and energy consumption for both UE and network nodes. By executing NN layers across multiple CUPEs, the study demonstrates that INCaS-AI can significantly enhance the efficiency and responsiveness of 6G networks, advancing the vision of a fully integrated AI-network ecosystem. Additionally, the obtained results shed light on the importance of distributing and offloading part of

the computational-relevant tasks on networking devices for the UE. By offloading substantial computational demands from user equipment to network devices, INCaS-AI also simplifies UE design, reducing device complexity, energy consumption, and the need for extensive on-device processing power—an essential advantage for lightweight, battery-dependent devices. In a few words, this research establishes INCaS-AI as a crucial enabler of 6G networks, providing a scalable and intelligent infrastructure capable of supporting distributed AI applications, reducing UE complexity, and realizing the full potential of next-generation networks. This research has been conducted in the context of the *one6G project* [7] to support the project's vision of an IUP enhanced with INC capabilities to natively accommodate for AI-relevant workflow [2, 4, 3].

In essence, this thesis is aimed at addressing two pivotal aspects of next-generation 6G networks: secure-by-design principles through a distributed in-network security fabric, and resource-efficient AI integration via INC-assisted Split-AI. By embedding intelligence within network elements for both security and computation, this work contributes to the foundation of a robust, adaptable 6G infrastructure, capable of supporting secure, intelligent, and scalable digital ecosystems. The remainder of the thesis is organized as follows: a brief introduction about the main concepts behind programmable networks, INC, Net4AI, and AI4Net is given in Chapter 2; the first line of research that elaborates on the in-network security fabric is described in Chapter 3; while, the study on the structural and logical modifications envisioned for the upcoming 6G architecture to seamlessly integrate AI-relevant workflows is provided in Chapter 4. Finally, Chapter 5 concludes the thesis offering a comprehensive summary of the thesis's primary contributions and drawing considerations about potential future research directions concerning programmable networks in 6G.

Chapter 2 Programmable and Intelligence-enabled Networks

2.1 From Traditional to Programmable Networks

The transition from traditional to programmable networks represents a significant leap in the evolution of network technology, fundamentally altering how networks are designed, managed, and optimized. Traditional networks, which served as the foundation for many technological innovations, (Figure 2.1) are however typically characterized by rigid architectures, where network devices such as routers and switches are configured manually and operate with fixed, hardware-based functionalities. These systems, often vendor-specific, present challenges in terms of scalability, flexibility, and the ability to rapidly adapt to new demands or threats. As a result, traditional networks struggle to meet the dynamic requirements of modern applications and services.

In response to these limitations, programmable networks have emerged as a transformative solution, enabling a more flexible and responsive network architecture. At the core of this shift is the concept of *SDN* (see Figure 2.2), which decouples the control plane from the data plane [8]. This decoupling allows network intelligence and routing policies to be applied through a logically centralized controller, managed in a software-based manner. The network elements that form the data plane—such as switches and routers—are thus simplified into programmable packet forwarding devices, controllable through open interfaces like OpenFlow (OF) [9].

The evolution from SDN to *PDP* (Figure 2.3). marks a further enhancement

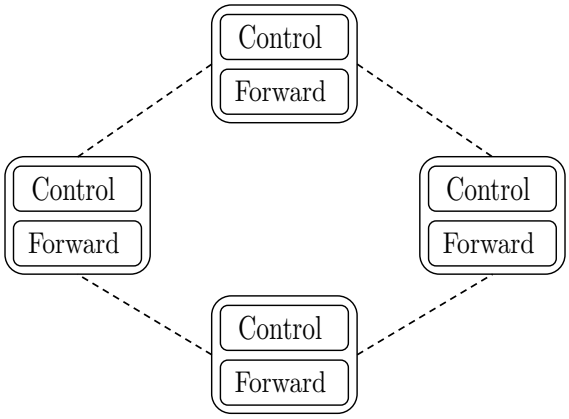


Figure 2.1: Traditional Networks

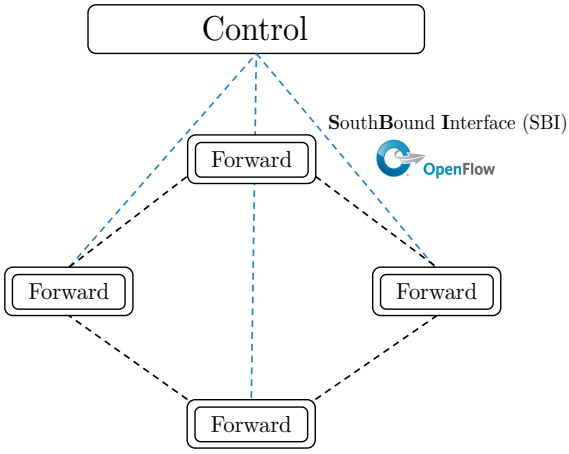


Figure 2.2: Programmable Networks: Programmable Control Plane (SDN)

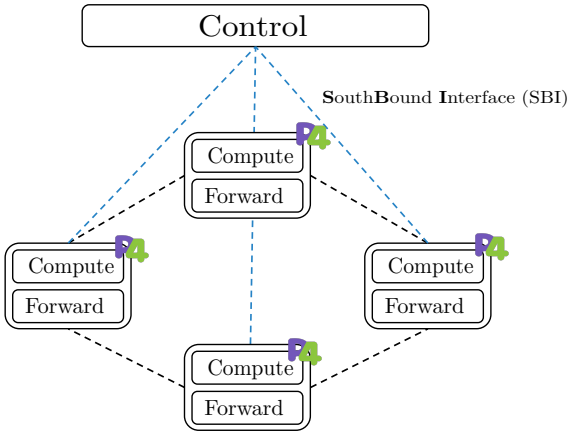


Figure 2.3: Programmable Networks: Programmable Control & Data Plane.

in network programmability. Unlike traditional networking devices, where packet processing functions are hard-coded into the hardware, programmable data planes allow these functions to be defined and modified through high-level programming languages, like High-Level Synthesis (HLS) and P4. This flexibility enables network operators to rapidly deploy new data plane functionalities and application-specific logics, such as network traffic classifier, data aggregator, and In-band Network Telemetry, facilitating the prototyping and adoption of innovative network services without the need for costly redesigns of switch hardware, such as Application-Specific Integrated Circuits (ASICs).

One of the most promising developments within programmable networks is the emergence of *INC*. This paradigm leverages programmable data plane technology to enable network devices, to perform computational – on the network path – tasks traditionally handled by central and far servers. INC transforms network elements from mere conduits of data into active participants in the computational process, thereby creating a more secure, power-efficient, and stable computation fabric with high processing capacity. This approach not only enhances the performance and efficiency of network operations but also complements existing computational paradigms by distributing processing workloads closer to the data source. The integration of computation within the network fabric is particularly advantageous in scenarios requiring real-time data processing, such as Internet of Things (IoT), Edge Computing (EC), and AI applications. By processing data as it traverses the network, in-network computing reduces latency, lowers bandwidth consumption, and

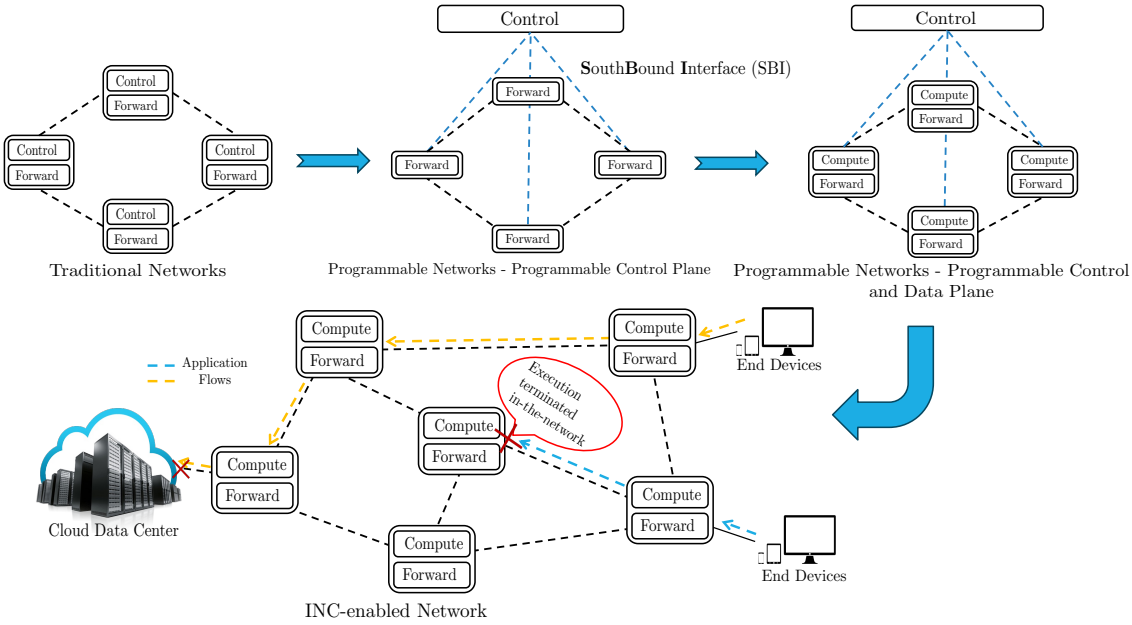


Figure 2.4: From Traditional to Programmable Networks

improves the overall efficiency of the network. Furthermore, the ability to program network elements to handle specific tasks on the fly provides network operators with unparalleled flexibility in traffic engineering and the development of new protocols and applications. Figure 2.4 shows the evolution from the traditional to the full-programmable networks, including networks enhanced by INC. The transition from traditional to programmable networks, therefore, not only addresses the limitations of conventional network architectures but also opens up new possibilities for innovation and optimization. As networks continue to evolve, the principles of programmability and in-network computing will play a crucial role in shaping the future of networked systems, enabling them to meet the demands of increasingly complex and dynamic computational environments.

2.2 Software-Defined Networking (SDN): The Revolutionary Shift to Programmable Networks

Traditional networks have played a pivotal role in transforming and revolutionizing the world. However, the proliferation of connected devices, the rising demand for cloud services, and the substantial volume of data generated by the widespread deployment of Massive IoT have exposed the limitations of conventional networks. These traditional systems are increasingly inadequate in delivering the necessary flexibility and scalability required to address the diverse and time-sensitive demands of contemporary and future applications. Their fixed-functions vendor-specific hardware has posed an insurmountable barrier to evolution and research innovation for many years, causing the commonly known *Network Ossification* problem, which made the operation of service additions hard and hindered from ease and real-time network upgrades [10]. To overcome these issues, for the last two decades, networks have undergone a profound evolutionary process. The first important revolutionary step, that introduces the concept of programmable networks, is SDN.

Although its widespread introduction and adoption can be traced to the period between 2010 and 2015 – mainly due to the pioneering work of Scott Shenker on programmable networks – SDN has its roots in the concept of *Active Networks*, which in the 1990s and 2000s provided an initial glimpse of the programmable network concept [11, 12]. Active Networking proposed a paradigm in which network devices are capable of reacting based on the content carried within packets. Instead of merely handling raw data, each packet contains a program. When a network device receives a packet, it executes the program contained within, performing actions such as forwarding or dropping the packet based on the data plane design. Despite the concept of the active network being more relevant to INC and PDP, it laid the foundations for programmable networks to be conceived. Indeed, differently from it, the SDN is more focused on control plane programmability rather than data plane. The core idea of the SDN paradigm resides on the concept of *decoupling the control plane from the data (or forwarding) plane*, making the control plane programmable as software. To achieve this objective, a new networking architecture has been introduced. It consists of three layers:

- *Application/Management Plane*: it consists of a set of applications focusing on network devices that communicate with the control plane issuing directives to install custom behavior (e.g. application-specific routing and forwarding policies) on data plane devices.
- *Control Plane*: it decouples the control and the forwarding logic of forwarding devices. The control plane becomes programmable, allowing customized networking management and routing solutions to be implemented in the data plane. This allows for flexible and adaptable network management, with design solutions that can be tailored to applications.
- *Data Plane*: it is the forwarding operation plane. Forwarding devices, that reside in this plane, receive and parse packets, match key fields against tables of rules, and transmit packets to the next hop.

The layers of the SDN architecture, then, are capable of interacting by means of two interfaces: *SouthBound Interface (SBI)* and *SouthBound Interface (NBI)*. To implement the SBI, there exists a standardized protocol that was introduced by Shenker et. all [9] in 2008, and that gave the first glimpse into the concept of a programmable network. his protocol, known as OpenFlow (OF), has since become widely adopted. Of course, other protocols can be used to implement the SBI, among which the widely adopted alternative to OF is NETCONF. Although a standardized protocol exists for the SBI, no formal standards have been established for the NBI, which serves as a communication interface between the diverse application landscape and the SDN controller. The three-layer architecture that characterizes SDN is shown in Figure 2.5. In addition, if more than one SDN controller controls the network, the architecture provides two additional interfaces: East-and-WestBound interfaces. These two interfaces allow communication between the set of SDN controllers that reside in the control plane. It is quite interesting to notice that, as their names imply, the interfaces in SDN architecture are named after the cardinal directions—North, South, East, and West—based on the communication flow between the various entities within the architecture It appears clear that, in such a described setting, the networking devices are turned into simple forwarding devices, that manage network packets on the basis of the rules installed by the Control Plane through the use of the SBI interface. More specifically, the applications guide the behavior of the Control Plane by means of high-level behavior description by means of the NBI interface. This high-level description is then encoded, and

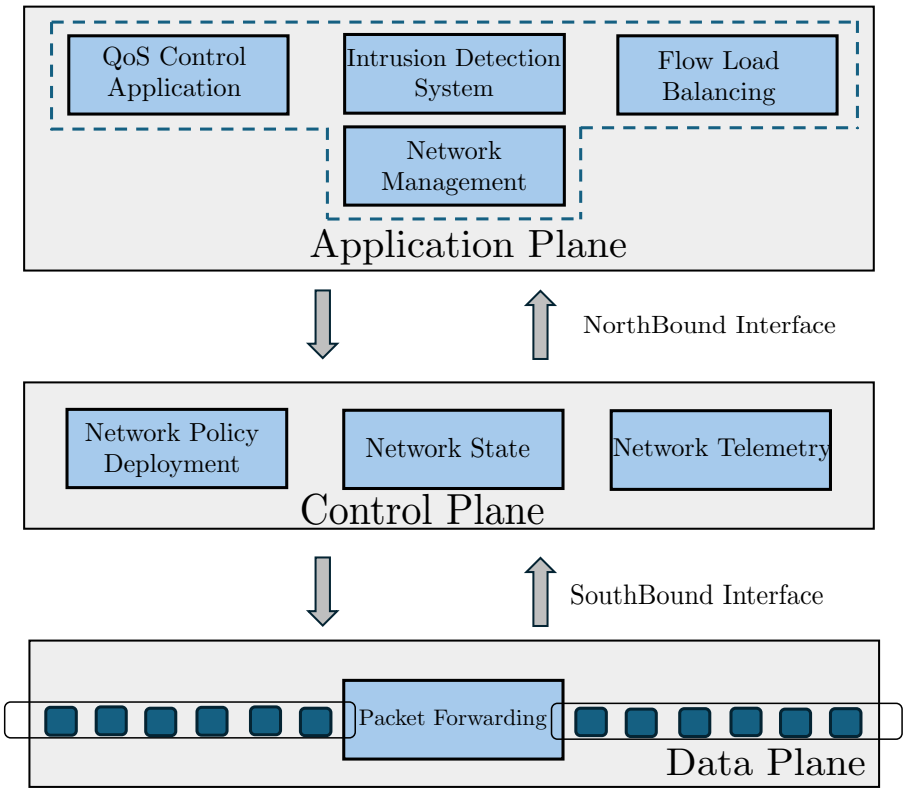


Figure 2.5: Software-Defined Networking (SDN) Architecture

here it comes to play the SBI, in rules (flow rules if OF is used) that are installed on the networking devices allowing them to actuate that particular behavior.

Although SDN offers significant advantages and features that serve as the foundation for network programmability, it also presents certain drawbacks that create challenges in meeting the stringent requirements of modern AI-relevant applications, particularly regarding latency and real-time decision-making processes. In order to program data plane devices, a *closed-loop* between the application plane, control plane, and data plane needs to be created [13]. Applications, on the basis of the network information gathered by network devices, determine the relevant forwarding policies. These forwarding policies are communicated – via NBI – to the SDN controller which, conversely, translates them into forwarding rules that it installs on the data plane devices by means of the SBI. This closed control loop results in three main critical issues:

1. High Inter-Layer Communication Latency: the interaction between any two

layers imposes additional long delays. This is not suitable for critical delay-sensitive applications (e.g. Ultra-Reliable Low-Latency Communication applications that require 1ms latency).

2. Real-Time Adaptation and Decision-making: data plane devices are only capable of actuating forwarding policies previously installed by the SDN controller. Quickly adapting to network changes and making decisions in a real-time manner are challenging, since the SDN controller must intervene in each decision. The time taken by the SDN controller to gather new network topology information, evaluate it, and compute and install the new forwarding policies introduces severe delay in network management.

3. Coarse Network Status: networking devices can provide basic network status information like throughput, bytes, and number of managed packets. It is not possible to instruct networking devices to compute more complex or composite metrics, such as queue length of interfaces or specific flow rate.

To overcome these issues allowing a more customizable network management and pursuing the stringent requirements of the real-time and critical applications, PDP has been introduced. It also extends the programmability feature to the data plane devices, making the whole network a fully programmable object [13].

2.3 In-Network Computing (INC): let the network compute

PDP, as a recent advancement in networking, promises unprecedented flexibility and innovation, emerging as a key enabler for In-Network Computing. The INC paradigm, indeed, is strongly based on the idea of leveraging networking elements, such as programmable switches, Field Programmable Gate Arrays (FPGAs), and smart Network Interface Cards (NICs), as a general purpose and programmable computing entity to carry out application-specific tasks. A standard and commonly accepted definition of INC does not exist. Through the years, researchers, striving to characterize the features of such a paradigm, provided different definitions [14]. A definition from ACM is given as follows:

Definition 1 (ACM [15])**.** In-Network Computing refers to the execution of programs typically running on end hosts within network elements. It focuses on com-

puting in the network, using devices that already exist within the network and are already used to forward the traffic.

Another definition is given by one of the earliest contributions regarding INC [16] and it is summarized in the following:

Definition 2 (Sapio et al. [16])**.** In-Network Computing is offloading a set of compute operations from end hosts into network elements such as switches and smart NICs.

In addition, Ports and Nelson envisioned the idea of making the network a computing entity, providing their definition of INC in [17] as:

Definition 3 (Ports and Nelson [17])**.** Application-specific functions that can run in programmable network hardware at line rate, offering orders of magnitude higher throughput and lower latency than can be achieved by a traditional server.

From the above definitions, it is possible to draw the main features that characterize the INC paradigm:

- Networking elements as an active participant to the computation which derives from the offloading of part of an application-specific task or process.
- To support general-purpose computation, networking devices can now be programmed.
- The traditional tasks of the network (e.g. routing and forwarding) remain unchanged and are still conceived as the default procedure of the network nodes.
- Enabling application-specific tasks to be completed *in-the-network*, terminating the relevant request on the network path and avoiding delays resulting from reaching remote servers or clouds.

Considering its breakthrough characteristics, INC application spans in various areas and research fields including network security, data caching, and aggregation, network traffic classification as well as distributed intelligent systems. Literature witnesses the pervasive adoption of *INC* as a crucial means to build future generation networks. For instance, in [18] PDP devices are used to detect and mitigate on the network path a SYN Flooding attack. ASIC programmable switches are

used in NetCache [19], in which processes like indexing, caching, and serving of hot key-value items is performed within the data plane devices. This solution allows for not only an increasing service throughput but also a reduced query response latency. Since networking devices are primarily involved in handling packets, it appears to be useful to leverage INC to classify network packets as they traverse PDP devices. In this direction, an exemplary solution is given in [20] in which a design of ML models for PDP is proposed allowing networking devices to classify network traffic. With P4xos [21], PDPs implement a consensus protocol reducing the latency imposed by the communication needed to reach a distributed consensus. The mentioned literature works enumerate just a few of the existing solutions enabled by INC but they serve as a valuable demonstration of how INC is poised to revolutionize not only the design but also the management of the upcoming future generations of networks.

2.4 Programmable Data Plane: The step To Complete Network Programmability

The rise and widespread adoption of the INC paradigm is primarily driven by significant efforts put in extending the "programmable" capability beyond the control plane (as in SDN) also to the data plane. In SDN networking devices are programmed by means of OF rules. However, OF does not allow general-purpose programming for PDP devices. The introduction of the Protocol Independent Switch Architecture (PISA) architecture, a generalization of the Reconfigurable Match Tables (RMT) model, opened the door to the capability to fully customize the packet processing process without the need for hardware reconfiguration overcoming the vendor-specific and fixed sets protocols [22]. This advancement was also crucial in overcoming the critical challenge of *Network Ossification*, a problem that hinders the ability of network infrastructures to keep pace with the rapid innovations in general-purpose computing. The PISA architecture, depicted in Figure 2.6, is composed of three building blocks [1]:

- Programmable Parser: it allows users to define standard and custom protocol headers. The parser is implemented as a finite state machine through which protocol headers (e.g. Ethernet, IP, VLAN, TCP/UDP, as well as

custom-defined protocol headers) are sequentially extracted from the incoming packets.

- Programmable Match&Action Pipeline: The data extracted from the packet header through the Parsers, is processed by a sequence of Match&Action (M/A) tables. M/A tables constitute the basic and fundamental units enabling the lookup of a key value – derived from the packet header (e.g. the IPv4 source IP address) – in a table and mapping the resulting entry to a specific action which operations can (or not) affect the processed packet.
- Programmable Deparser: after being processed, packet headers must be reconstructed. After the Parser reconstructs the headers and their fields, the packet is emitted and transmitted to the network.

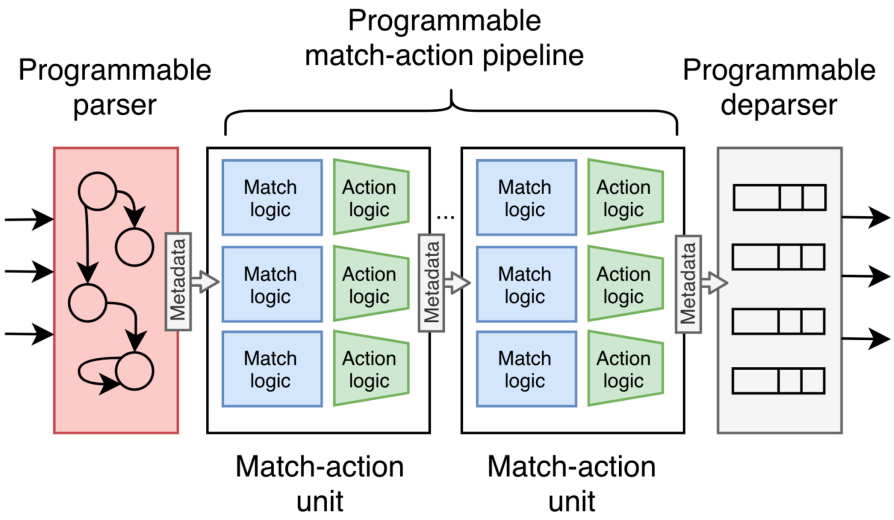


Figure 2.6: Protocol-Independent Switch Architecture [1].

The PISA architectural model can be considered as a general abstraction of the programmable packet processing pipeline of a PDP device. In order to program each of the mentioned building blocks of the packet processing pipeline, a Data Plane Programming Language (DPPL) is needed. The P4 language is a widely adopted domain-specific language designed to customize the programmable elements of the packet processing pipeline, enabling switches to implement user- or application-specific logic. P4 was introduced in 2015 and it is currently maintained by the P4 Language Consortium [23]. The latest version of the language is the $P4_{16}$ (2023) and is described in the standard specification document provided by the P4consortium

[24]. The language requires a compiler, which starting from a P4 source file generates a binary file loaded on the PDP device. More specifically, the compiler translates P4 programs into a target-specific configuration binary code, which can be executed on a device known as a *P4 target*. Each P4-programmable networking device (e.g. smartNIC, switch) is defined as P4 target. Each target vendor provides its implementation of the PISA architectural model along with the specific compiler. In such a way, building on top of the PISA several architectures have been designed, which are called P4 architectural model: v1Model which has been designed for the virtual implementation of a target PDP devices, namely Behavioral Model version 2 (BMv2) [25]; the XSA architecture for Xilinx FPGA-based smartNIC [26] P4 target as well as the Tofino Native Architecture (TNA) for Intel-based Tofino programmable switch [27]. Therefore, both the P4 architectural model and compiler are provided by the manufacturers (or vendors) of the reference P4 target. Building on them, the user designs the P4 program that describes the custom behavior that the target has to implement. It is worth noticing that manufacturers must also provide data plane API that permits the control plane to manage the PDP devices behavior at run-time. The overall deployment process of a P4 program is summarized in Figure 2.7.

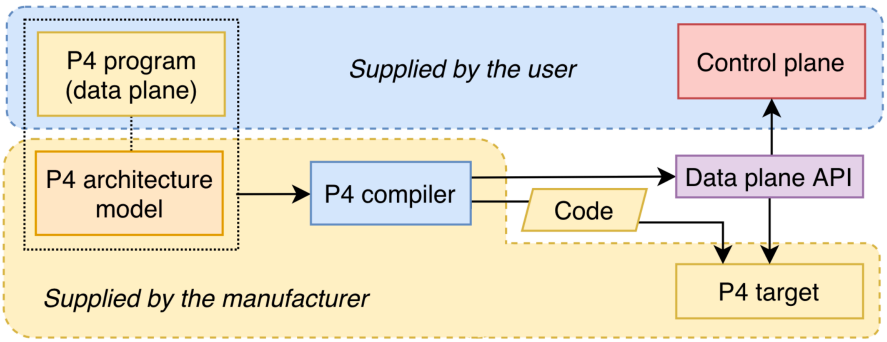


Figure 2.7: Deployment process of a P4 program [1].

One of the key advantages of P4 lies in its flexibility and ease of redeployment. Once a P4 program is loaded onto the target, it can be updated in real-time via the control plane. This capability allows for dynamic behavioral modifications and timely, real-time maintenance of individual or multiple P4 targets simultaneously. As a result, the time spent on manual configuration is significantly reduced, and the downtime of networking devices during reconfiguration is minimized.

2.5 Network For AI (Net4AI) and AI For Network (AI4Net)

In recent years, the rapid advancement and widespread adoption of AI have profoundly transformed various fields, including networking. As AI technologies have evolved, they have been increasingly utilized to enhance and support a wide range of networking activities. From automating network management tasks to optimizing traffic flows and improving security, AI has played a crucial role in addressing the growing complexity of modern networks. This integration of AI into networking has led to more intelligent, adaptive, and efficient networks, capable of managing the vast amounts of data and dynamic environments that characterize today's digital landscape. This trend, often referred to as AI4Net highlights the growing reliance on AI-driven tools to optimize the performance and reliability of modern networking infrastructures. Figure 2.8 depicts the networking architecture that embeds the AI into the network by introducing the so-called *Knowledge Plane*. The knowledge plane exploits its holistic view of the network to observe its overall evolution applying global effectiveness algorithms to optimize routing and traffic management. AI-relevant applications (e.g. ML and Deep Learning (DL)) reside in this plane with the aim of processing network information and generating custom and intelligent forwarding policies. In this setting, the Control Plane is only responsible for collecting data, and translating the new forwarding policies into actions to be distributed to the data plane. Through advanced ML models and real-time data analytics, AI can automate complex tasks such as network configuration, fault detection, predictive maintenance, and traffic engineering. For example, AI algorithms can dynamically route traffic, avoiding congestion and minimizing latency, while self-healing mechanisms can detect and resolve network issues before they impact service availability. These capabilities not only enhance network performance but also reduce the operational costs associated with manual network management. Additionally, AI4Net has enabled more robust network security, with AI-driven systems that continuously monitor network traffic to detect anomalies and potential threats in real-time. However, as AI applications become more data-intensive and latency-sensitive, the relationship between AI and networks has deepened further, giving rise to the complementary concept of Net4AI. While AI for networking fo-

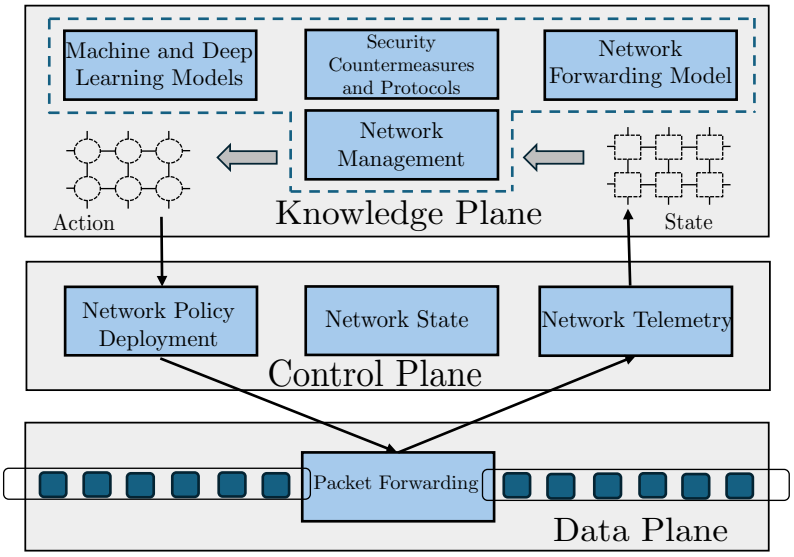


Figure 2.8: AI4Net Architecture Design.

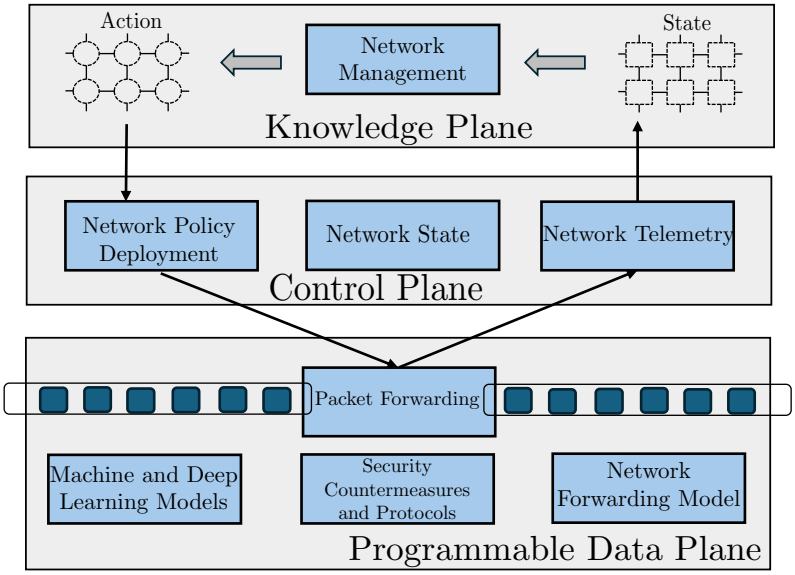


Figure 2.9: Net4AI Architecture Design.

cuses on improving network functionality through AI techniques, Net4AI shifts the focus to the network itself, ensuring it can meet the demanding requirements of AI workloads [28]. The Net4AI paradigm – primarily enabled thanks to the introduction of PDPs and leveraging the INC paradigm – involves designing and optimizing network infrastructures As AI models grow larger and more complex, particularly

in fields such as ML, DL, and natural language processing, the data processing and bandwidth demands they place on networks are enormous. Networks must be able to handle vast amounts of data in real-time, with minimal latency, to ensure the smooth execution of AI-driven tasks. This includes tasks such as training AI models across distributed data centers, managing large-scale ML operations, and supporting AI inference at the network edge for applications like autonomous vehicles, smart cities, and industrial IoT. Net4AI requires advanced networking techniques, such as INC and PDP, to distribute computational workloads closer to where data is generated. By doing so, AI workloads can be processed more efficiently, reducing the latency and bandwidth consumption associated with sending data to centralized cloud servers. High-performance networks that support AI need to ensure not only speed and *scalability* but also *reliability* and *security*, especially as AI applications become increasingly critical in sectors like healthcare, finance, and transportation. specifically to support the computational and data transfer needs of AI applications. Figure 2.9 depicts the structural changes applied to the AI4Net architecture shown in Figure 2.8. In essence, Net4AI ensures that the infrastructure can keep up with the computational and communication demands of modern AI systems. The ability to seamlessly handle the massive data flow generated by AI workloads, provide *real-time responsiveness*, and *scale* across distributed environments is key to unlocking the full potential of AI technologies. Together, AI4Net and Net4AI represent a mutually reinforcing cycle where AI enhances networking capabilities, and next-generation networks (e.g. the upcoming 6G network) enable the deployment and advancement of AI-driven innovations. This evolving synergy between AI and networking is set to shape the future of both fields, driving unprecedented advancements in how networks are designed, managed, and leveraged for AI-driven applications.

These paradigms serve as foundational elements for the next generation of networks, specifically 6G, where the seamless and profound integration of AI and networking will be a central feature and objective. As shifting towards 6G, the convergence of AI4Net and Net4AI will drive the development of networks that not only leverage AI to enhance network performance and management but also optimize network infrastructure to support the advanced requirements of AI applications. This dual focus will enable 6G networks to deliver unprecedented levels of efficiency, flexibility, and responsiveness, meeting the sophisticated demands of

emerging technologies and ensuring that both AI and network capabilities evolve in tandem to meet the needs of future digital ecosystems.

Chapter 3 Distributed and Ubiquitous Active In-Network Defense

3.1 AI4Net: The design of a Distributed and Ubiquitous In-Network Intrusion Detection System

One of the key aspects characterizing the transition from previous generation systems to 5G and to the imminent 6G is the ability to manage complex data flows of various kinds, originating from a multitude of devices and objects augmented with communication capabilities. This brings with it a growing threat represented by the exponential increase and diversification of network access points, which implies a more extensive attack surface. Future networks will therefore be increasingly exposed to external attacks, which in turn are becoming more and more sophisticated [29], making the need to design equally sophisticated network protection mechanisms extremely pressing. Among these are Network Intrusion Detection System (NIDS), which can be classified as passive, when they detect and record malicious behavior, or active, when they also take actions such as dropping packets or blocking Source IP when an attack is detected. NIDS can become more efficient by exploiting emerging paradigms offered by future 5G/6G networks, including programmability of data planes and related enabling languages (as discussed in Chapter 2). Another feature to be exploited is the one that will perhaps differentiate 6G networks the most from their predecessors, namely the massive distribution of intelligence across their data plane components for effective network management. Enabled by the general-purpose computing capabilities now embedded in network devices—no longer limited to simple packet forwarding—researchers have

been able to deploy AI models directly within these components. This problem, in current literature, is mainly addressed by focusing on finding the best encoding method of high-level ML/DL models in order to fit PDP computational and memory limitations. One of the key motivations of this thesis arises from the recognition that this challenge can be effectively addressed through an alternative approach that shifts the perspective by leveraging the synergy between distributed AI, split-AI techniques, and INC. Here, split-AI refers to the decomposition of a complex AI model into core functional partitions of reduced complexity. Deploying these partitions across network devices alleviates computational and memory demands—significantly lower than executing the full model—by taking advantage of the streamlined complexity of each partition. Nevertheless, this approach poses an additional challenge. To reconstruct the final output, computed in a distributed and disaggregated manner across the network, it is essential to establish a cooperative framework among AI-enhanced PDPs. Leveraging these concepts and in the wake of the mentioned innovations, the goal is to significantly contribute to the advancement of the state of the art of next-generation network intrusion detection techniques by proposing a disruptive solution that jointly exploits the concepts of Distributed and Split AI, INC, and PDP. Specifically, the main contribution does not simply consist in equipping elements of PDPs of future networks with pre-trained ML models (Decision Tree (DT), Random Forest (RF), etc.), to enable them to classify flows and packets *while crossing the network*; this is a topic widely covered by valid studies already available. Rather, the focus is centered on the evaluation of the potential of splitting a stronger ML model into weaker models distributed across groups of multiple PDP devices and exploiting their coordinated action to implement a fully distributed and cooperative mechanism to identify potential attacks and block them directly in-network from the first signs of their presence. This will thus prevent the malicious behavior from having adverse effects on the remaining portion of the network it would have traversed and on the target victim. Compared to what is available from the state of the art on the subject, the disruptive approach proposed in this thesis consists of making the network data plane itself capable of countering network intrusion attempts not only by classifying the traffic but also by applying countermeasures directly in-network. In this way, the network itself becomes an integrated Anomaly-based (ML-enabled) active IDS capable of adapting to changes in both traffic and topology and, therefore, much

more effective than single devices usually located at the network edges. In such a way, the aim is to design an *Ubiquitous and Pervasive In-Network Defense Fabric*. Even the alternative in-network solution that concentrates strong learners in single nodes shows the main weakness of losing effectiveness due to the excessively heavy load that it would add to the data plane devices in addition to that of the classic functions performed (routing & forwarding). Instead, the proposed solution based on in-network chains of highly distributed weak (and lightweight) learners can be very accurate and timely in countering attacks and can reduce the additional load on network traffic. The main goals of the new Intrusion Detection System (IDS) paradigm are summarized below:

- Block network attacks (timely and accurately).
- Minimize the time to detect and apply countermeasures.
- Maximize network coverage in terms of security by dynamically building minimum sets of network switches on which to dynamically deploy ML models.
- Minimize the impact of IDS features on the performance of network devices and the QoS of traffic passing through them.

3.2 Related Works

3.2.1 In-Network Security: ML/DL-aided Traffic Classification

With the advent of PDP and INC capabilities, recent efforts have focused on the design of in-network IDS solutions (also referred to as in-network classifiers) to address security-related challenges. A significant area of research investigated the use of the programmable PISA switch architecture by means of RMT, enabled by the introduction of the P4 language [30]. In [31] the authors proposed N2Net, a solution that implements the forwarding pass of a Binary Neural Network (BNN) in a P4-enabled switch, outlining the limitations of modern programmable networking devices in accommodating complex ML/DL models characterized by intricate computations and mathematical operations. Following this direction, the authors

of BaNaNa Split [32] extended the use of the BNN to SmartNICs to overcome the mentioned limitations: the joint work of programmable networking devices and end-host applications. Nevertheless, the proposed solution does not fit well the concept of ubiquitous and pervasive in-network security, since it does not work without a server that shares the workload with the networking device. With Taurus [33] and Homunculus [34], Swamy et al. proposed to equip the programmable networking devices with dedicated hardware capable of supporting map-reduce abstraction to perform complex mathematical operations. The main challenge of this approach is the need to redesign networking devices with custom and expensive hardware to enable them to perform ML/DL-relevant tasks. Parallel efforts have focused on encoding ML models within programmable networking devices, particularly Random Forests (RFs) and Decision Trees (DTs). In this direction, SwitchTree [35] and Forest [36] stand out as the most valuable examples. Both proposals strove to find the best encoding methodology to embed DTs and RFs within constrained and instruction set-limited PDPs. Following this trend, the works in [37, 38, 39, 40] show effort in designing a framework capable of encoding general RF/DT within P4-enabled networking devices. Recent research has demonstrated the remarkable capabilities of eBPF (extended Berkeley Packet Filter), showing nearly equivalent performance to P4 in managing general-purpose tasks offloaded to networking devices [41]. An important contribution in this domain is found in [42], where the authors focus on developing an efficient and effective encoding of a DNN using eBPF technology. A common effort emerging from the literature is the search for optimal encodings of the entire (sometimes complex) ML/DL models to adapt them to network devices with reduced impact on packet forwarding performance. None of them addresses how to intelligently distribute in-network classification modules to achieve pervasive and ubiquitous security through a fully distributed and collaborative approach to such modules, which is the objective of the novel paradigm studied and proposed in this thesis.

3.2.2 In-Network AI Function Distribution

The paradigm of the distribution of computational functions relevant to AI (both training and inference) finds its first evidence in the context of Edge and Cloud Computing. Many works in the literature addressed the concept of decomposing

a deep neural network (DNN) into its layers to distribute the workload between an edge mobile device and the cloud, proposing optimization models for this purpose. Among others, in [43] the best split is determined via regression models that predict the computational and energy consumption of each DNN layer, while in [44] the optimal solution is determined by considering device and network resource utilization to minimize end-to-end latency between the edge and the cloud. Only recently, with the emergence of the potential of the in-network computing paradigm, attention has shifted towards a distribution of learning functions that also exploits the network segments that connect Edge and Cloud. Understanding the close and crucial integration between artificial intelligence and future 6G networks, the authors of [4], [3] and [2] envisaged and analyzed the structural changes needed for the future 6G networks to naturally accommodate distributed artificial intelligence activities within their Data Plane.

Instead, Saquetti et. al. [45] focus on the constrained nature of PDP devices as well as the limitations imposed by the reference PDP programming language (i.e. P4) when dealing with distributed intelligence in the network. Through a simple PoC – a neural network with 3 layers and a total of seven neurons – they proposed an optimization model to distribute the DNN within the network at single neuron granularity, with a one-to-one mapping between PDP and neuron. However, it turns out that this type of distribution is not feasible when the neural network is complex, severely limiting the applicability of the proposal.

In the wake of the recent effort to deploy intelligence "in-the-network" by leveraging key enablers envisioned for upcoming 6G networks, the research conducted in this thesis aims to help fill a crucial literature and structural gap regarding network security for future generation networks. The close integration between AI and networks is a key factor for pursuing the concept of *integrated security*. By deploying virtualized anomaly detection and response functions (VADFs) across the network and enabling their collaborative action a security fabric can be created that makes the network the first line of defense against malicious attempts. The research conducted is deeply rooted in the understanding that this approach is essential to avoid the mistakes made with previous generations of networks, in which security was not designed in perfect synergy with the network itself but was treated as an "additional" functionality, thus opening the door to more intelligent and malicious attacks. The potential of the approach described is accompanied by new challenges,

such as finding the optimal positioning within the network of the AI-empowered security capabilities mentioned above to minimize both the delay in completing tasks and the resource consumption of the network devices involved. The research conducted in this thesis aims precisely to contribute to finding a solution to this compelling research problem.

3.3 In-Network Security Fabric: Proposed Network Architecture

This section provides details on the key components and reference functional architecture of a Ubiquitous In-Network NIDS that leverages a PDP enabled to support in-network distributed intelligence. Fig. 3.1 depicts the main framework components with relevant interactions. More specifically, the framework is logically organized into three planes, **A**rtificial **I**ntelligence **P**lane (*AIP*), **C**ontrol & **O**rchestration **P**lane (*C&OP*), **AI-e**nhanced **P**rogrammable **D**ata **P**lane (*AIePDP*).

The main tasks associated with each Plane are briefly detailed below to give the reader a glimpse of the proposal. In addition, in the following sections, each of the main functional blocks of the proposed framework is deeply detailed.

3.3.1 Artificial Intelligence Plane (AIP)

The AIP is responsible for ML/DL management-related operations. Specifically, it encompasses the environment devoted to model training, decomposition, and mapping onto Virtual Network Functions (VNFs). The first component, *Model Training*, handles the training of a model and performs all the tasks necessary to achieve this goal. After a good-quality set of data is obtained from the previous task, the next task is *model-building*, which consists in designing, testing, and evaluating different DL/ML ensemble models by feeding them with the previously generated dataset. Considering different types of models (e.g., RFs, DTs etc.), a *Model Fine-Tuning* step is performed by extensively searching within the hyperparameters space to find the parameter values of a model best suiting the context domain coded within the dataset. Since the focus of the proposed split-AI strategy is the Ensemble Learning approach, the output of this task is a Strong Learner (SL) or a set of SLs that are then processed by the next component, namely *Model Decomposition*,

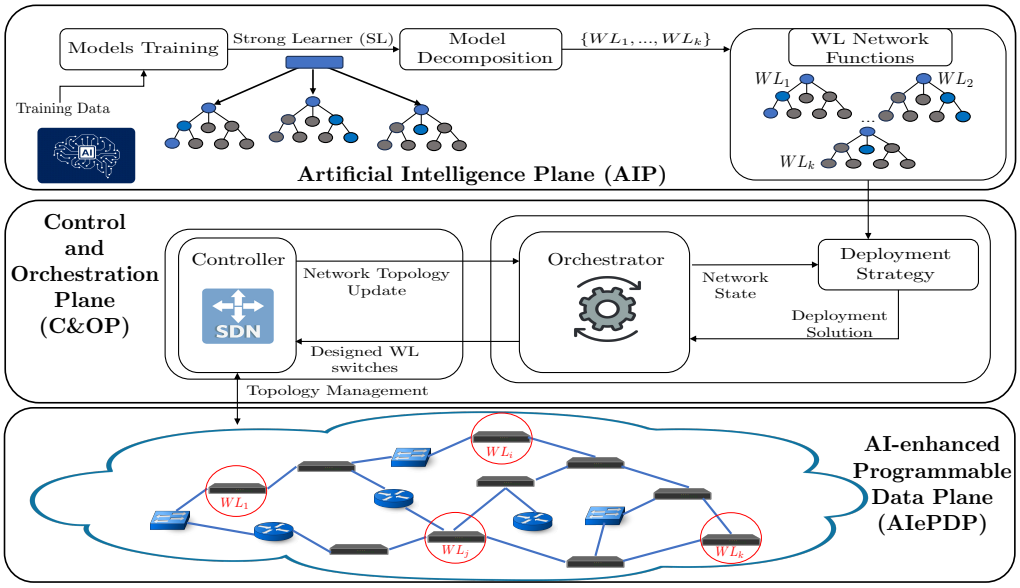


Figure 3.1: Logical Architecture of the Proposed In-Network Computing-based Active Intrusion Detection System

which decomposes them into individual Weak Learners (WLs). Once WLs have been generated, they are encoded as WLs^{VNF} and made available to the underlying plane through the *WL Network Functions* catalog. Further details on the Ensemble Decomposition are given in Section 3.3.1.1.

3.3.1.1 Split-AI: Projecting The Ensemble Learning over the Network

This section delves into the description of the proposed split-AI strategy to allow executing complex ML models *in-the-network*. This strategy is part of the AIP and it is applied after the training and testing procedures of an ML/DL model is correctly carried out. In light of this, it is crucial to let the reader note that the proposal builds on pre-trained ML/DL models which are then decomposed into main and lighter functional blocks to be deployed on networking devices without affecting their resources too much.

The idea behind the proposed split-AI strategy is built upon the concept of *Ensemble Learning*. More specifically, the technique builds a complex model, called SL, by aggregating the results from multiple individual less accurate models. The main rule to observe when applying Ensemble Learning is that each individual

model called *Weak Learner (WL)*, must demonstrate an accuracy of at least 51% (slightly greater than a random guessing). Formally, a SL is defined as follows:

$$SL = \bigcup_{i}^{n} (WL_i), \tag{3.1}$$

where n represents the number of WL that composes the complex model. Each WL, trained on a random subset of the set of features that are fed to the SL, processes the input feature vector $I = \{f_1, f_2, ..., f_k\}$ of the data, producing its own intermediate inference O_i:

$$\forall\ WL_i\ \in\ SL\ \rightarrow O_i = P_{WL_i}(I) \tag{3.2}$$

with P_{WL_i}, the prediction function is applied on the basis of the specific implementation of the WL. Once each WL_i completes its process, the intermediate inferences are gathered and aggregated applying to them a function V that produces the final inference O:

$$O = V(\{O_1, O_2, ..., O_n\}). \tag{3.3}$$

The function V is meant to analyze the comprehensive impact of each intermediate inference O_i on the final inference O_i.

This function can be chosen in different ways. For the purposes of this thesis, a simple typical implementation, based on the Majority Voting technique has been chosen: the most frequent class between the intermediate inferences is chosen as the final inference. This relies on the concept of the *wisdom of the crowd*. Obviously, in future activities, more sophisticated functions can be further explored.

It appears evident how the described behavior can be easily mapped onto the concept of VNF chaining and orchestration. The proposed split-AI strategy builds upon the decomposition of the WL components that made up the SL, distributing and executing them on programmable networking devices as SL^{VNF} and the decomposed WL_i^{VNF}, $i = \{1, ..., n\}$, to which they belong to:

$$SL = \{WL_1, WL_2, ..., WL_n\} \rightarrow SL^{VNF} = \{WL_1^{VNF}, WL_2^{VNF}, ..., WL_n^{VNF}\} \tag{3.4}$$

Clearly, since the SL is decomposed and its WLs distributed across the network, intermediate inferences are distributed as well. Therefore, it is essential that this

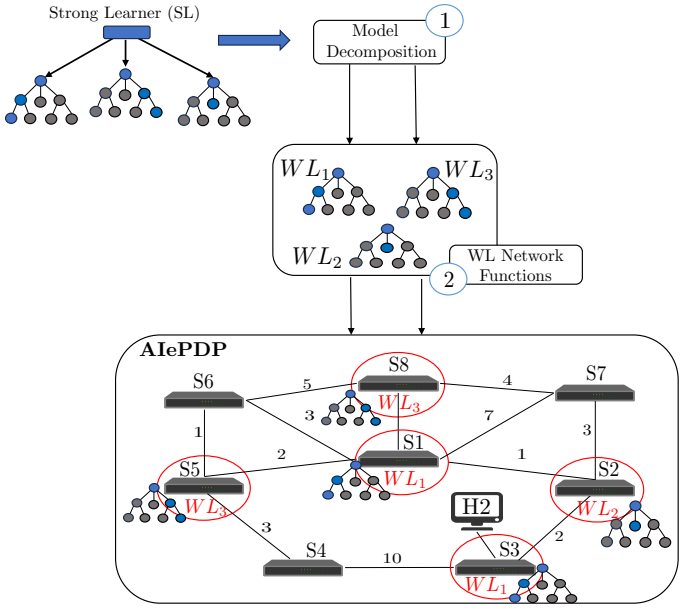


Figure 3.2: Proposed Split-AI Technique: a toy example.

information is transmitted from device to device running a WL in the network, in order to properly concatenate and orchestrate the WLs^{VNF}. Figure 3.2 shows a graphical representation of the split-AI process.

This solution is not only meant to build a ubiquitous and pervasive defense that takes place directly in the network, where the network packets are directly handled. The most appropriate choice of the nature and number of WLs in which the SLs are split must be carefully designed in order to fully exploit the ML/DL workflows and to do so by influencing the forwarding activities of the network devices as little as possible. In fact, the latter, by their nature, are not, individually, able to run complex AI models without reducing their performance; therefore, instead of abandoning the idea of running complex AI models within the network, the proposed split-AI technique must introduce a solution to the problem that relies on a cooperative and distributed action of the devices involved. In this direction, several state-of-the-art research works have addressed the problem of deploying entire AI models – to perform anomaly detection to identify malicious behavior in-network – into networking devices by mainly focusing on finding the best encoding method in order to meet the constraints and limitations of such devices. The proposal discussed in this thesis represents a shift in perspective for addressing the challenge

of deploying complex AI-relevant workflows in-the-network. This approach focuses more on the network itself, prioritizing its role, and exploiting the joint action of Distributed and Split AI and INC.

3.3.2 Control & Orchestration Plane (C&OP)

The functions implemented in this plane have the purpose of (i) developing the *Deployment Strategy* of the WLs^{VNF} in the catalog, determining their optimal displacement across the PDP devices, and (ii) enabling the selected switches (by deploying on them the WLs^{VNF}) to operate together as an in-network active IDS. As for the first function, given the WLs^{VNF} catalog and the current snapshot of the network, the objective is to find the strategy that maximizes the *security coverage* of the network. This means finding a set of WLs^{VNF} and relevant hosting switches that can effectively cover the considered network, intercepting the greatest possible number of flows to analyze (and on which to intervene). Subsequently, the SDN controller takes the output from the orchestrator and implements the chosen WL^{VNF} deployment plan in the network. The functions of this plane are crucial because the optimal deployment of such WLs^{VNF} allows to reduce the probability that an attack may not be detected and can reach its target. Besides, network topology and traffic distribution may change over time, hosts could be shut down, links may fail, and data plane devices may go down. Therefore, it must be possible to dynamically change the implemented deployment strategy to ensure that the *security coverage* is preserved. For this reason, the SDN controller sends network snapshots to the WL^{VNF} orchestrator to notify it of any network topology changes that require the computation of a new WL^{VNF} deployment plan. An intelligent deployment plan strategy is defined, based on an optimization problem that addresses and combines both the shortest path nature of the network and the constraint of letting flows pass through the decomposed parts of the entire model. Further details about it will be provided in Section 3.4.

3.3.3 AI-enhanced Programmable Data Plane (AIePDP)

This plane hosts both programmable and classic network forwarding elements (e.g. smart NICs, Programmable Switches, off-the-shelf hardware that hosts general-purpose VNFs, etc.). According to the WL^{VNF} *deployment plan* some of them

can be elected as in-network IDS elements that execute a WL^{VNF} function. Once a WL^{VNF} enabled PDP device detects a malicious flow, a cooperative and distributed analysis is started. The suspicious flow is marked as malicious and will be analyzed by the other on-the-path network elements equipped with the remaining WLs^{VNF} that need to be executed on the flow to re-construct the original SL. The network flow will cross the whole chain of WLs^{VNF}, by carrying along with it the produced intermediate inferences results encoded in the mentioned custom header. The last WL^{VNF} receiving the flow produces its inference, verifies the existence of all the information needed to make a decision and determines the final outcome about the flow by performing a Majority Voting. This newly proposed approach is called "*Projecting the Ensemble Learning over the network*". Flows detected as malicious, are thus blocked by the cooperating in-network IDS elements *without the need for human or controller intervention.* In such a way, the network is able to self-adapt to malicious events, counteracting them in an autonomous and timeliness way Obviously, enabling a limited-resource element, like a switch, to execute an ML/DL model without affecting its primary task, i.e. forwarding network flows, is a key issue to address. The proposed implementation of the AIePDP is inspired by the research presented in [46], and the WLs^{VNF} are designed and implemented using the P4 language. The concept of Match and Action (M/A) tables is exploited to encode the models within the programmable data plane. In this way, if the model needs to be retrained on new data sets, it can be easily redeployed to the programmable network device as a new set of M/A rules for the tables implemented via a Software-defined Network (SDN) controller through the southbound interface. RF is chosen as the ensemble SL model. Consequently, its WL components are DT. The encoding of RF and DT is designed following the implementation provided in [46].

3.3.3.1 PDP with Advanced Feature Extraction

The execution of ML/DL models strictly depends on the input feature vector, which is processed and based on which the inference is computed. Therefore, in addition to the representation of SL/WLs^{VNF} within the programmable data plane, it is essential to provide the PDP devices with a feature extraction function FE. This function takes as input the raw packet flow fl and extracts from it the network

features needed to feed the WL. Given a flow fl composed of a set of packets $p_1, p_2, ..., p_z$, the function FE computes the feature vector I:

$$FE(fl) = I = \{f_1, f_2, ..., f_n\}. \tag{3.5}$$

To implement such a function in the programmable data plane devices, the proposal leverages the stateful registers provided by the P4 to store the feature value. To enable the control plane to dynamically instruct the network device which features to extract, depending on the running WL^{VNF}, the proposal extends the state of the art with the concept of *One-Feature-One-Table*. The concept, as the name suggests, is based on the assumption of associating a table for each feature. Once each table has been designed on the network device, a procedure is needed that allows a flexible and timely method to instruct the switch which features it has to compute. Also for features, the proposed approach is based on the M/A rules used to instrument the programmable device. The table associated with each feature is composed of a key representing the unique ID used to identify the specific feature. The corresponding possible actions are instead *NoAction* or the action that has been implemented to extract that specific feature. In Fig. 3.3 an example of tables used to extract the SYN flag counter set to 1 in a given packet flow.

```
table syn_count{
        key = {
                feature_id: exact;
        }
        actions = {
                syn_count_extraction;
                NoAction;
        }
        default_action = NoAction;
}
```

Figure 3.3: Example of a Feature table.

The technique adopted to enable or disable feature extraction, via M/A rules installed by an SDN controller, is based on the *NoAction* action. More specifically, when it is necessary to disable feature extraction for a specific feature, the M/A rules that will be installed will consider the feature id as the key and *NoAction* as the action. This way, when the programmable network device tries to extract the

feature, by applying the relevant table, the resulting action will be *NoAction* and the feature will be ignored. For the remaining features that need to be extracted, the M/A rules will be in the form of "table_feature feature_id feature_id_extraction". Additionally, the default action is set to NoAction. This implies that only rules for features that need to be extracted need to be installed. Those that are not required will cause a table miss that will result in the default action, NoAction, being executed. This technique prevents the switch from extracting a predefined set of hard-coded features, which does not take into account the unique ones needed to query the running model. The feature extractor proposed in [46] is also improved, expanding its original capability of extracting only 12 network features to now extract 43 features, for both TCP and UDP traffic. This extension has been inspired by the CICFlowMeter feature extractor developed by the Canadian Institute for Cybersecurity (CIC) [47] and used in the creation of the CIC-IDS-2018 dataset. This improvement allows the network device to analyze a broader range of features for any type of network performance assessment. By expanding from the original 12 features to 43, the model can also exploit a more complete set of features, significantly improving its performance and accuracy. Finally, the designed feature extractor works on a per-flow basis. Each flow is identified by the 5-tuple consisting of source and destination IP addresses, source and destination ports, and protocol number. This 5-tuple is then hashed to create a 32-bit index used to look up registers that store the values of the relevant features for the flow.

Section 3.3.3.2 will be devoted to explaining the devised algorithm that guides the cooperative behavior of the distributed WLs allowing them to reconstruct the distributed ensemble.

3.3.3.2 Reconstructing the Ensemble

In order to allow distributed WLs^{VNF} to inform each other on the inferences carried out for a network flow, it has been considered a custom header, P4-encoded, as well as a logic procedure carried out by the PDP device augmented with the WLs^{VNF}. Fig.3.4 depicts the structure of the custom header.

Algorithm 3.1, instead, describes the procedure performed by the AI-augmented PDP devices that handle the information extracted by the custom header previously presented.

Algorithm 3.1 In-NetworkInferenceDistribution

```
1: Input: Incoming packet P, Threshold T
2: Output: Egress packet P
3: Data: WL identifier WL_i
4: procedure InNetworkInferenceDistribution
5:     H ← get_five_tuple(P)
6:     f_t ← get_flow_counter(H)
7:     if f_t = 0                                   ▷ P is the first packet of the flow H
8:         initialize_flow(H)
9:         Egress P
10:        exit
11:    end if
12:    f_t ← update_flow(H, P)        ▷ Increase the flow counter of the flow H and extract relevant features
13:    h_c ← get_info_header(P)                     ▷ Get the custom in-network classification header h_c
14:    if f_t < T
15:        if not empty(h_c)
16:            save_other_WL_inference(h_c)
17:        end if
18:        Egress P
19:        exit
20:    end if
21:    if empty(h_c)
22:        I ← get_existing_info(H)
23:        if empty(I)
24:            initialize_custom_header(h_c)        ▷ h_c :[WL_1, ..., WL_i, ..., WL_n|O_1, ..., O_i, ..., O_n]
25:        end if
26:    end if
27:    if inference_already_done(h_c)
28:        O_i ← get_inference(h_c)
29:    else
30:        O_i ← classify(H)
31:        save_intermediate(O_i)
32:    end if
33:    if completed_inference(h_c)
34:        final_inference ← majority_voting(h_c)
35:        if outcome = 1
36:            mark the flow H as malicious and signal the other PDP in the path.
37:            Drop packet P and those belonging to H.
38:            exit
39:        else
40:            Egress P
41:            exit
42:        end if
43:    else
44:        add_header_info(WL_i, O_i, h_c)          ▷ h_c[WL_i] = O_i
45:        Egress P
46:        exit
47:    end if
48: end procedure
```

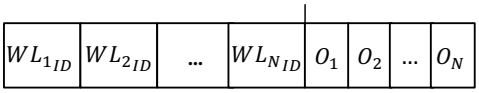


Figure 3.4: Custom Header for distributing intermediate inferences.

Since the SL is decomposed and its components are distributed among the net-

work devices, it is essential to orchestrate the WLs^{VNF}' operations in order to allow them to correctly exchange intermediate inferences. A custom header has been introduced to achieve this goal. Specifically, given a SL composed of n WLs – $SL = \{WL_1, WL_2, ..., WL_n\}$ – the proposed custom header, depicted in Fig. 3.4, is defined as follows:

- $WL_{i_{ID}}$ s.t. $WL_{i_{ID}} \in \{0,1\}^k \setminus \{0\}^k$ with $k = \lceil log_2(n) \rceil$ is the unique identifier of the $i-th$ WL. This field holds 00 only to denote a lack of available information, meaning that the relevant WL^{VNF} has not computed its inference yet.
- $O_i|_{i=1}^n$ represents the intermediate inference of each $i-th$ WL.

Once each WL^{VNF} finishes its process and computes the relevant intermediate inference, it puts its identifier and the computed result into the custom header and passes it to the programmable network device that runs the next WL^{VNF}. The condition $WL_{i_{ID}} \in \{0,1\}^k \setminus \{0\}^k$ is crucial for the designed bit-wise operation that allows a programmable network device, running a WL^{VNF}, to understand whether to apply or not the V function:

$$\bigwedge \bigvee_{i}^{n} (WL_{i_{ID}}) = 1. \tag{3.6}$$

As an example, consider the situation where the SL has been split into $n = 3$ WLs, and $k = \lceil log_2(n) \rceil = 2$ bits are employed to represent the three unique IDs. In this case, the set of IDs for the WL^{VNF} is $\{01, 10, 11\}$. By applying Eq. 3.6, $R1$ can be computed as $R1 = \bigvee(01) = 1$, $R2 = \bigvee(10) = 1$, and $R3 = \bigvee(11) = 1$. Finally, by using the bitwise AND operator, i.e., computing $R1\ AND\ R2\ AND\ R3 = 1$, the network device can understand that all $WL_{i_{ID}}$ fields have been filled in, and thus all intermediate inferences are available. It is worth noting that if any of the $WL_{i_{ID}}$ is 00, i.e., if the entire SL has not been reconstructed yet, the result of this calculation is 0 and the network device is aware that the V function cannot be applied.

The proposed method is designed to rely entirely on bit-wise operations natively supported by any programmable network device. Finally, the proposed solution does not rely on any ordering constraint between WLs^{VNF}: once any of the WLs^{VNF} has the chance to aggregate all intermediate inferences, it will apply the V function that will produce the final result O. As already stated, in the proposed

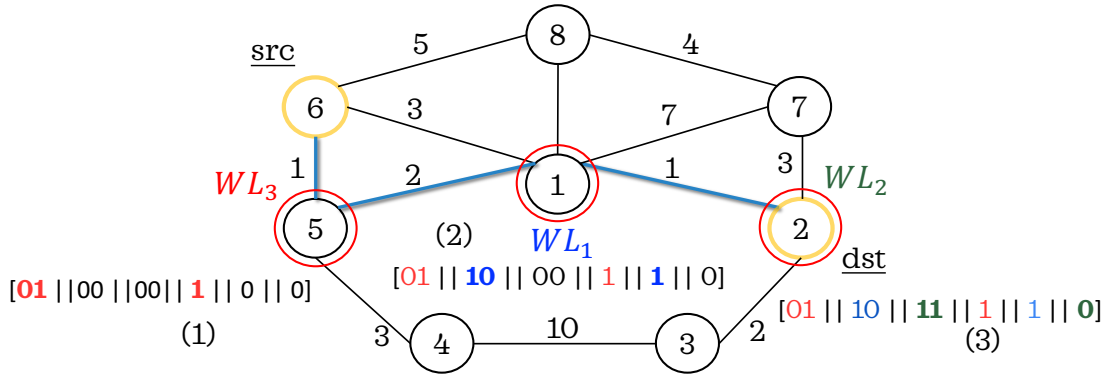


Figure 3.5: Toy Example with $n = 3$ WLs.

implementation, the V function is a majority voting technique. For the sake of clarity, Fig. 3.5 depicts a toy example of the process described so far. Future research activities could also be oriented to make each WL somehow instructed to trigger the majority voting process even if it has managed to aggregate not all intermediate inferences, but only the minimum number of them required to ensure an acceptable quality of the final result. On the basis of the defined custom header, the Algorithm 3.1 enables cooperation among the WLs that compose the SL. In NFV terms, the joint combination of the custom header and the Algorithm 3.1, allows chaining the WLs^{VNF} to reconstruct the decomposed SL. The algorithm processes the incoming packet P, that enters the ingress pipeline of the PDP device. The procedure starts with computing the five-tuple ID H to identify the network flow to which the packet P belongs. More specifically, H is obtained by hashing the five-tuple composed of (source IP, destination IP, source port, destination port, protocol id). To better understand, given a TCP (protocol id is 6) network flow between a source at 10.0.0.1:5542 and a destination at 10.0.0.2:5543. The packets exchanged between the source and the destination of that network flow are univocally identified by the five-tuple (10.0.0.1, 10.0.0.2, 5542, 5543, 6). The obtained identifier H is used to access the register that contains the current amount of packets f_t gathered by the PDP device for the corresponding flow. If the current amount of packets for H is 0, it means that the packet P is the first packet of the flow. In this case, the algorithm initializes the flow by storing its identifier H and initializing the registers that are meant to store the relevant network features. The packet P is then sent to the egress pipeline of the PDP device in order to be sent to the next hop through the

appropriate port. If f_t is not zero, it means that the device has already registered a previous packet belonging to H and, therefore, network features associated with this specific flow need to be updated. At this point, it is crucial to check whether the packet is bringing with it information relevant to the network flow classification performed by the WLs in the network. The algorithm extracts the custom in-network classification header h_c (see Figure 3.4) from the packet P. If the custom header has information relevant to intermediate inferences O_i performed by previously traversed PDP devices, this information is stored, since it can be crucial to complete the execution of the majority voting procedure. In addition, if the amount of packets for this flow – on this specific PDP device (WL^{VNF}) – it means that it cannot execute the WL model. The packet P is sent over the network by following the routing plan. If the algorithm reaches line 21 it means that the amount of packets for the flow has reached the pre-defined threshold and therefore the extracted features can be used to query the WL^{VNF}. If the custom header is empty it means that this packet has not encountered any other WL^{VNF} augmented PDP and the current is the first one encountered. The algorithm will check whether the current PDP has previously stored intermediate inferences carried out by other WLs^{VNF} on the flow identified by H. If this is not the case, the custom header h_c for P is initialized. Then it is crucial to check if the packet has been previously classified by the current current PDP. If it is true, the model is not queried and the relevant intermediate (previously saved) inference is read from the register. Otherwise, the WL^{VNF} is executed and it is fed using the extracted features relevant to the flow H. If the current PDP device was the last WL^{VNF} needed to complete the chain and reconstruct the SL, the majority voting (the V function) is executed, and if the outcome is 1 the flow is recognized as malicious and all its packets will be dropped.

3.4 Network and Security Aware Deployment Plan for the C&OP: All Pairs Shortest Path Coloring

This section will dive into the detail of the proposed deployment strategy that, given an SL and its decomposed WL_i^{VNF}, distribute such VNFs on the available programmable networking devices. This is the core of the (C&OP). Particularly,

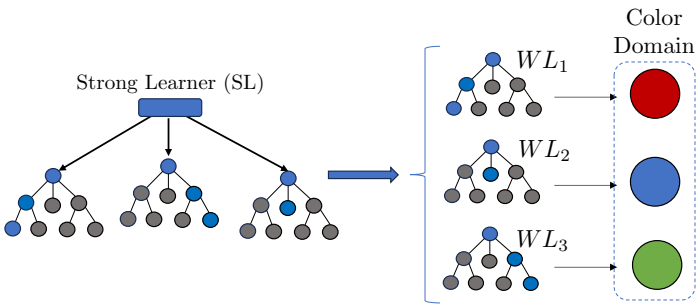


Figure 3.6: From WL-VNFs to Colors domain.

this deployment must be computed and applied by taking into account two main constraints: (i) flows of packets traverse the network following the shortest path from a source to a destination; (ii) being the SL decomposed into its individual WLs, routing paths must be computed with the constraint of letting network flows passing through the programmable nodes that execute WLs. This is crucial to being able to reconstruct the output of the SL. For such a reason, a variant of the shortest path problem is proposed to optimize the deployment of VNFs in next-generation networks. Given that a web network is required to determine if a data flow from a source node to a target node is malicious, the flow must go through a set number of different types of VNFs. After passing through these functions, a majority voting system is used to decide if the flow is malicious. If most of these VNFs agree that the flow is suspicious, it is identified as malicious. The network is represented using a graph. The nodes of the designed model represent the nodes of the network in which the VNFs will then be deployed, while the edges denote the physical and virtual connections among network elements. Node coloring is used to represent the implementation of specific VNFs, where each color corresponds to a different type of VNF and the coloring cost corresponds to the associated implementation cost. It is worth noticing that in the considered case, the VNFs/colors[1] refer to SL^{VNF} and the associated WLs^{VNF}, once the SL is decomposed. For instance, an SL composed of three WLs will determine three WL^{VNF} and therefore three different colors (e.g. red, green, and blue), as shown in Figure 3.6. The graph edges are weighted to reflect a network connection characteristic, such as latency or bandwidth. The objective is to find the optimal deployment of VNFs to en-

[1]The terms "color" and SL/WL^{VNF} will be used interchangeably. More specifically, SL^{VNF} refers to a scenario in which only one color is needed.

sure comprehensive network security coverage, thereby making the network itself the first line of defense against cyber-attacks. This approach guarantees pervasive and ubiquitous network protection, aligning with the need for robust cyber-security measures in the evolving landscape of next-generation networks. Practically, the behavior of the shortest path problem is modified by adding and taking into account coloring constraints designing and introducing a new model named All-Pairs Shortest Path Coloring problem (APSPC), where the cost to be minimized includes both the costs of the different paths between pairs of source nodes and target nodes, ensuring that each path passes through at least one colored node for each color, as well as the cost of coloring the nodes themselves. In the next sections a detailed mathematical model that represents the problem and a new decoder (meta heuristic) for a Biased Random-Key Genetic Algorithm (BRKGA), designed to effectively address this variant of the problem, will be presented. More specifically, an exact mathematical model is first defined and introduced, capable of solving the problem within a finite amount of time for instances of limited size. Subsequently, a genetic meta-heuristic is employed to overcome this limitation.

3.4.1 Exact Model

This section delves into the mathematical complexities of the All-Pairs Shortest Path Coloring (APSPC) problem through the development of an Integer Linear Programming (ILP) model. The problem is formulated on an undirected connected loopless graph $G = (V, E)$, with the goal of determining the simple shortest paths between all pairs of nodes (source-target) such that each path includes at least one vertex colored for each color in the set $C = \{1, 2, \ldots, h\}$. Despite the undirected nature of the graph, this model incorporates directed flow constraints, which are necessary for the formal definition of paths from a source node s to a target node t. For this reason, with the abuse of terminology, once the nodes s and t have been fixed, any node can have outgoing and incoming edges. Three sets of binary variables are introduced to indicate whether an edge is traversed and whether a vertex is colored with a specific color; specifically, let x_{ij}^{st} be a binary variable equal to 1 if and only if the edge (i, j) is visited in the path s–t, and y_{ic} be a binary variable equal to 1 if and only if the vertex i is colored by c in the graph. The last set of variables keeps track of the coloring of the nodes in each path s–t. In

particular, given the color c, fixed the source s and the target t, z_{ic}^{st} must be equal to 1 if and only if in the path s–t the vertex i is colored with c and is traversed. In addition, let $w_{ij} \in \mathbb{Z}^+$ be the positive weight associated with each edge (i, j) and $p_c \in \mathbb{Z}^+$ the cost of coloring a node with color c. The APSPC problem presented can be formulated using the following programming model.

$$\min \sum_{(s,t)\in V\times V} \sum_{\substack{(i,j)\in E:\\ i\neq t \wedge j\neq s}} w_{ij} \cdot x_{ij}^{st} + \sum_{(i,c)\in V\times C} p_c \cdot y_{i,c} \tag{3.7}$$

s.t.

$$\sum_{j\in V\setminus\{s\}} x_{ij}^{st} - \sum_{j\in V\setminus\{t\}} x_{ji}^{st} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \quad \forall\, i, s, t \in V \tag{3.8}$$

$$\sum_{(i,j)\in E(S)} x_{ij}^{st} \leq \sum_{i\in S\setminus\{k\}} \sum_{j\in V\setminus\{s\}} x_{ij}^{st} \quad \begin{matrix} \forall s,t\in V; \forall k\in S; \\ \forall S\subsetneq V\setminus\{s,t\}: |S|\geq 2 \end{matrix} \tag{3.9}$$

$$\sum_{c\in C} y_{ic} \leq 1 \qquad \forall i \in V \tag{3.10}$$

$$\sum_{i\in V} z_{ic}^{st} \geq 1 \qquad \forall\, s, t \in V; \forall c \in C \tag{3.11}$$

$$z_{ic}^{st} \leq \sum_{j\in V\setminus\{s\}} x_{ij}^{st} \qquad \forall\, s, t \in V; \forall i \in V \setminus \{t\}; \forall c \in C \tag{3.12}$$

$$z_{tj}^{st} \leq \sum_{j\in V\setminus\{t\}} x_{jt}^{st} \qquad \forall\, s, t \in V; \forall c \in C \tag{3.13}$$

$$z_{ic}^{st} \leq y_{ic} \qquad \forall\, s, t, i \in V; \forall c \in C \tag{3.14}$$

$$x_{ij}^{st} \in \{0, 1\} \qquad \forall (i, j) \in E \tag{3.15}$$

$$y_{ic} \in \{0, 1\} \qquad \forall (i, c) \in V \times C \tag{3.16}$$

$$z_{ic}^{st} \in \{0, 1\} \qquad \forall s, t, i \in V; \forall c \in C. \tag{3.17}$$

The objective of the model (3.7) is to minimize the total weight of the traversed edges and the cost of coloring the nodes. Constraints (3.8) ensure flow conservation, and equations (3.9) are subtour elimination constraints represented in cutset form, named Generalized Cut-Set (GCS) inequalities. This latter set of constraints ensures that the number of edges with both extremes in S, i.e., $|E(S)|$, cannot be greater than the number of vertices in S traversed from the s–t path. This type of constraint is necessary due to the coloring constraints (3.11)–(3.13), which could generally induce cycles disconnected from the simple path s–t. Constraints (3.10) ensure that each node is colored with at most one color, and constraints (3.11) ensure that in each shortest path s–t, there is at least one colored vertex for each color

Table 3.1: Sets, parameters, and decision variables.

Sets	
V	set of vertices
E	set of edges
C	set of colors
$E(S)$	set of edges induced by $S \subseteq V$
Parameters	
$w_{ij} \in \mathbb{Z}^+$	weight of the edge (i, j)
$p_c \in \mathbb{Z}^+$	cost of color c
Variables	
x_{ij}^{st}	binary variable equal to 1 iff the edge (i, j) is visited in the path s–t
y_{ic}	binary variable equal to 1 iff the vertex i is colored by c
z_{ic}^{st}	binary variable equal to 1 iff the node i is colored by c and is visited in the shortest path s–t

$c \in C$. The constraints (3.13) establish a relationship between the variables z_{ic}^{st} and x_{ij}^{st}, similarly, constraints (3.14) relate the z_{ic}^{st} variables to the y_{ic} variables. Finally, constraints (3.15)–(3.17) define the variable domains.

Additionally, a separation procedure is developed for the computationally expensive subtour elimination constraints (3.9). So, initially, the relaxed problem is considered, meaning the subtour elimination constraints are temporarily omitted. During the resolution process, any violated subtours in the current solution are identified. Regarding the separation routine, a method considered in [48] is used, focusing on identifying the strongly connected components in the graph induced by the current solution. Violated *GCS* constraints are dynamically added to the model using a modified version of Tarjan's algorithm ([49]), as proposed by [50]. Table 3.1 describes the notation used to formulate the mathematical model.

3.4.2 Genetic Meta-heuristic

The BRKGA is a significant advancement in genetic algorithms, developed to tackle complex and large-scale combinatorial optimization problems. Based on the principles of natural selection and survival of the fittest, the BRKGA uses a population of solutions represented as vectors of real numbers between 0 and 1, known as random

keys. This representation allows for a clear separation between the solution space of the algorithm and the solution space of the specific problem, greatly simplifying the development and application of the algorithm across various domains. A key component in the BRKGA is the decoder, a deterministic function that maps the random-key vectors to the solution space of the specific problem. The decoder ensures that each vector is translated into a solution, maintaining consistency and reproducibility of the results. The BRKGA has been successfully applied in various real-world contexts, such as over-the-air software update scheduling, network design, and combinatorial auctions. Its modularity and ability to hybridize with other optimization techniques make it a powerful and versatile tool for addressing challenging and dynamic optimization problems. In the conducted study, it is considered a multi-parent and multi-populations BRKGA with bidirectional Permutation-based Implicit Path-Relinking (IPR-Per) ([51]). This combined approach further enhances genetic diversity and solution quality, allowing for more effective exploration of the solution space. During the evolution process of the BRKGA considered, several key operations are utilized. It starts by creating the first generation of m populations and using a seed to generate all the chromosomes. The size of a single population is calculated as $p := \alpha \cdot n$, where $\alpha \geq 1$ is called population size factor; the elite population is defined as $p_e := p \cdot pct_e$, where $pct_e \in [0.1, 0.25]$ is the elite percentage parameter; finally, the size of the mutant population is $p_m := p \cdot pct_m$, where $pct_m \in [0.1, 0.3]$ is the mutant percentage. In the second step, the decoder converts the chromosomes in the APSPC solutions and consequently computes the fitness values. After decoding the chromosomes, the IPR-Per is performed to try to improve the best current solution, receiving two chromosomes as input. If the stopping criteria are not reached, then the next step is to create a new generation and the process is repeated by decoding new populations. In particular, the population of the current generation is divided into two parts according to fitness: the elite population p_e containing the chromosome with the best fitness and the non-elite population p_{ne} which contains the rest of the chromosomes. The elite individuals are directly copied to the next generation to preserve high-quality solutions. Mutation introduces new random individuals to explore new areas of the solution space. The remaining part of the population, $p(1-pct_e-pct_m)$, is generated by the *multi-parent crossover*. For this crossover it is necessary to choose three parameters, the number of the total of parents (π_t) and elite parents (π_e) to be selected; the probability that

each parent has of passing genes on to their child. The probability is calculated taking into account the bias of the parent, which is defined by a pre-determined, non-increasing weighting bias function (ϕ) over its rank r. Multi-parent crossover allows multiple parents to contribute to the new offspring, increasing genetic diversity. Multi-population evolution enables multiple populations to evolve in parallel and exchange their best individuals, reducing the risk of premature convergence. Regarding global stopping criteria, two rules have been taken into consideration. The procedure is interrupted if either the set time limit or the maximum number of consecutive iterations without improvement (*wi*) are reached.

Algorithm 3.2 decode

```
1: input chromosome, n := number of nodes (dimension of the chromosome)
2: procedure DECODE
3:     Initialize random generator gen with seed chromosome[0]
4:     Reset nodeColors to −1 for all nodes
5:     for i ← 0 to n
6:         Select a random color using gen
7:         colorCost ← colorCosts[color]
8:         if SHOULDCOLORNODE(i, chromosome, colorCost, gen)
9:             nodeColors[i] ← color
10:        end if
11:    end for
12:    fitness ← CALCULATEFITNESS(nodeColors)
13:    return fitness
14: end procedure
```

Algorithm 3.2 is designed to transform a chromosome into a solution for the APSPC, evaluating its quality through a fitness function, i.e., it represents the decoder. The procedure begins with the initialization of a random number generator *gen* using the first value of the chromosome as the seed (line 3). This ensures that the random generation operations are reproducible throughout the entire genetic evolution. In line 4, all nodes are initially uncolored. This is represented by setting *nodeColors* to -1 for each node. The procedure iterates with a **for** loop over all nodes to determine whether each node should be colored or left uncolored. In particular, for each node, in line 6 a random color is selected using the random number generator *gen*. The cost associated with the selected color is calculated by accessing the *colorCosts* vector. It is then checked whether the node should be colored using

the *shouldColorNode* function (line 8), which takes the node, the chromosome, the color cost, and the random number generator as inputs. In line 9, if the node should be colored, the color is assigned to the node. Once colors have been assigned to all nodes, the fitness of the solution is calculated using the *calculateFitness* function in line 12, which evaluates the quality of the solution based on the assigned colors. Finally, the procedure returns the calculated fitness value.

Algorithm 3.3 shouldColorNode

```
1: procedure SHOULDCOLORNODE(node, chromosome, colorCost, gen)
2:     nodeDegree ← GETNODEDEGREE(node)
3:     avgNodeWeight ← GETAVGNODEWEIGHT(node)
                                        ▷ Phase 1: Probability based on color cost
4:     ColorCostFactor ← colorCost / (avgNodeWeight × (n - 1))
5:     if ColorCostFactor ≤ 0.1
6:         return true
7:     end if
                                ▷ Phase 2: Probability based on other node characteristics
8:     if chromosome[node] ≥ 0.1
9:         NodeProbability ← chromosome[node] × nodeDegree / avgGraphDegree
   × avgGraphWeight / avgNodeWeight
10:    else
11:        NodeProbability = 1
12:    end if
13:    dis ← UNIFORMREALDISTRIBUTION(0.0, 1.0)
14:    return (dis(gen) < NodeProbability)
15: end procedure
```

The focus will now be posed on the functions 3.3 and 3.4, to describe them in detail. Algorithm 3.3 is designed to determine whether a node in the graph should be colored based on the node's characteristics. The procedure begins by getting the degree of the input node and the average weight of the edges incident to the node (*avgNodeWeight*). The decision process is divided into two phases to ensure a balanced evaluation, it is sufficient that one of the two phases is verified for the node to be colored. In Phase 1, the procedure calculates the *ColorCostFactor* as a function of the color cost and *avgNodeWeight* (line 4). This probability assesses the cost-effectiveness of coloring the node. If the ratio is very low, the node is colored with certainty. Intuitively, this means that the node is colored if the cost of coloring

is relatively small compared to the benefit gained from coloring it. Phase 2 focuses on other characteristics of the node. The procedure calculates the *NodeProbability* as a function of the ratio between *nodeDegree* and *avgNodeWeight*, and the chromosome gene associated with the node (line 9). This operation allows to determine how important it is to color a node based on the number of connections and the strength of those connections (average edge weight). If these values indicate that the node is influential in the network, then the probability of coloring it increases. If the gene is too low, the probability is set to one to avoid invalidating the probability calculation. Finally, a random number is generated using a uniform distribution between 0.0 and 1.0, and the node is colored if this random number is less than *NodeProbability* (line 11). The procedure returns the boolean result, indicating whether the node should be colored or not.

Algorithm 3.4 calculateFitness

```
 1: procedure CALCULATEFITNESS(nodeColors)
 2:     totalCost ← COLORCOST(nodeColors)
 3:     for i ← 0 to n
 4:         for j ← i + 1 to n
 5:             (cost, path) ← DIJKSTRAWITHCOLORS(nodeColors, i, j)
 6:             if cost ≠ −1
 7:                 totalCost ← totalCost + cost
 8:             else
 9:                 totalCost ← FITNESS_MAX
10:                 break
11:             end if
12:         end for
13:         if totalCost == FITNESS_MAX
14:             break
15:         end if
16:     end for
17:     return totalCost
18: end procedure
```

Algorithm 3.4 is designed to calculate the fitness of a solution by evaluating the total path cost between all pairs of nodes in the graph, given their color assignments. The procedure starts by initializing *totalCost* to the result of the *colorCost* function, which computes the total cost of the color assignments. In lines 3–5, the procedure

iterates over each pair of nodes and calculates the cost of the shortest path using a modified Dijkstra algorithm, incorporating color constraints. If the cost is not -1, indicating a valid path, the cost is added to *totalCost* (line 7). If no valid path exists, *totalCost* is set to *FITNESS_MAX* to indicate an infeasible solution, and the loops are terminated. Finally, the procedure returns the *totalCost*.

3.4.3 Intelligent SL split Selection

The number of colors available to color the nodes of a given graph is chosen using the function defined below, denoted as $cd : \mathbb{R} \to 2\mathbb{Z} + 1$. Given a real number x, this function, returns the largest odd integer less than x or returns 3 if the largest integer less than x is 2. Formally:

$$cd(x) := \begin{cases} 3 & \text{if } \lfloor x \rfloor = 2 \\ \lfloor x \rfloor - 1 & \text{if } \lfloor x \rfloor \in 2\mathbb{Z} \setminus \{2\} \\ \lfloor x \rfloor & \text{if } \lfloor x \rfloor \in 2\mathbb{Z} + 1. \end{cases}$$

The exact number of colors, $\#colors$, available for the graph $G := (V, E)$ is given by evaluating the function cd in the average number of nodes present in all classical shortest paths, i.e. without the coloring constraint.

$$\#colors = cd\Big(\frac{2}{|V| \cdot (|V| - 1)} \sum_{(i,j) \in E | i<j} d(i, j)\Big), \tag{3.18}$$

where $d(i, j)$ is the number of nodes present in the classical shortest path between i and j calculated using the Dijkstra algorithm.

3.5 Performance Evaluation

This section assesses the proposed distributed Split-AI strategy, highlighting the benefits it offers not only in terms of scalability but also in preserving the forwarding capabilities of AI-augmented networking devices. The performance evaluation is divided into two parts: (i) an assessment of the devised optimization strategy – namely APSPC – in determining the optimal deployment of the decomposed partitions of the SL; (ii) an evaluation of network-relevant performance metrics.

3.5.1 Model Evaluation Campaigns

This section provides a summary of the results obtained from computational experiments conducted on the exact model and the defined meta-heuristic. Specifically, an in-depth analysis is presented to evaluate the impact of various network characteristics on the overall effectiveness and efficiency of the proposed system.

Both the BRKGA and mathematical model have been implemented in C++ using clang version 14.0.3. For the compilation, the C++17 standard was set using the CMAKE_CXX_STANDARD 17 specification in the CMake configuration file. All the optimization computational tests were conducted using an Apple M2 Max processor with CPU 12-core and GPU 38-core and 96 GB LPDDR5 of RAM running macOS Ventura 13.3. The exact model was solved with the optimization solver CPLEX 12.10 imposing a time limit of 3600 seconds (s) for each instance.

Considering the complexity due to numerous parameters of the BRKGA, a preliminary tuning phase is conducted a using the *irace* package (refer to [52] for details). This tool performs automatic configuration to optimize parameter values. The main code is written in R, while the interface between *irace* and the designed metaheuristics has been developed in Python.

3.5.2 Instances and Parameter Setting

In order to evaluate the performance of the proposed approach, a set of instances was generated as described below. The set is composed of random topology networks, each of which is identified by a unique combination of the following parameters: number of nodes (n), edge density (d), and color cost ranges (cr). In particular, the following choices are then considered: four values for the number of the nodes, i.e., $n \in \{10, 15, 25, 30\}$; four values for the edge density, that determines the number of the edges $\#e = d \cdot n(n-1)/2$, with $d \in \{0.25, 0.35, 0.45, 0.55\}$; and four ranges of values for the color cost, i.e., $cr_1 = [1, 125]$, $cr_2 = [50, 150]$, $cr_3 = [75, 175]$, $cr_4 = [100, 200]$. For each instance, the number of colors is uniquely determined by the function (3.18). In more detail, given a certain number of nodes, a minimum spanning tree $G = (V, E)$ is first generated to ensure connectivity, and then edges are randomly added to E until the required number of arcs, determined by the edge density parameter, is reached. The costs of the edges are determined as a sample from a uniform distribution in the interval $[1, 200]$. The color costs are determined

as a sample from a uniform distribution in the color costs value range parameter. For each scenario, identified by a given combination of values of n, d, and cr, six different random instances are generated, for a total of 384 instances, by varying the seed used to initialize the random number generator.

Each set is organized into four classes, based on edge density, named $\{\mathrm{ED}_i\}_{i=1}^{4}$. In addition to the parameters mentioned in 3.4.2, three parameters need to be set for IPR-Per parameters. The parameter pct_p sets the path size which determines whether the path-relinking procedure covers the entire path between the two chromosomes or if it is truncated at a specific point. The minimum distance (md) that two chromosomes must respect to be considered different and to be chosen as candidates for path-relinking. Finally, the *sel* parameter represents the method of choosing the guide and base chromosomes. There are two possible options, *randS* and *bestS*, which represent random selection and best chromosome selection, respectively. In both cases, the reference population is the elite one. For the metaheuristic parameters, tuning phase is carried out using *irace* software. This tuning was done using four random instances of each of the AD_i sets. Table 3.2 summarizes the tuned parameters of the BRKGA, grouping them into three sets: *Operator*, *IPR-Per*, and *Others*.

Table 3.2: Tuned BRKGA parameters.

Operator		$IPR-Per$		*Other*	
pct_e	0.1	*sel*	randS	α	20
pct_m	0.6	md	0.15	m	2
π_t	3	pct_p	0.85		
π_e	1				
ϕ	$1/r^2$				

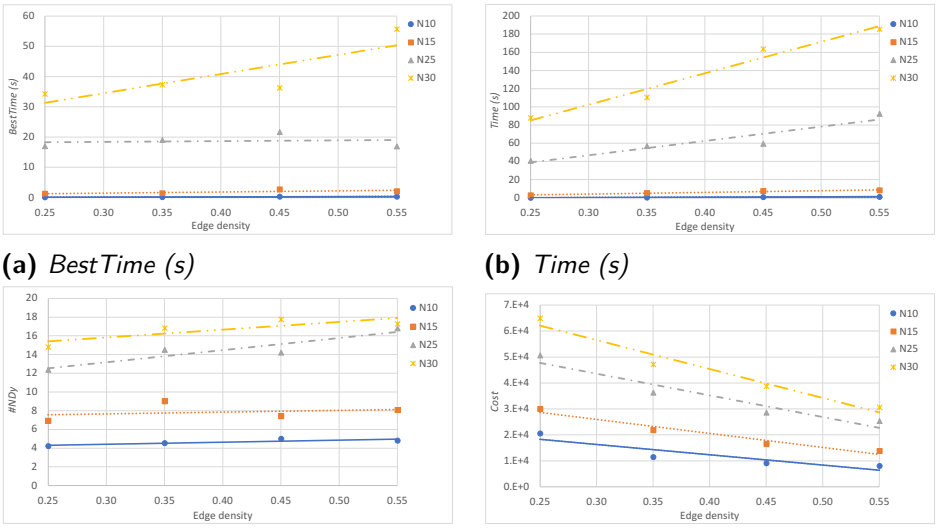
3.5.3 Experimental Results

The summary table will be presented by grouping instances according to their density class ED_i and the number of nodes. Each row in the tables refers to a subset of instances from a given set that share the same edge density and, where specified, the same number of nodes. These are indicated by the descriptor in the *Set* column, where the acronym "ED" stands for edge density and "N" stands for nodes.

Furthermore, all time values are measured in seconds. Table 3.3 provides detailed information on the results obtained by applying BRKGA to the set of all instances. Each row reports the average values for the following parameters: the number of available colors in the instances ($\#colors$), calculated using the *cd* function; the time taken by the metaheuristic to identify the obtained solution ($BestTime$ (s)); the total execution time ($Time$ (s)); the number of deployed nodes ($\#NDy$); the total solution cost ($Cost$); color-related costs ($Cost_c$); and path cost ($Cost_p$). The number of referred instances is 24 for each row aggregating on both the edge density and the number of nodes, and 96 for the AVG rows aggregating only on the arc density. For all the experiments, the time limit is set equal to 900 seconds, and the maximum of consecutive iterations without improvement *wi* to 10. The plots in Fig. 3.7 represent the trend of the average results for $BestTime$, $Time$, $\#NDy$, and $Cost$ as edge density and number of nodes vary.

Analyzing the behavior of the average best time, it increases as expected as both the number of nodes and the density increase. However, the effect of the number of nodes is more significant compared to the density, while still remaining below 1 minute. In particular, as shown in the Table 3.3 and in Fig. 3.7.(a), it can be observed that: with 10 nodes, the $BestTime$ consistently remains around 0.08–0.32 seconds regardless of the density. With 15 nodes, it increases significantly compared to 10 nodes, but remains manageable, ranging between 1.22 and 2.70 seconds. With 25 nodes, there is an increase, but still limited, in fact, it rises to 16.99 seconds for *ED1* and 21.62 seconds for *ED3*. With 30 nodes, the highest recorded $BestTime$ is observed, with values ranging from 34.21 seconds for *ED1* to 55.65 seconds for *ED4*. In general, it is observed that as the density increases, the $BestTime$ increases linearly for each number of nodes. This increase becomes greater as the number of nodes increases, i.e., the slope of the linear trend line increases. In addition, for each density class, it is noted that as the number of nodes increases, the $BestTime$ increases in a non-linear manner. Similarly, the total runtime of the BRKGA follows a linear trend as the density increases for each number of nodes (as shown in Fig. 3.7). (b), and a non-linear trend as the network size increases for each density class.

Regarding the average number of colors identified by the *cd* function, it is observed that, on average, the number of colors increases as the density decreases. Specifically, in all instances with 10 and 15 nodes, $\#colors$ is always equal to the minimum available, which is 3. With 25 nodes, the average ranges from 3.08 in the *ED3*



(a) *BestTime (s)* **(b)** *Time (s)*

(c) *#NDy* **(d)** *Cost*

Figure 3.7: Trend of the average results obtained from the experiments with respect to edge density for each node class.

class to 3.17 in *ED1*, while in the *ED4* class, all instances have $\#colors$ equal to 3. Overall, 5 instances with 5 colors were recorded. With 30 nodes, the highest $\#colors$ values are recorded, ranging from 3.08 in the *ED4* class to 3.42 in *ED1*. In total, 4 instances with 7 colors and 3 instances with 5 colors were recorded. Therefore, for each density class, as the number of nodes increases, $\#colors$ also increases. These trends can be explained by the fact that, in fully random topologies with a greater number of nodes and/or relatively low density, it is more likely to find, on average, the shortest path with higher length, which requires the use of more colors, as expected from the definition of the *cd* function.

As expected, $\#NDy$ increases with the total number of nodes in the network. For example, in the case of 10 nodes and density class *ED1*, the average number of deployed nodes is 4.21, while with 30 nodes in the same class, it increases to 14.79. This trend is consistent across all classes, confirming that as the graph size increases, more nodes are involved in the deployment of learning models and VNFs necessary to ensure network security coverage. With the same number of nodes, it is observed that as density increases, the number of deployed nodes tends to increase. For instance, for $N = 15$, $\#NDy$ increases from 6.92 in density class *ED1* to 9.00 in class *ED2*, and 8.08 in class *ED4* (as shown in Fig. 3.7.(c)). The

scalability of the proposed model is evident from the way it adapts to networks of varying sizes and densities. The increase in $\#NDy$ with the growth in both the number of nodes and density shows that the model can handle larger and more complex network topologies. This scalability is crucial for next-generation networks, where the number of nodes and connections will continuously increase, requiring an efficient distribution of learning functions across the network.

The increase in the number of nodes has a significant impact on the total costs for each density class. For example, observing the results in the table, for 10 nodes and *ED1*, the *Cost* is around $2 \cdot 10^4$, while for 30 nodes in the same density class, the cost rises to approximately $6.5 \cdot 10^4$. This increase is attributable to the rise in both deployment costs ($Cost_c$) and shortest path costs ($Cost_p$), as larger networks require the distribution of VNFs across more nodes and covering longer distances. Density, however, follows a different trend. As density increases, $Cost_p$ decreases because the paths between nodes become shorter. Nevertheless, $Cost_c$ tends to rise slightly with the increase in density, as more nodes are needed to manage the more connected network. Therefore, since $Cost_p$ constitutes the vast majority of the total cost for each set of instances (over 90%), the average total cost decreases, as can be seen from the AVG rows.

3.5.4 Network Evaluation Campaigns

In a subsequent experimental campaign, the performance of the data plane devices is compared when handling an entire ML model versus when the ML model is decomposed according to the proposed deployment approach. The considered performance evaluation metrics include time to obtain the classification outcome – namely *classification time* – and the *throughput* guaranteed by the networking devices that execute the additional and AI-related task.

The objective is to assess that under heavy network load, e.g. volumetric Distributed Denial of Service (DDoS), the reduced workload imposed on the single data plane device will lead the network to scale well in these critical situations guaranteeing the forwarding activities. The network is tested by considering different attack intensities, starting with 100 pkt/s generated by each of the attackers and reaching 1000pkt/s with an incremental step of 100 pkt/s. To characterize the size of the DoS/DDoS packets, the DDoS evaluation dataset (CIC-DDoS2019) [53]

Table 3.3: Detailed results of the BRKGA

Set	$\#colors$	$BestTime\ (s)$	$Time$ (s)	$\#NDy$	Cost	$Cost_c$	$Cost_p$
N10ED1	3.00	0.08	0.16	4.21	20605.67	516.46	20089.21
N15ED1	3.00	1.22	2.82	6.92	30081.75	941.67	29140.08
N25ED1	3.17	16.99	40.80	12.38	50684.38	1641.13	49043.25
N30ED1	3.42	34.21	88.02	14.79	64995.96	1920.00	63075.96
AVG	3.15	13.13	32.95	9.57	41591.94	1254.81	40337.13
N10ED2	3.00	0.17	0.45	4.58	11526.38	579.38	10947.00
N15ED2	3.00	1.40	5.29	9.00	21894.17	1200.54	20693.63
N25ED2	3.17	19.04	57.06	14.50	36370.63	1908.21	34462.42
N30ED2	3.25	37.27	110.52	16.83	47228.67	2190.71	45037.96
AVG	3.10	14.47	43.33	11.23	29254.96	1469.71	27785.25
N10ED3	3.00	0.32	0.81	4.88	9140.13	618.00	8522.13
N15ED3	3.00	2.70	7.61	7.42	16526.63	865.92	15660.71
N25ED3	3.08	21.62	59.56	14.21	28602.08	1900.33	26701.75
N30ED3	3.17	36.21	163.72	17.75	38783.88	2277.92	36505.96
AVG	3.06	15.21	57.92	11.06	23263.18	1415.54	21847.64
N10ED4	3.00	0.30	1.15	4.96	8436.79	681.58	7755.21
N15ED4	3.00	2.09	8.26	8.08	13872.04	1076.79	12795.25
N25ED4	3.00	16.94	92.46	16.79	25438.17	2081.63	23356.54
N30ED4	3.08	55.65	185.47	17.25	30632.79	2004.63	28628.17
AVG	3.02	18.75	71.84	11.77	19594.95	1461.16	18133.79

is studied and analyzed. The dataset contains real-world data, recorded by the Canadian Institute for Cybersecurity (CIC), representing the most common DDoS attack types – characterized by means of 80 network features – such as SYN flooding, UPD DDoS, DNS-based DDoS, WebDDoS, and many others. On the basis of the analysis conducted on the average packet size (*Avg Packet Size* feature), the attack packet size is uniformly chosen in the range [317,2208] bytes (see Fig. 3.8). Following the work in [54], in order to parameterize the attack scenario with respect to the network topology, it is considered a number of attackers that is set to 50% of the total hosts of the network.

In order to recreate a real experimental scenario, a typical benign background traffic is generated based on the CIC-IDS 2018 [47]. In particular, the dataset days Wednesday-14-02-2018-TrafficForML-CICFlowMeter, Wednesday-21-02-2018-TrafficForML-CICFlowMeter, Wednesday-28-02-2018-TrafficForML-CICFlowMeter are taken into account. More specifically, the conducted study is based on the analysis of the probability distribution of the interarrival times registered in the benign

flows (more than 1.5 million samples) finding an exponential distribution with a $\lambda = 0.4$. To generate the benign background traffic the Distributed Internet Traffic Generator (D-ITG) generator [55, 56] is employed setting the lambda parameter equal to 0.4; while the packet size is uniformly distributed within a range of [16, 360] bytes.

Finally, since the proposal extends the shortest-path nature of the networks for the sake of security, an additional study is carried out to evaluate how much the traditional short path is affected by the security and cooperative behavior constraints. In other words, the analysis focuses on the *traffic detouring* from the traditional shortest path that is caused by complying with network security constraints. The network topologies used to test the experiments are publicly available at [57].

Three topologies of increasing dimensions are involved to evaluate the scalability of the proposal: the first one with 10 nodes and 25 edges, for which the value of $\#colors$ computed by Eq.3.18 is 3 (i.e., SL is splitted into three WLs); the second one with 25 nodes and 48 edges and a computed $\#colors$ equal to 5 ; and the third, bigger, topology with 30 nodes and 51 edges, for which $\#colors = 7$. The chosen topologies allow to test the scalability degree of the proposal while increasing the model complexity and therefore the amount of WLs that need to be deployed to obey and guarantee the network security coverage. According to [35], these are appropriate SL complexities when dealing with network traffic classifications. However, the proposal is general enough to be extended to more complex models, making it adaptable for other AI-relevant tasks.

The proposal has been implemented using P4-enabled virtual PDP, namely BMv2 [25] that are based on the v1Model architecture. Due to the limited instruction set of the P4 language (it does not support basic operations such as division, exponentiation or logarithm), 43 features of the CIC-IDS 2018 are extracted. The P4 code that implements the models and the associated feature extractor will be publicly made available.[2] The computation of the features is added to the execution of the SL itself, whose complexity depends on the number of WLs that compose it. Implementing the proposed distributed approach mitigates the computational burden on the programmable switch by partitioning the model. This partitioning effectively reduces the overall complexity of the model by proportionally distributing the WLs, which compose the SL, across multiple PDPs. Consequently, this

[2]GitHub repository at [57]

approach diminishes the query load on any single PDP device.

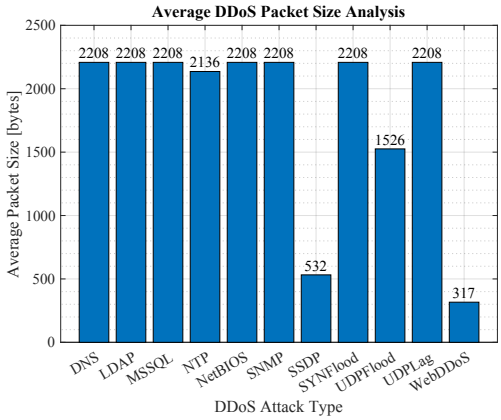


Figure 3.8: Average Packet Size for DDoS attack in CIC-DDoS2019.

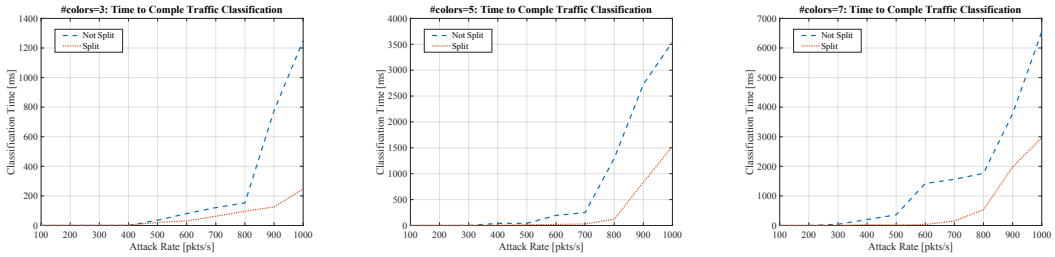


Figure 3.9: Average Classification Time for Experimental Scenarios: a)$\#colors = 3$, b)$\#colors = 5$,$\#colors = 7$.

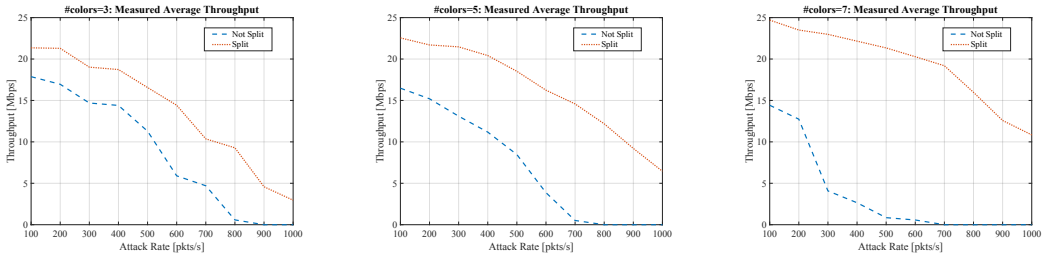


Figure 3.10: Average Throughput for Experimental Scenarios: a)$\#colors = 3$,b)$\#colors = 5$,$\#colors = 7$.

3.5.5 Classification Time Analysis

This subsection aims to analyze the average classification time of the networking devices within the proposed distributed approach, under increasing traffic loads. By dividing a complex model into simpler components, the approach significantly reduces classification time compared to scenarios where network devices run a non-split, complex model. This improvement is particularly evident under heavy traffic conditions (like DDoS attacks), where the distributed approach allows the network to respond more effectively, maintaining performance and reducing delays, due to the reduced overhead imposed on network elements.

In Fig. 3.9a the achievable average classification time under a varying attack rate is shown. With the first small topology (10 switches, 50 hosts of which 25 are attackers) – which requires a SL composed of three WLs to guarantee the security coverage – it can be observed that while the amount of handed packets is around 200-400 pkts/s the SL^{VNF} configuration performs better, showing an average classification time that is about 60% less than the WL^{VNF} (an average of 0.62 ms of the SL^{VNF} against 1.7 ms of the WL^{VNF}). This is due to the additional intermediate communication that happens between the PDPs to get the final classification. However, as the attack rate intensifies and the switches become overwhelmed with network packets to analyze, this advantage diminishes, allowing the WLs^{VNF} configuration to demonstrate its strengths in handling critical attack situations. The differences can be appreciated when the attack rate is in the range of 600-800 pkts/s, with the classification time more than halved. Under heavy attack load, 900-1000 pkts/s, the SL^{VNF} configuration is not able to timely handle the classification tasks, reaching a maximum time to complete classification which is more than 1000 ms against the $\sim$ 200 ms achieved through the adoption of the proposed model splitting and distribution paradigm.

In Fig. 3.9b the results with the medium network topology (25 network switches and 125 hosts – 75 attackers) and a SL composed of five WL^{VNF}. In this case, due to the lesser model complexity, the benefits of the proposal can be appreciated starting from 300-400 pkts/s and it shows its effectiveness around 500-600 pkts/s by reducing the time to complete the classification of more than 90%. Even under the highest attack rate (1000 pkts/s), the reduction achieved by the proposal is more than 50% ($\sim$1500 ms with the proposal against $\sim$3600 ms with the SL^{VNF}

configuration).

This trend is confirmed by the experiments carried out with the largest topology (see Fig. 3.9c), in which the optimization problem suggested an SL with seven WLs to cope with network security coverage. In this case, the highest complexity of the SL^{VNF} leads the network to be unable to timely handle classification tasks starting from an attack rate of 300 pkts/s. At 500 pkts/s the gap starts to be prominent, with an average classification time of ∼360 ms for the SL^{VNF} against 20 ms for the split configuration. When the attack rate is around 1000 pkts/s, the benefits of the proposal are indeed highlighted allowing the network to adapt to the huge attack rate, showing a reduction of 55% in the average classification time.

In light of the considerations made so far, it can be concluded that as the size of the network topology and the load it is subjected to increase, using a split-AI approach to distribute the workload within programmable data planes, allows for an effective integration of complex AI-relevant tasks within the network, but also a scalable and adaptable solution to network changes. These results shed light on the importance of split-AI approaches to cope with the upcoming seamless and tight integration of networking and AI, for future 6G networks.

3.5.6 Throughput Analysis

In a further test campaign the average throughput of the PDPs in both configurations, i.e. SL^{VNF} and WL^{VNF}, is measured by varying the network topologies and the related value of $\#colors$. This is to demonstrate that the proposed approach of optimizing the distribution of active IDS features is scalable in terms of network devices' capacity in managing network traffic.

It is observed in Figs. 3.10 that the WLs^{VNF} deployment setting shows the best gain for the network, both in terms of throughput and delays, with the increase in the amount of traffic generated by the distributed malicious hosts. When considering the SL^{VNF} configuration, the throughput experienced by the network devices decreases as the SL complexity increases (from three to seven WLs), mainly due to the increasing number of WLs that need to be queried on a single PDP. With the simplest SL, the average network throughput starts to drop below 5 Mbps when the attack rate is 700 pkt/s, quickly approaching 0 Mbps at 800 pkt/s. This trend worsens when considering more complex SLs. The $\#colors = 5$ scenario shows

that the network throughput drops to zero when approaching an attack rate of 600-700 pkt/s. Even worse is the case of the most complex SL ($\#colors = 7$), whose overhead causes the average network throughput to approach zero starting from an attack rate in the range of 400-500 pkt/s. In such situations, data plane devices experience substantial degradation in their forwarding capabilities.

However, when the SL is split and distributed across the network, the computational load imposed on the PDP devices is alleviated, making it possible to consider the integration of even complex AI models within the network without affecting the normal network operation too much. In fact, when considering the $\#colors = 3$ scenario and the split configuration, the average network throughput starts to drop below 5 Mpbs with an attack rate of 900-1000 pkt/s.

Considering an attack rate in the range of 100-500 pkts/s an increase in the throughput of 20% on average is registered. Under a higher attack rate, this advantage is further improved ($\sim$50-55%), until the benefits of the distributed approach guarantee that the network is still able to handle traffic at a minimum throughput against the SL^{VNF} in which the network is completely torn down.

The benefits of the proposed approach become more pronounced as the complexity of the SL increases and the size of the network expands. When a SL composed of five WLs is needed to cover the network, the reduced computational burden further preserves the average throughput of the network. Indeed, the average network throughput is in the range [$\sim$6, $\sim$15] Mbps also under the attack rates 700-1000 pkts/s, for which the not split configuration leads to zeroing out the average throughput measured on the PDPs.

Finally, in the $\#colors = 7$ network topology, the complex SL causes significant performance degradation at attack rates starting from 400 packets per second leading to a rapid drop in average throughput toward zero, the proposed workload distribution approach enhances scalability. This method effectively manages computational overhead, enabling the network to handle large attack volumes while maintaining a satisfactory level of throughput.

Nevertheless, a truly zero-cost solution does not yet exist. The execution of models still imposes a measurable impact on network throughput, with an observed average of approximately 35 Mbps when no SL/WLs^{VNF} are active within the switch. This limitation stems from the technological constraints of current networking devices which are not yet inherently designed to fully support the seamless

integration of networking and AI workflows.

However, it is expected that these issues will be resolved in future 6G networks, which will likely incorporate advanced, high-performance chips capable of significantly increasing computational power and minimizing the impact on network performance. Having said that, the advantages of the proposed distributed and split AI approach are clear, making it a viable solution for supporting AI-relevant tasks within current as well as future PDP devices. Finally, it is important to highlight a key feature of the proposed approach: it can effectively operate (without any modification) with both encrypted and unencrypted network traffic, as it relies exclusively on header information, which is always transmitted in plaintext.

3.5.7 Shortest Path Detouring Analysis

To evaluate the impact of the proposed in-network inference model deployment and the associated coloring constraints on network performance, the *AWDelay* metric is introduced. Given a pair of source and target nodes (s, t), let $SP(s, t)$ denote the cost of the classical shortest path between s and t, and equivalently, let $SP_C(s, t)$ denote the cost of the shortest path obtained for the problem with coloring constraints. The weighted average of the delays can be defined as a function of the lengths of the classical shortest paths. Let $delay(s, t)$ be the relative delay between the constrained shortest path and the classical one between source s and target t, i.e.,

$$delay(s,t) := \frac{SP_C(s,t) - SP(s,t)}{SP(s,t)},$$

then the weighted average of delays is defined as follows:

$$AWDelay := \frac{2}{|V| \cdot (|V| - 1)} \cdot \sum_{(i,j) \in E | i<j} \overline{w_{ij}} \cdot delay(i,j),$$

where $\overline{w_{ij}}$ is the normalization of the following weights that depend on the length of the classical shortest paths defined as:

$$w_{ij} := e^{length(SP(i,j))}.$$

Based on this metric, the instances studied in Section 3.5.3 are analyzed. Specifically, the influence of network density and size on path detours is assessed by

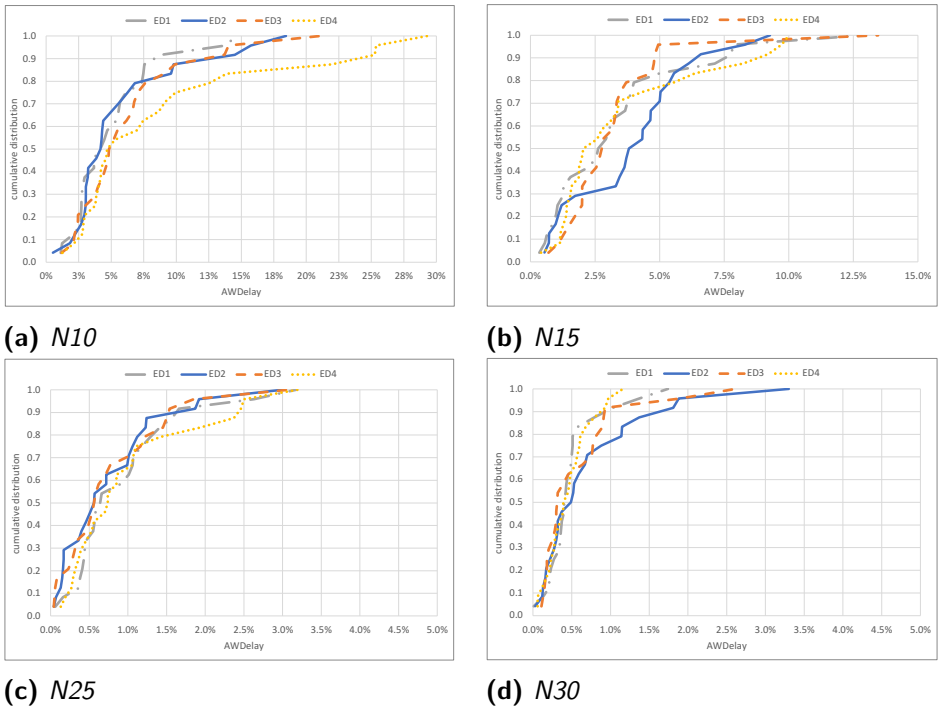


Figure 3.11: Cumulative distribution of *AWDelay* for the density classes for each node class.

examining the average weighted delay. Table 3.4 presents the average *AWDelay* values grouped by the number of nodes N and the density class D. The *AWDelay* values shown for each combination of N and D represent the average computed across all instances discussed in Section 3.5.3. The AVG row reports the average calculated based on the nodes, while the column AVG shows the average relative to the density. Additionally, the row labeled VAR indicates the variance of all *AWDelay* values for each number of nodes N, providing a measure of data dispersion and allowing to assess the variability based on the number of nodes.

Instead, the plots shown in Fig. 3.11 represent the cumulative distribution of *AWDelay* for the density classes for each node class. Thus, each curve shows the cumulative percentage of recorded results that exhibit an *AWDelay* less than or equal to a specific value indicated on the x-axis.

For $N = 10$, a clear upward trend in the curves is observed, where a high percentage of observed values (around 60%) is concentrated within the lower *AWDelay* range (0–5.5%), especially for the first three density classes. On the other hand, the results for *ED4* show generally higher delays, but more spread out over a wider

interval. Specifically, the curves associated with the first three density classes show a rapid accumulation around 5% *AWDelay*, while the *ED4* curve shows a slower accumulation, suggesting a more dispersed distribution of delays, with the presence of paths experiencing higher delays. In this class of nodes, the minimum and maximum *AWDelay* values are 0.52% and 29.30%, respectively, with an overall average of 6.58%.

For $N = 15$, the graph in Fig. 3.11.(b) shows a behavior similar to what was previously observed, but with some significant differences. First of all, for all density classes, 80% of delays are below about 5%. A slight difference is seen in the *ED3* class, where about 95% of the values are concentrated in the lower *AWDelay* range (0–5%). Another difference is that in this class, the trends of the four curves are quite similar. The minimum and maximum *AWDelay* values are 0.34% and 13.45%, respectively, with an overall average of 3.54%.

Compared to the previous plots, the graph with 25 nodes (Fig. 3.11.(c)) shows a more concentrated distribution of *AWDelay* values. All the plots reach 90% of the cumulative distribution at lower *AWDelay* values compared to the previous plots. This indicates that most of the paths in networks with 25 nodes experience lower delays, concentrating below around 2.5% *AWDelay*. Specifically, the curves for the first three density classes show almost identical behavior, with very rapid accumulation (90%) for delays below about 1.5%. The *ED4* curve shows a similar trend, although it has a slightly more gradual increase, suggesting greater variability in delays compared to the other density classes, but still well-contained compared to cases with fewer nodes. The minimum and maximum *AWDelay* values are 0.04% and 3.19%, respectively, with an overall average of 0.88%.

Similarly, in the plots of Fig. 3.11.(d), as previously observed for the instances with 25 nodes, the *AWDelay* values are concentrated within a very narrow range (up to 3.5%). Similar to the previous case, all the curves reach 90% of the cumulative distribution at *AWDelay* values below about 2%. Specifically, the curves representing *ED1*, *ED3*, and *ED4* show almost identical behavior, with a high percentage of observed values (90%) having delays below about 1%. The *ED2* curve shows a similar trend but with a slightly more gradual increase, indicating greater variability in delays compared to the other density classes. The minimum and maximum *AWDelay* values are 0.01% and 3.30%, respectively, with an overall average of 0.57%.

The curves associated with 30 and 25 nodes converge much more quickly compared to those for 10 and 15 nodes. This suggests that, as the number of nodes increases, the effect of network density becomes less pronounced, leading to more similar delay distributions.

The results of the experiments, as shown in the table, indicate that the average weighted delay behaves consistently as the network grows in size. Specifically, *AWDelay* significantly decreases with an increasing number of nodes. For example, in networks with 10 nodes in the density class $ED1$, the average delay reaches around 5%, while for networks with 30 nodes, the delay drops to approximately 0.5%. This trend can also be observed in the average delay, which decreases from 6.58% with 10 nodes to 0.57% with 30 nodes. This indicates that the overhead introduced by the coloring constraints becomes less significant in larger networks, making the approach more scalable and efficient as the network grows.

Interestingly, when varying the density for a fixed number of nodes, except for the case with 10 nodes, the average *AWDelay* remains almost constant. The variance of all *AWDelay* values decrease from 3×10^{-3} for $N = 10$ to 3×10^{-5} for $N = 30$. This behavior is attributed to the fact that as density increases, and consequently, the number of available paths increases, the probability of significant detours from the classic shortest path decreases, thus mitigating any further delay reduction. For example, networks with $N = 30$ and higher density classes (such as $ED4$) consistently show lower *AWDelay* values, supporting the hypothesis that denser networks provide more direct alternative paths even with coloring constraints. The stability of *AWDelay* across different density classes reinforces the robustness of the proposed approach, as the method maintains a consistent balance between security and efficiency without significantly compromising network performance, even in denser topologies.

This trend is further supported by the variability observed in Fig. 3.12, where the box plots illustrate the distribution of *AWDelay* across different densities. In particular, the interquartile ranges expand in sparser networks, showing greater variability in path efficiency due to the limited number of feasible paths that meet the coloring constraints. The box plots also highlight that in more connected networks, such as those with $ED4$, the *AWDelay* distribution is more compact, suggesting a more uniform detour behavior.

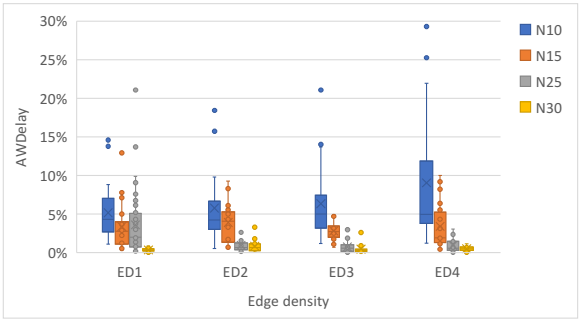


Figure 3.12: Box plots of *AWDelay* for each edge density class.

Table 3.4: *AWDelay* results for density and number of nodes

	N10	*N15*	*N25*	*N30*	AVG
ED1	5.16%	3.37%	0.94%	0.50%	2.50%
ED2	5.77%	3.92%	0.78%	0.73%	2.80%
ED3	6.34%	3.26%	0.79%	0.58%	2.74%
ED4	9.04%	3.60%	1.02%	0.46%	3.53%
AVG	6.58%	3.54%	0.88%	0.57%	
VAR	0.003	0.001	0.0001	0.00003	

3.6 Main Insights and Takeaways

In the research of this thesis, a disruptive paradigm has been introduced aimed at defining a new approach to intrusion detection in which very lightweight learning models, resulting from splitting a strong model, are dynamically distributed and appropriately chained within the switches of a next-generation PDP. The objective is to have future 6G networks natively protected from malicious traffic by the same devices in their data plane, and at the same time to avoid overloading the forwarding functions with heavy learning models implemented in the switches. An initial set of measurements was shown to provide a proof-of-concept of the new paradigm and to highlight its potential. These results shed light on the importance of considering model-splitting techniques such as in the proposed paradigm of "projecting ensemble learning on the network".

Chapter 4 Native support for INC-assisted split-AI in Future 6G Networks

Featuring ultra-high bandwidth, low latency, and pervasive intelligence, the emerging 6G networks are poised to revolutionize numerous applications, from real-time healthcare and autonomous vehicles to immersive entertainment and smart manufacturing [58]. However, unlocking the full potential of these applications depends on the efficient execution of complex tasks such as image recognition, natural language processing, and decision-making.

Traditional approaches, in which computational capacity resides exclusively on end devices or centralized servers, struggle to meet the rigorous latency and resource demands of these emerging applications [59]. To address these challenges, the integration of Artificial Intelligence (AI) within communication networks, known as AI4Net, becomes crucial. AI4Net uses AI and machine learning (ML) techniques to improve network performance. Despite the well-established advantages of AI4Net, its complementary concept, namely communication networks that support artificial intelligence (Net4AI), remains relatively unexplored [28].

Net4AI emphasizes leveraging network infrastructure to facilitate and accelerate AI tasks. A significant breakthrough in this effort is the innovative concept of Split-AI. Split AI involves dividing NNs into segments running on different nodes, such as UEs and base stations or edge devices. This approach overcomes the limitations of traditional methods by distributing computational loads, thus reducing UE power consumption, inference latency, and network traffic overhead. Research has demonstrated Split-AI's effectiveness in improving key performance indicators (KPIs) [43, 60, 61], and standards bodies have recognized its potential [62]. In

parallel, the paradigm of In-Network Computing (INC) [63] is emerging as a cornerstone of 6G networks. INC involves embedding computational capabilities within network elements, enabling data processing along the data path. This complements the Split-AI approach, offering substantial improvements in KPI for end users, network operators, and application providers. As such, INC holds significant promise for enhancing the efficiency and responsiveness of AI-driven applications in the 6G era, facilitating the realization of the full potential of these advanced technologies [2, 64].

In light of these considerations, and building upon the works [4, 3], where the potential and primary benefits of the INCaS-AI (IN-Network Computing assisted Split-AI) approach are investigated, this thesis investigates the synergistic strengths of INC and split-AI. These combined methodologies form the foundation for a comprehensive model geared towards efficient AI deployment within the 6G network framework.

Earlier studies [2] analyzed the architectural prerequisites for implementing INCaS-AI in 6G networks (extending the established 5G terminology), with a focus on User Plane (UP) entities that facilitate AI-oriented computational tasks. Extending this research, the present work further develops both Control Plane (CP) and UP enhancements. These modifications are intended to enable native, seamless support for Split-AI operations within forthcoming 6G infrastructures. To substantiate the proposed model, this thesis also details a Proof-of-Concept simulation for INCaS-AI, accompanied by an initial performance assessment. These findings illustrate the viability and advantages of integrating AI capabilities directly within the network by strategically splitting and distributing NN computational loads 'in-network.'

4.1 Related Works

A brief introduction to the relevant key concepts and their related works is provided below.

4.1.1 In-Network Computing

The idea of shifting *computation into the network* exploiting the new programmability capabilities of the data planes of telecommunication networks is explored in [65].

A recent very interesting and complete survey on the topic is provided in [66]. A key benefit of this application lies in providing orders of magnitude higher throughput and lower latency compared to running them on end hosts, as explored in [67] and [68]. In the literature it is clear how the paradigm strongly exploits recent advances in programmable network devices and the development of network programming languages such as P4 or others described in [69], to gain practical relevance. This paradigm can also be achieved by leveraging network function virtualization and containerization [70]. Additionally, new network programming features, e.g. Express Data Path (XDP) and Enhanced Berkeley Packet Filter (eBPF), go beyond P4 to further increase INC capabilities. A convergence of communication, computing, and caching is addressed in recent works, among which [64] [71], aimed at creating service-aware beyond 5G and 6G solutions, capable of meeting the stringent requirements of novel applications.

4.1.2 Split-AI

According to the Split-AI paradigm, portions of a complex ML model are executed sequentially in different compute nodes. For example, the UE could execute a first part of the NN model and forward the intermediate results to an edge server, which computes the rest. Such a collaborative computation of an NN model can be beneficial in terms of end-to-end latency, UE energy consumption, generated network traffic load, or compute pressure on the application server. Due to the different characteristics of the NN layers (e.g. in terms of complexity or the generated output data size) and due to dynamic network conditions, it is complex to find the optimal point where to split the NN. The Neurosurgeon approach [43] defines an optimization problem, so to find a collaborative NN execution between UE and server that either minimizes the latency, the energy consumption, or the data center throughput. Similarly, [44] finds an optimal split reducing the inference time by solving an optimization problem that considers relevant constraints, such as the compute capabilities of the compute nodes, the available network bandwidth, or the required accuracy needed by the application.

4.1.3 INC-assisted Split-AI

Previous works on Split-AI are limited to splitting the execution of the NN between a UE and an application server. Progressing this and leveraging INC for Split-AI in 6G could mean that the NN is split into several (more than two) parts, and execution is distributed among several compute nodes, including 6G User Plane (UP) entities. That is, ANs, UPFs, UE, and the application server collaboratively compute the NN. A first work on such an approach is presented in [61]. On the example of a Blind Source Separation (BSS) problem, the authors show, by means of a proof-of-concept implementation, the feasibility of executing parts of a NN in the 6G UP. In [4], the term INCaS-AI has been coined and introduced. The main functional and structural modifications envisioned – as a consequence of the vision introduced with the concept of INCaS-AI – for the upcoming 6G networks Control and User Plane, to natively accommodate such AI-relevant workflows, will be detailed in Section 4.2.

4.2 Intelligent User Plane as a Split-AI Enabler

The concept of the 6G IUP is exploited for realizing Split-AI natively in 6G networks, as described in more detail in the following section. In the following, its the generic key enablers are briefly introduced and subsequently mapped to the specific case of Split-AI.

4.2.1 Overview on the 6G IUP and its Key Enablers

The following key enablers are required to implement the 6G IUP, providing INC capability in 6G networks [2]. A CUPE refers to a User Plane (UP) entity, i.e. AN* and UPF*[1], capable of carrying out computations, besides packet forwarding. To achieve this, it needs significant enhancements in processing hardware and computational resources such as CPU, RAM, GPU, storage, etc. A Compute Service (CS) unites all computational instructions and precisely describes the actions

[1]The 6G counterparts of AN and UPF are referred as AN* and UPF* since names for 6G are not yet defined.

to be executed on a flow's packets and can be seen as *one unit of computation* to be performed at a CUPE. A CCCE, responsible for any 6G INC-related controlling task such as defining which CUPEs to be used (UP path selection) and which CSs therein, and defining the CC Profile (a counterpart to the QoS profile, providing link and compute requirements). Conceptually, the CCCE is not necessarily a single entity, but it can be composed of several, logically separated Network Functions (NFs), e.g., in 5G terms, the CCCE could be jointly realized by Application Function (AF) and Session Management Function (SMF). A Communication and Compute Flow (CC Flow) can be seen as an evolution of QoS Flows, enhanced by per-flow level computation management. It is an application flow whose packets are modified along the UP path, by specifying and parameterizing the CSs to apply.

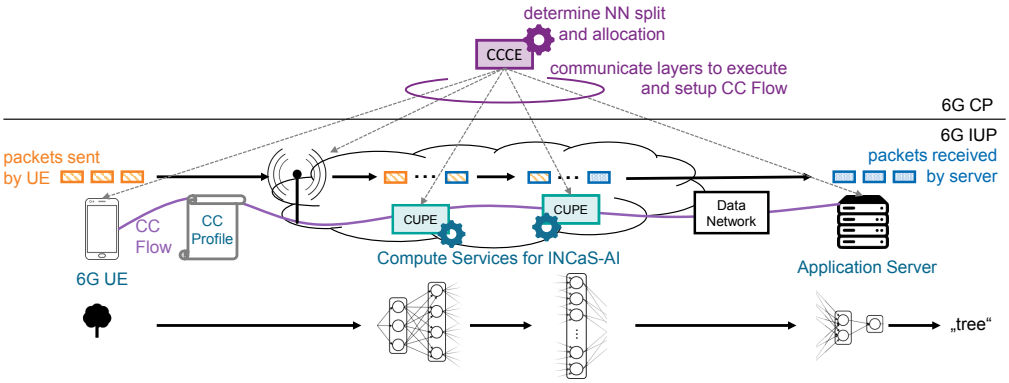


Figure 4.1: Reference 6G IUP to enable INCAS-AI [2].

4.2.2 Using the 6G IUP Key Enablers for INCaS-AI

An overview of the reference vision for the research conducted in this thesis is provided in Figure 4.1 [2]. The CCCE is located in the 6G CP. It determines how to split the NN and how to allocate the layers to computing nodes. During CC Flow setup, the CCCE communicates to the 6G IUP the layers to execute. After successful CC Flow establishment, the 6G UE is connected via the AN and via two 6G CUPEs to the Data Network (DN), where the application server resides. Through processing the CC Flow's packets during transmission (i.e., NN layer computation in the IUP), the packets initially sent by the UE differ from the packets received by the application server (as indicated by different colors). Figure 4.1 only shows a single example, where the two CUPEs executing NN layers are the 6G counterpart

of a 5G UPF. However, the AN could also act as a CUPE and execute NN layers, with the UE also capable of locally computing parts of the NN. Table 4.1 shows how the generic 6G IUP concepts [2] can be specifically used for Split-AI.

Table 4.1: Mapping of generic concepts proposed in [2] to the case of Split-AI.

Concept	*Specific INCaS-AI Case*	*Description*
CC Flow	A network flow connecting to the endpoints of an AI application, which uses an NN for its inference task, supporting computation on top of communications / CC Flow relating to an AI application.	Parts of the NN (i.e., some layers) or even the complete NN are executed at the CUPEs besides transmitting the data.
CS	A computation unit, related to the execution of NN layers. Several options for CS realizations are possible: **Option 1:** One CS unites all layers to be computed at one CUPE. **Option 2:** One CS unites all computations for a single NN layer to compute. **Option 3:** One CS represents the computation of only a single neuron.	Depending on the realization, there are (a) different granularities of the computation (fine-grained on a per-neuron basis or coarse-grained on a set of layers), and (b) the CUPE must be capable of supporting i) the execution of more than one CS (in case one CS is one neuron) or the ii) dynamic construction of a CS (in case a CS composes several layers)
CCCE	It is responsible for (i) splitting the NN, (ii) determining the UP path, and (iii) allocating layers to compute nodes. When determining the split and the allocation, different factors are involved, such as AI application requirements (e.g., inference speed), compute node capabilities (CUPEs, UE, AN), the current network load (in terms of computation and communication).	Optimally splitting and allocating the NN is not trivial. The UP path selection, possible split points of the NN, and the possible allocations result in several degrees of freedom and a high number of possible combinations. As compared to the capabilities of a 5G SMF, the CCCE would need more information available (through enhanced signaling/interaction with other NFs) as well as enhanced compute capabilities.
CC Profile	Set of information associated to each CC Flow. It includes all Split-AI-relevant computation information, which must be known at the CUPEs.	Link and computation requirements at each CUPE should be included. Depending on how the CCCE signals the layers to be executed to the CUPEs, it may also contain (parts of) the NN layer allocation.
CUPE	6G UP entity, specifically supporting CSs for the execution of one or more NN layers	Significant speed-up can be expected [72] if CUPEs are equipped with specific hardware that is dedicated for NN inference.

6G CN CP performs the coordinated allocation of communication and compu-

tational resources at each service session based on the requirements issued by the application layer. This allocation will be based on a trade-off between the required CC Profile and the availability of communication resources (radio links, transport network) and compute resources (e.g., CPU, memory, or any type of accelerators such as GPUs, ASICs, and FPGA in the network, at the mobile network operator's edge, or in the end-device). The 6G CN CP needs to consider several factors, such as the availability of computational resources and the set of supported CSs at different CUPEs, for determining the appropriate UP components to perform Split-AI tasks in the envisioned communication and compute architecture for 6G.

Benefits of INCaS-AI include supporting low-compute, limited-battery devices, minimizing overall latency with dedicated hardware, and reducing compute pressure at the application server while optimizing the overall energy consumption. Also, application providers can use adaptive partitioning to delegate computation-intensive tasks to high-performance INC resources with hardware accelerators. This improves application performance, reducing latency and improving inference time for AI-based applications. For example, 3D video rendering can be executed in dedicated INC resources, while NN layers can be executed in INC resources equipped with hardware accelerators.

4.3 Challenges & Requirements for INCaS-AI

This section elaborates on the key requirements for the 6G architecture as well as on the most important challenges when it comes to realizing INCaS-AI presented in [4].

4.3.1 Key Challenges for INCaS-AI in 6G

Challenge 1: Varied and potentially conflicting KPIs. The combination of mobile network with a complex NNs, results in a large number of performance metrics that could be improved or optimized by INCaS-AI; some could be in conflict. This includes, e.g., minimizing the UE energy consumption, minimizing the overall energy consumption, reducing the generated traffic volume, or keeping the end-to-end latency of the application as low as possible. Further, to save on compute resources and traffic volume (i.e., if capped by the user's subscription plan), it could

be essential to minimize the compute load in the network and minimize the traffic that would be billed to the user. The KPIs to optimize might even relate to various possible stakeholders in the network (e.g., user, operator, and the application). The question is, hence, *which of the many possible optimization targets to consider*, and consequently which entity (user, application, network) should be prioritized to benefit from Split-AI. Often not a single but a set of KPIs are optimized and hence, the question would be *which KPIs to prioritize over others?* As mentioned earlier, some of the optimization goals may even be *conflicting* with each other. For example, if the UE's energy consumption should be minimized, an obvious approach would be to offload as much of the compute load as possible to the network or the application server. This would therefore reduce the UE energy consumption with the cost of increasing traffic and compute load in the network. Evidently, this trade-off must be carefully considered when designing the end-to-end system.

Challenge 2: Complexity of finding the optimal split. Even in the simplest case, where one single optimization target is set (e.g. minimize end-to-end inference latency), it is still challenging to find the optimal NN split and the respective compute node allocation. This is due to the degrees of freedom (DoF) when it comes to allocating NN layers to compute nodes: i) the number of splits and the split positions, ii) the selection of UP entities (UP path selection), iii) the allocation of NN layers to compute nodes (UE, server, CUPEs). Considering a simplified case, with the UP path being short and already set: UE-AN-UPF-Server, i.e., DoF ii) is eliminated. Further, consider a very simple NN with only three layers. This creates four different split options (no split at all, 2 options for one split, and the option of splitting at each layer). This simplified consideration already results in 17 different options for allocating the NN layers to the four compute nodes (UE, AN, UPF, server)[2]. The inclusion of just one alternative UPF that could be used as a CUPE for NN execution, already leads to the double amount, i.e., 34 possibilities.

Challenge 3: Large amount of information to be collected and processed. Not only is the complexity of determining the optimal split a challenge, but also the fact that it requires a huge amount of information to solve the optimization problem. The optimal split and allocation depend on a multitude of factors, related to the end user, the application workload (the NN), as well as the network. The

[2]Please note that it has been considered for the example a flow from UE to server as shown in Figure 4.1. The NN layers must be executed in order.

Table 4.2: Novel 6G Capabilities and Signaling Information Required for INCAS-AI [4].

	Required Capability	*Needed Information Exchanged*
6G CP	- Determine number and locations of splits of the NN - Determine UP path and NN layer allocation - Charging (user/App provider to pay for INC service)	**Application-related**: Requirements (e.g. inference time), per-layer info (complexity, output size) as it determines the processing latency and traffic generated between compute nodes) **User-/UE-related**: Privacy requirements (e.g. user consent to share info with network), energy-related info (e.g. current battery level), user preferences (e.g. minimize INC usage as much as possible to save monthly cap limit according to subscription plan) **UP-related**: Communication-related information (UL/DL volume, delay), computation-related information (static, e.g., available CPU/GPU and dynamic, e.g., current CPU/GPU utilization)
CP ↕ UP	**Efficient signaling**: CP → UP: Enforcement of NN layers to be executed UP → CP: Various monitoring information	**CS (NN-layer info:)** Activation Functions, weights, biases (possibly compressed NN model) Provide UP-related information needed at the CP on a given time-granularity
6G UP	- Execution of NN layers as an INC CS - Support required level of dynamicity	Compute requirements (e.g. the CUPE's time budget for executing the assigned NN layers, provided by the CP via the CC Profile)

first line, far right column of Table 4.2 denotes the key information[3] required in the 6G CP, such that the CCCE can determine an optimal split. The information refers to all involved stakeholders of INCaS-AI: application, user/UE, and network (the 6G UP in this case).

Challenge 4: Higher level of signaling overhead. It is expected that INCaS-AI operations will generate additional signaling overhead, given that the communication stack and computing stack should come together. All information denoted in Table 4.2 (third column) needs to be present at the CCCE, which is assumed to be residing in the 6G CP. The split and allocation decision needs to be propagated from there to the entities in charge. Enhanced interaction is required between the 6G system and the application provider, e.g., via the AF and NEF (Network Exposure Function). The two most important enablers to mention here are *(i) signaling for providing the NN-related metadata information* from the application provider to the 6G system and *(ii) signaling for providing the information on how the NN layers are scheduled to the different execution nodes, e.g. to the network and the application*, as the application and the network need to know which parts they have

[3]This list is not exhaustive. The required information can depend on the optimization target, operator-specific settings, and many more factors.

to execute. Further, enhanced interaction between CP and UP is needed. This refers to the collection of monitoring information (e.g. current UL/DL volume or compute load of the CUPEs), as well as control messages, to inform the CUPEs about the layers they need to execute.

One important aspect in the context of scheduling and the associated signaling complexity relates to the scalability of the whole system. Since NN models can be huge, efficient mechanisms are needed to efficiently handle the representation of a whole or partial NN within the 6G system. During CC Flow setup, each CUPE is informed about the layers it needs to execute. High dynamicity is required, because the layers to execute can be different for each flow. Thus, the signaling overhead would grow exponentially if for each CC Flow setup huge NN models must be shared between CP and UP.

Challenge 5: Business relations and trust between multiple systems. Large volume of data, and the associated learning, are related to the applications. Given the data being as asset for the application providers, as well as users' privacy policies, opening the NN model over the network infrastructure, with different business owner, is not a done deal. Despite the willingness of mobile operators to host storage or compute for applications at their access network, over the past two decades, such services have not become a reality. Although recent years have shown a great deal of convergence between different involved sectors, there is still no clear model for such business relationships.

4.3.2 Key Requirements for 6G to Enable INCaS-AI

From the challenges denoted above, consider again Table 4.2 to derive the required enhancements needed in the 6G architecture (specifically UP and CP), so to practically realize the proposed INCaS-AI framework. In general, it can be stated that *6G networks must be significantly enhanced* to support this increased level of complexity for Net4AI (and the specific case of Split-AI). In the following, a summary of what 6G should at least provide:

- Support for INC with nodes reporting rich information about their capabilities e.g. hardware, free resources per time unit, location, capacity, support for sub-NN processing (software);

- Support for fast analytics to collect vast amounts of information about the whole network;
- Fast optimization engines that can harness KPIs defined by the network, UE, and application and resolve trade-offs between conflicting targets, and derive optimal solutions even for complex problems;
- Enhancements of signaling to enable exchange and collection of new information at the CCCE, so to *determine* the optimal split. Further, 6G should provide new information spreading techniques from the CCCE to the UP, so to *enforce* specific execution, i.e. instruct the CUPEs and the application (provider) which layers to execute. It must provide means to share such information as *efficiently* as possible (please note that NN models can become very large in size);
- Novel solutions for exchanging and representing NN-related information in the 6G system (e.g. info sharing between application and 6G system to share the NN to execute for a specific application);
- An enhanced session management entity (e.g. an evolved 5G SMF), which integrates the CCCE, and which is equipped with the necessary logic and sufficient computing power to be able to solve the complex problem of determining the proper split locations of the NN and to determine a proper allocation of the resulting partial NNs (i.e., the NN layers) to compute nodes for execution;
- Targeted interaction between the application (provider) and the 6G system to enable the collaborative computation of NNs between the application and the network.

4.4 CP and UP Enhancements to Support Split-AI

The section presents two key challenges for realizing Split-AI in 6G and how they can be addressed by enhancing the CP and UP.

4.4.1 Key Challenges for Split-AI in 6G

The way the NN is split (number and location of splits) and how the resulting partial NNs are allocated to the compute nodes have a huge impact on associated KPIs. Finding an optimal split and allocation is not trivial. First, there are many possible optimization targets, including inference duration, generated network traffic load, or UE energy consumption. Further, the optimal split depends on various dynamic conditions (such as the UL/DL volume in the UP or the compute load of CUPEs), as well as on underlying system capabilities, such as the compute capabilities of the CUPEs or of the UE. Consequently, a first challenge to address is: *How can the 6G system decide on the proper NN split and optimal allocation of resulting partial NNs – depending on (i) dynamic underlying conditions, (ii) system capabilities, and (iii) network operator preferences?* A further challenge is the immense amount of possible combinations for splitting the NN and allocating its resulting partial NNs to compute nodes. The specific computation a CUPE has to carry out for the split-AI operation can be different *with every single flow* traversing that CUPE. Additionally, a once determined, optimal allocation can change during a flow's lifetime, requiring a re-allocation of the partial NN computation (i.e. re-allocating the partial NN execution from a heavy-loaded CUPE to another compute node). Hence, a high level of flexibility and dynamicity is needed, making static configuration of the CUPEs unfeasible. This leads to the second challenge: *How to enable the 6G UP entities for NN layer execution, providing the required high level of flexibility and dynamicity*?

4.4.2 Enhancements in the Control Plane

In the following, an elaboration on CP enhancements is provided, focusing on how the 6G system can determine the optimal NN split and effectively allocate partial NNs.It is important to note that the enhancements are assumed to pertain to the control plane, as the configuration of the user plane regarding flow handling involves actions managed by the control plane in 5G. Therefore, it is reasonable to assume that this will remain unchanged in 6G. In the reference architecture (see [2, 4] and Figure 4.1) it is assumed that the CCCE is the CP entity in the 6G system that has to i) determine the split of the NN and the allocation of the resulting partial NNs to compute nodes; as well as to ii) instruct compute nodes about the NN layers to

execute, including triggering the NN layer activation in the UP. Determining the optimal split requires various information and a sophisticated logic behind, as it will be subsequently discussed.

4.4.2.1 Determining the Optimal Split - Required Information

In order to determine and implement the optimal split, the CCCE needs information about the network topology and compute capabilities of the CUPEs, which would be provided by the Management Plane (MP). Further, it needs the NN and its characteristics, which would be provided by the third-party application provider, e.g., via an Application Function (AF), and information related to UE's status and capabilities as well as networking and computing loads.

4.4.2.2 Determining the Optimal Split: Underlying Logic

With the above described information provided, the CCCE would be capable of determining how to optimally i) split the NN and ii) allocate the resulting NN layers to compute nodes. Finding the optimal split can be done in a similar way as in prior arts, e.g. [43, 44]. Both works take into account similar information as described above (capabilities of the compute nodes, available bandwidth, output data size after each layer) and find an optimal split and allocation by solving an optimization problem that optimizes a single KPI. To allow for multiple KPIs optimization in Split-AI operations, it is considered the CCCE to hold a set of Split-AI rules, which can be dynamically chosen prior to CC Flow setup (for a detailed CC Flow description please refer to [4]). A Split-AI policy is envisaged to define how to select the Split-AI rule to apply for a given CC Flow. This policy can be programmed via the MP by the network operator. For instance, the policy allows to define how to select the rules according to the operator's preferences and based on underlying conditions and capabilities.

4.4.3 Enabling NN Execution in User Plane

The remainder of the Section describes two distinct options for enabling the 6G UP for NN execution, with the required high level of flexibility being ensured in parallel.

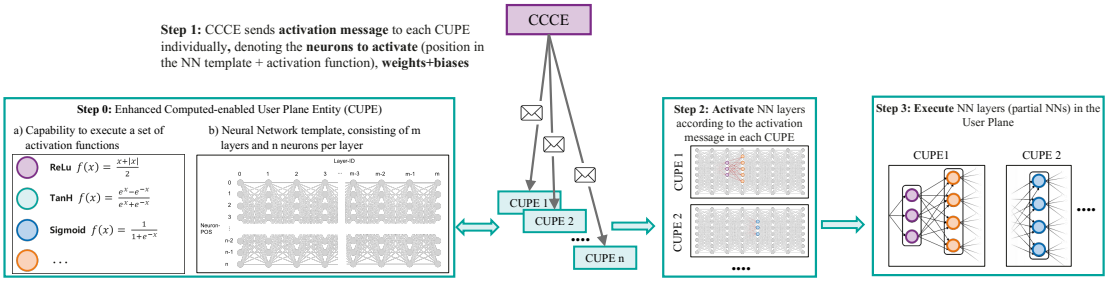


Figure 4.2: Workflow for activating the NN layer(s) to be executed in a CUPE [3]. Each CUPE is equipped with a NN template and the capability of executing a set of activation functions. A message received from the CCCE triggers the activation of the respective partial NN. After that, the CUPE is ready to execute the partial NN it has been allocated.

4.4.3.1 Option 1 - CUPEs with High Capability

This first option assumes that the CUPEs are enhanced in terms of their capabilities. The ultimate goal is to reduce as much as possible the signaling from CP to UP, when instructing each CUPE on what to execute for the Split-AI operations. Figure 4.2 illustrates how the NN activation can be realized with this first option.

a) Availability of a set of activation functions: The neurons of a NN compute activation functions, that transform the multiple input values and the bias to one output value. Some examples of well-known and typically used activation functions are tanh (hyperbolic tangent), ReLU (rectified linear unit), sigmoid, softmax, and binary step. Each of these activation functions is associated with a computation formula, e.g., $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ in the case of tanh, or $f(x) = \frac{1}{1+e^{-x}}$ for sigmoid. To reduce the signaling load from CP to UP, it is assumed that *each CUPE is capable of executing a set of activation functions*. That is, the CUPE is aware of what to execute (i.e. which computes the formula), just by an indication of a reference to the activation function (e.g. its name). This means that the CUPE stores the compute formula, e.g. realized as a compute service, and the associated name. This is indicated as step 0, enhancement a) in Figure 4.2.

b) Availability of an NN activation template: This enhancement assumes that each CUPE is equipped with a template, e.g., as denoted in step 0, enhancement b) in Figure 4.2. It supports a sufficiently large, maximum number of **m** NN layers (along the x-axis), as well as a sufficiently large number of **n** neurons per layer (along the y-axis). The NN template is fully connected (each neuron of layer n connects to each neuron of layer n+1), thus providing the highest flexibility when

it comes to dynamic NN composition. Please note that the neurons are "blank", meaning that the neurons themselves and their connections do exist, but it is not specified i) whether they are used, and ii) when they are, which activation function they compute. Each neuron can be uniquely addressed by a combination of the layer-ID and the neuron-POS.

c) Capability of dynamically composing NN layers to be executed, using the available set of activation functions and the NN template: With the enhancement a) and b), the CUPEs are capable of dynamically composing any arbitrary partial NN (a set of consecutive NN layers), upon reception of an activation message (step 1 in Figure 4.2) from the CCCE. This activation message is unique for each CUPE and contains i) the neurons to activate (by indicating the position in the NN template), ii) for each neuron the activation function to compute (by indicating the name of the activation function), and iii) the weights and biases as input to the neurons. This activation message is sent to each CUPE in charge of executing a part of the NN, and each of the involved CUPEs dynamically activate their respective partial NN (step 2 in Figure 4.2).

Finally, as denoted in step 3 of Figure 4.2, after activation, each CUPE is capable of executing the partial NN it has been allocated for execution by the CP. Please note that with this option, the signaling overhead may be reduced, as the program code to execute is available already at each CUPE and does not need to be signaled.

4.4.3.2 Option 2 - CUPEs with Low Capability

As pointed out in section 4.1.3, the CUPE may be logically composed by a 3GPP network UP function (like for instance a UPF or the CU-UP of a Radio Access node) and a compute instance which is part of a distributed Platform as a Service (PaaS). The PaaS constitutes the Compute Plane, logically separated from the 3GPP User Plane. This approach does not prevent that the compute resources are physically co-located and efficiently interacting with the networking devices, exploiting the benefits of in-network integration of the compute capabilities. An exemplary implementation of this approach is described in [73] as ISAP (In-Network Service Acceleration Platform).

A benefit of this approach is that the Compute Plane may span beyond the 3GPP System domain: PaaS instances may be logically associated with 3GPP User Plane

functions as well as with cloud points of presence outside the 3GPP scope: for instance, the PaaS instances may be realized within the telco edge cloud or centralized data centers or embedded within the end-user device. Moreover, each PaaS instance of the Compute Plane may offer generic compute capabilities (capacity, memory, and acceleration capabilities) that can be allocated to any application workload: this approach makes it possible to exploit the distributed PaaS for a wide range of applications, not limited to the specific split-AI use case in the scope of the current thesis.

The logical separation of 3GPP UP and Compute Plane implies that each Plane will have its own controller/management functions: the 3GPP UP is controlled by the 3GPP CP, while the Compute Plane would be controlled/managed by a PaaS Controller. The main challenge of this approach is the coordination between the 3GPP CP and the PaaS Controller, which should be realized by a standardized reference point to enable multi-vendor interoperability. A key challenge of this architectural approach is to determine which Plane should receive requirements issued by the application layer and, consequently, coordinate the allocation of network and compute resources. This coordination may be done either by the 3GPP Control Plane or by the PaaS Controller. In either case, the coordinating controller/Plane should interact with the other Plane to collect run-time resource metrics and seize resources. This aspect requires further study.

4.5 Simulation Model and Neural Network Configuration

This section describes the Proof-of-Concept simulation model developed to showcase the benefits of the proposed INCaS-AI for seamless integration of future-generation networks and AI-related capabilities.

4.5.1 Simulation Setup

The performance evaluation experiments were conducted using the Graphical Network Simulator Version 3 (GNS3)[4], in which each network device (CUPE) was

[4]https://www.gns3.com/

implemented via a virtual machine (VM). Each CUPE was configured with 4098 MB of RAM and 2 CPU cores. The VGG-16 convolutional network was trained using Python3 with the TensorFlow and Keras libraries on the CUB-200-2011 [74] dataset, which consists of nearly 12,000 images representing 200 bird species.

The simulations were run on a system equipped with an Intel® Core™ i9-9980HK CPU at 2.40 GHz and 32 GB of RAM. During the experiment, 100 images of birds, corresponding to classes recognizable by the DNN, were sent at intervals of one second.

Regarding the network characteristics, typical conditions in an average network are considered, analogously to what has been specified in [61]. As a result, data is transmitted via UDP and the bandwidth between CUPEs is set to 1 Gb/s with a propagation delay of 10 ms. Finally, the UE is modeled as one of the latest and most powerful smartphones commercially available, namely the Samsung S20.

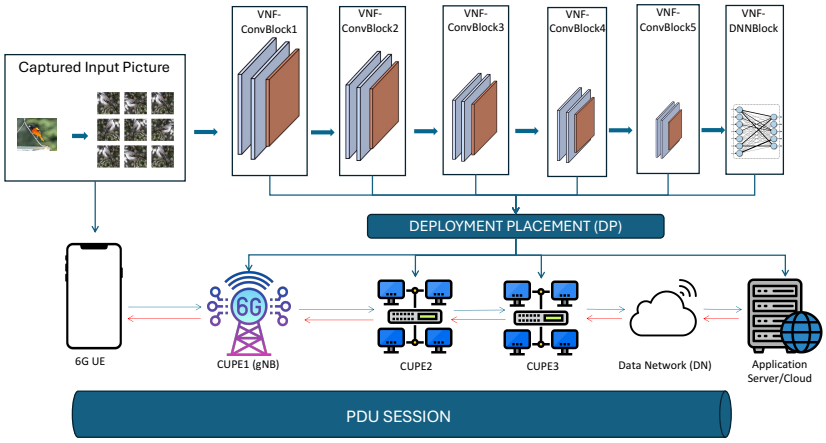


Figure 4.3: Experimental Setup Illustration.

4.5.2 VGG-16: Neural Network Characteristics

In the envisaged 6G architectural design natively supporting the Split-AI framework, it has been considered and evaluated the functional split of the VGG-16 depicted in Figure 4.4.

To aid the comprehension of the proposed split, the VGG-16 structure is briefly mentioned in the following:

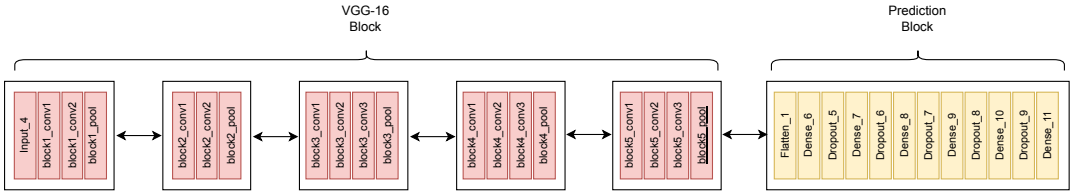


Figure 4.4: VGG-16 Structural Characteristics.

- Five Convolutional Blocks (ConvBlock1-5). Designed to capture features from the image, such as textures, edges, and color gradients as well as higher-level features such as more complex shapes, patterns, or combinations of colors. Each ConvBlock is composed of two convolutional layers and a max-pooling layer to apply a spatial dimension reduction helping to focus on the most essential information for classification.
- The DNN Prediction Block (DNNBlock). It is represented by DNN that uses the features extracted by the VGG16 ConvBlocks to obtain the image classification. The network is composed of 10 hidden layers that use the ReLu activation function. The output layer uses the softmax to assign the class probability.

Following the structure of the VGG-16, the convolutional blocks are isolated from the DNN part intended to compute the classification based on the features extracted by the convolutional blocks.

4.5.3 Split-AI Specific Information

Building upon this splitting strategy, a set of placement configurations for the aforementioned NN blocks is defined, adhering to key principles for the functional split of an NN within the network, as envisioned for future 6G networks [61]. To avoid uncontrolled data expansion due to convolutional blocks, each VNF-ConvBlock is designed to end with a Max Pooling layer. This will drastically reduce the data crossing the network while maintaining the "Blocks" organization of the VGG-16 architecture. In addition, according to [61], each block results in a proportional split. This means that each unit of the decomposed NN has a memory footprint limited to the unit itself. This is paramount since it is not possible to deploy an entire complex NN, composed of millions of parameters and dozens of layers, without negatively affecting its basic functionalities (e.g. data forwarding). The effectiveness

Table 4.3: Performance Evaluation of the proposed VGG-16 split VNF-Blocks.

Metric	VNF-ConvBlock1	VNF-ConvBlock2	VNF-ConvBlock3	VNF-ConvBlock4	VNF-ConvBlock5	VNF-DNNBlock
CPU	17,79%	25%	25%	22,23%	13,95%	12,17%
RAM	23,08%	24,92%	22,72%	22,95%	24,46%	43,50%
FLOPs	$2,2481E^{+11}$	$7,11E^{+11}$	$2,368E^{+12}$	$4,727E^{+12}$	$1,422E^{+12}$	$2,39E^{+8}$
Params(num.-%.)	38592 (=0,028%)	221184 (=0,165%)	1471560 (=1,09%)	5898240 (=4,40%)	7077888 (=5,28%)	119352397 (89,03%)

of this splitting strategy is assessed via performance evaluation tests to measure each block's overhead in terms of Input/Output Data Relation, CPU, and RAM Utilization. Figure 4.5 shows the amount of data that each block requires as input and that produces as output while performing its dedicated operations. These results show that the first convolutional layer causes a huge expansion of the received raw input data. This information is of paramount importance since it allows to understand that this layer should be carefully handled when devising the placement configurations. CPU and RAM consumption values are reported in Table 4.3.

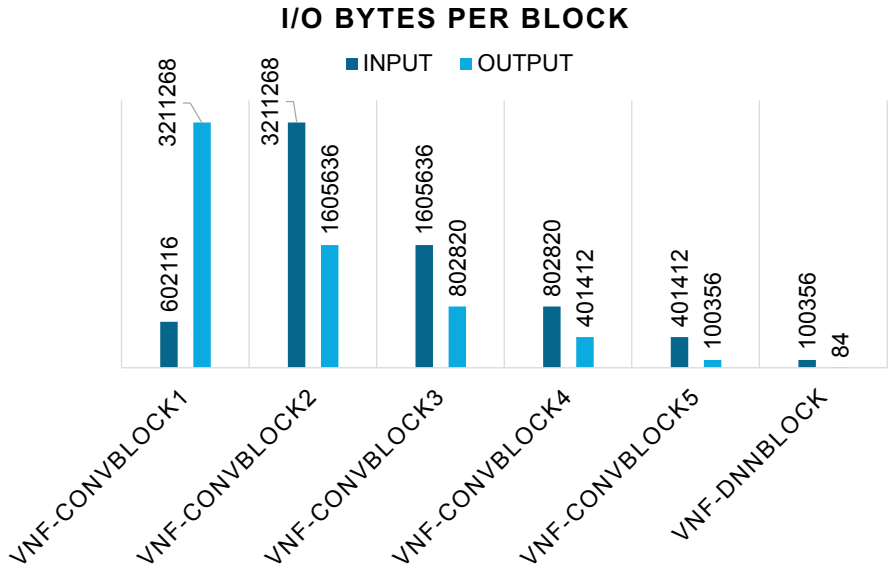


Figure 4.5: Input&Output Relationship per each block.

As for CPU consumption, the highest values are associated with VNF-ConvBlock2, VNF-ConvBlock3, and VNF-ConvBlock4. While for VNF-ConvBlock2 the reason is due to the high amount of data to process, that VNF-ConvBlock1 produces and sends to it, for VNF-ConvBlock3 and VNF-ConvBlock4 the reason is given by the

combined influence of data amount and the number of Floating-Point Operations per Second (FLOPS) performed to carry out the task. By measuring also the FLOPs (third row of Table 4.3), it is observed on VNF-ConvBlock3 the combined action of a high amount of processed data and the high number of FLOPs, while for VNF-ConvBlock4, instead, the highest number of FLOPs than the other NN blocks; those are the reasons for the observed increase in their CPU consumption.

As for the RAM, the one consumed by the convolutional blocks does not exceed 25%. While a higher amount of RAM is consumed by the VNF-DNNBlock. This does not depend on the amount of data, but rather on the fact that the DNN accounts for 90% of the parameters (weights and biases) that composed the entire VGG-16 (fourth row of Table 4.3).

4.6 Performance Evaluations for Split-AI in the 6G User Plane

This section is aimed at showing the results of the experimental campaign conducted by means of the designed PoC simulation model proving the benefits of the envisaged INC-assisted split-AI within the context of future 6G Networks.

4.6.1 Evaluations: User Plane Performance

After characterizing the expected overhead of the VGG-16 blocks, different alternative placement configurations can be tested. Without losing generality, three CUPEs (as illustrated in Figure 4.3) in the path towards the Cloud have been considered, i.e. the typical range of values for the UPF VNFs (in the considered case CUPEs) in an operator's network from a UE to the Data Network, as stated in [61]. The evaluated deployment configurations are chosen considering not only the usual approach (considering UE and offloading to the Cloud) but also leveraging INC principles. These Deployment Placement (DP) are described in Table 4.4.

Under these placement configurations, the inference delay (time to predict), the energy consumption of involved CUPEs and Cloud, as well as the network utilization overhead due to additional data exchanges to carry out split-AI related tasks, are evaluated. Figure 4.6 shows the inference delay guaranteed by each partition. The

Table 4.4: Deployment Placement Configurations.

Partition ID (DP)	Type of Split	Blocks in UE	Blocks in CUPE1	Blocks in CUPE2	Blocks in CUPE3	Blocks in Cloud
1	Full-Cloud	-	-	-	-	1,2,3,4,5,6
2	INC-Cloud(1)	-	1	-	-	2,3,4,5,6
3	INC-Cloud(2)	-	1	2	-	3,4,5,6
4	INC-Cloud(3)	-	1	2	3	4,5,6
5	Full-INC(1)	-	1,2	3,4	5,6	-
6	Full-INC(2)	-	1	2,3,4	5,6	-
7	Full-INC(3)	-	1,2,3	4,5,6	-	-
8	Full-UE	1,2,3,4,5,6	-	-	-	-
9	UE-Cloud	1,2	-	-	-	3,4,5,6
10	UE-INC-Cloud	1,2	3,4	-	-	5,6
11	UE-INC	1,2	3,4	5	6	-

latter represents the time required to achieve inference at the UE, under a specific NN partition configuration. The first four configurations – DP1, DP2, DP3, and DP4 – show a delay more than five times greater than DP5, DP6, and DP7, due to the involvement of the Cloud in the inference process. The last three deployments – namely DP9, DP10, and DP11 – taking advantage of a complete In-Network configuration, do not need to reach the Cloud to finalize the prediction and therefore the overall calculation is finished earlier in the path, thus reducing the uplink delay encountered. Since the uplink delay is the component that most influences the total time to obtain the results (as data traveling in the uplink direction is influenced both by the calculation delay in the CUPE and by the greater number of bytes to be transferred), it becomes evident how advantageous could be a wiser split that takes advantage of INC potential.

4.6.1.1 Placement Deployment involving the Cloud

Let's focus on the partition involving the Cloud - DP1, DP2, DP3, and DP4 - in Figure 4.7. The main characteristic of all four configurations that adopt this deployment is the notable delay in obtaining the inference, while obviously in this configuration the CPU execution delay is less thanks to the cloud computing resources. Therefore, an application or use case with latency requirements should avoid this type of placement. At approximately equal latency, the performance evaluation metric that could be used to compare between the four partitions includes the power consumed by hardware elements in different configurations and network utilization.

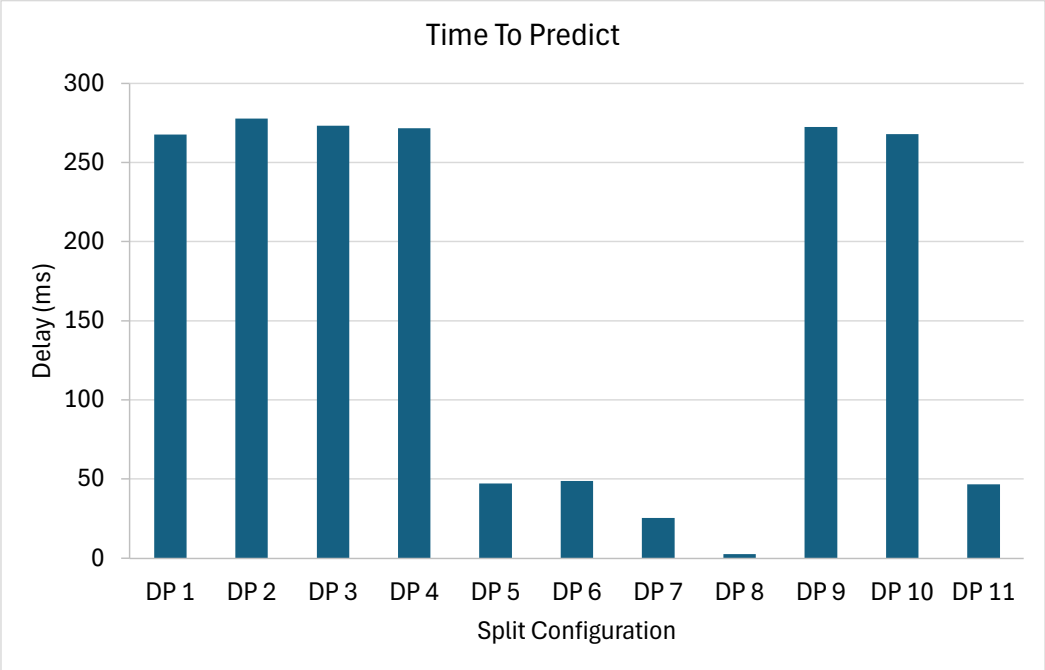


Figure 4.6: Time To Predict (i.e. inference delay) for each partition deployment.

By observing Figure 4.7, it is apparent that the configuration with the highest Network Usage is DP 2. In this configuration, the first block of the VGG16, VNF-ConvBlock1, is running on CUPE1, thus the prediction of this sub-model must traverse the entire network to reach the Cloud where the remainder of the DNN is placed. The same phenomenon is observed in DP 3, but here, having the VNF-ConvBlock2 in CUPE2 implies a significant reduction in the total Network Usage. This suggests that during deployment, it is crucial to place the VNF-ConvBlock1 block either within the same device that hosts VNF-ConvBlock2 or in close proximity to the CUPE hosting VNF-ConvBlock2 within the network. This strategy aims to significantly reduce the size of the final output from the CUPE. Minimizing the distance between the blocks prevents the massive amount of data from traversing multiple network points, thereby reducing network traffic. An interesting finding from the evaluation is the low Network Usage observed in DP 1. This is due to the compression of the model; having all the models in the same device means that only the image to predict and the prediction result travel across the network. This aspect makes this deployment a serious candidate if a "Cloud Configuration" deployment is chosen. From an energy standpoint, as expected, energy consumption

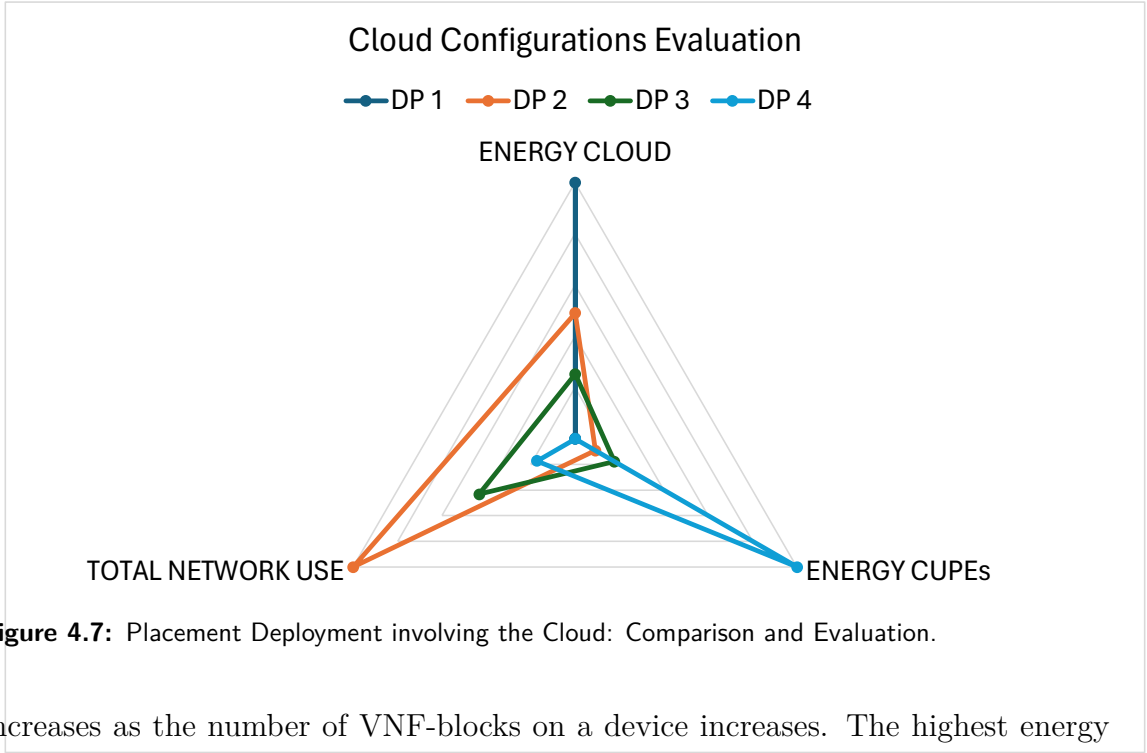


Figure 4.7: Placement Deployment involving the Cloud: Comparison and Evaluation.

increases as the number of VNF-blocks on a device increases. The highest energy consumption in the Cloud is found in DP 1, the full cloud configuration, while the highest energy consumption for the CUPE is found in DP 4, the configuration that has VNF-ConvBlocks running in INC. This important aspect shall be evaluated when choosing the deployment configuration to run.

4.6.1.2 Placement Deployment involving INC

Concerning the INC placement set shown in Figure 4.8, the deployment configurations in this group show significantly less delay than the ones in the previous group. Nevertheless, still some latency differences between them are observable, and thus the Time to Predict is a viable metric to compare these deployments. The Energy Consumed by the Cloud instead is not applicable as none of the models uses this element.

The distribution with the lowest latency is DP7. This is due to the concentration of all submodels in just two devices. It is also interesting to note that the energy of CUPEs does not experience a significant increase. However, the computation requirements of the CUPEs involved should be carefully checked before carrying out this placement, as inserting many submodels (e.g. VNF-ConvBlock/DNNBlock) into a single CUPE could cause its overload. On the other hand, the configuration

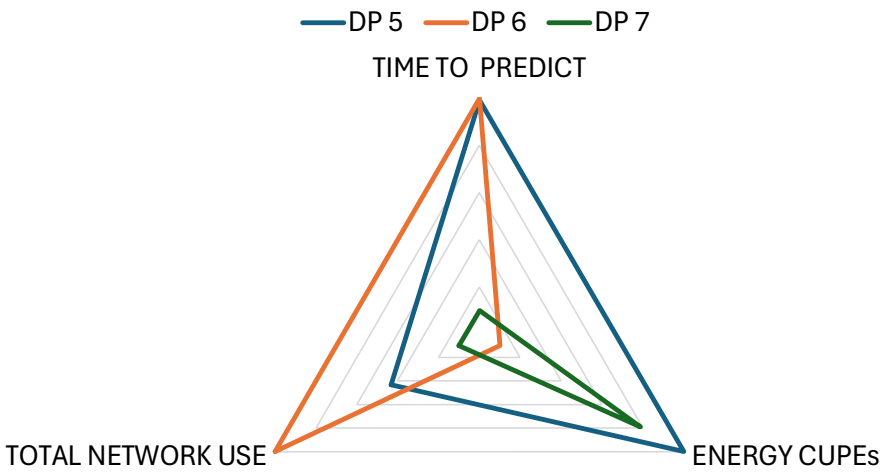


Figure 4.8: Placement Deployment involving INC: Comparison and Evaluation.

with the highest network usage is DP 6. This is due to the same phenomenon as before: the VNF-block with the highest data volume is isolated (VNF-ConvBlock1). Therefore, this division should not be implemented. The DP 5, although it has the highest delay and power consumption, shows attractive levels of network utilization and, unlike the DP 7, has a more balanced placement of VNF-blocks. This feature, in cases where the computing resources of the hardware running the CUPEs are not extensive, makes this configuration an interesting placement option. From the results of these experiments, it derives that failure to adhere to the principles presented above could lead to more challenges than benefits for future networks. Not to mention that the computational requirements of each of the VNF-blocks should be considered together with those of the CUPEs to determine the best allocation for each part of the network.

4.6.1.3 Placement Deployment involving UE

The last set of tests is meant to evaluate placement configurations also involving the UE in the NN outcome computation (as per placement configurations DP8, DP9, DP10, and DP11). Figure 4.9 shows the obtained results, obtained by normalizing values by the relevant maximum observation.

It can be observed that the Full Device setup (DP8) has, obviously, the shortest "Time to Predict" and the lowest "Total Network Use." However, this comes at the

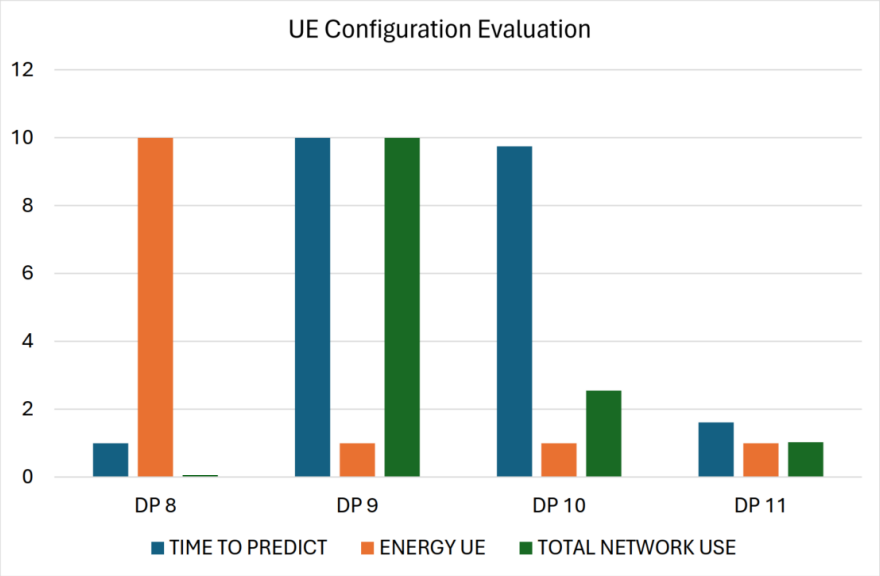


Figure 4.9: Placement Deployment involving UE: Comparison and Evaluation.

cost of the device's battery life, resulting in the highest UE energy consumption among all configurations. The configurations that use the Cloud (DP9 and DP10), as before, present a high "Time to Predict" due to the time required to reach the cloud. An interesting observation in these two deployments is the difference in the "Total Network Use". This one is produced because in the UE-Cloud configuration (DP9), the output of the VNF-ConvBlock2, which generates a significant amount of data, must travel across the network to arrive at the Cloud where the subsequent VNF is placed. Finally, the UE-INC configuration (DP11) presents the best balance considering all three performance evaluation parameters. This is given by not using the Cloud, which implies a reduction of the total prediction delay, and by having the highest data producer VNFs placed into the UE and the CUPE1, which allows for not overloading the network with data.

4.7 Main Insights and Takeaways

An approach for the structural and functional organization of future 6G networks is presented and evaluated, aiming to natively support distributed AI-relevant tasks. Building upon the envisioned reference architecture for future 6G networks outlined

in [2, 4], this thesis advances beyond previous works on Split-AI by also taking into account recent developments in the field of INC. More specifically, exploiting the concepts of the IUP and INC, it is foreseen a 6G architecture in which AI-relevant workflows – e.g. executing decomposed partitions of neural networks – can be carried out and possibly completed directly in the Packet Data Unit (PDU) session established for the end users. In light of this vision, a comprehensive analysis of the envisioned enablers at the CP and UP of future 6G networks is conducted. This is crucial to support and enable the tight integration of distributed and Split-AI relevant workflows within the upcoming generation of networks. Additionally, an initial PoC simulation model was developed to conduct a detailed experimental analysis. The PoC is focused on a Split-AI use case scenario in which the VGG-16 Convolutional Neural Network – used for image classification tasks – is decomposed into several split configurations. The conducted experimental campaign is aimed at evaluating and demonstrating the potential benefits for NN-relevant tasks achieved by efficiently leveraging existing networking devices and resources, varying the different split configurations of the considered NN. The results of this research shed light on the capabilities of the proposed approach to fully harness the computing capabilities of networking devices to enhance network utilization, reduce inference time, and optimize both computational and energy efficiency of the UEs.

Chapter 5 Conclusion

The advent of programmable networks and INC has led to a complete reshaping of the networks and how they are conceived. They are no longer simple communication mediums but constitute an active part of the computation of application-specific and general-purpose tasks and services. This trend led to breakthrough applications, like in-network aggregation or in-network caching, that allow the improvement of services' KPIs such as latency, throughput, and request completion time.

Recently, the widespread research and diffusion of AI applications have further accelerated this evolution. In the networking context, AI has established itself as a key-enabling technology to enhance network-relevant applications such as routing, load-balancing, security, and many others. The adoption of ML and DL for traditional networking tasks, combined with the flexibility of programmable networks, has resulted in significant improvements in both efficiency and performance. The tight integration of AI and networking unleashes the true power and benefits of the INC paradigm.

In this direction, this thesis explored the benefits of the breakthrough paradigm that dictates the tight integration between AI-relevant workflows and networking, envisioned as a main pillar for the upcoming 6G networks: AI4Net and Net4AI.

To support this vision, the first line of research focused on the design of a distributed, *in-network security fabric*. By leveraging programmable networking devices augmented with trained ML models, this work demonstrated how in-network anomaly detection could be effectively achieved. More specifically, by exploiting distributed and split-AI strategies, complex ML models can be decomposed into lighter components to be intelligently deployed on PDP devices to analyze the network flows that traverse them but preserving their networking capabilities. We achieved this objective by introducing the so-called *Projecting the Ensemble Learning Over the Network* split-AI strategy. We also supported the optimal and in-

telligent deployment and chaining of such decomposed and distributed lightweight AI components with the formalization of the *APSPC* problem. Results show that this approach provides a secure-by-design solution, reducing the computational burden on centralized entities while ensuring a robust and pervasive defense against malicious activities.

The second line of research delved into the INCaS-AI paradigm, which combines INC with programmable networks to efficiently execute distributed AI tasks across multiple network entities. In this direction, we started to devise the main logical and structural modifications for the Control and the User Plane of the upcoming 6G network architecture in order to natively accommodate for AI-relevant workflows. Specifically, we introduced the concept of IUP, in which the computation of AI-relevant workflows is not anymore solely demanded to the UE or the remote Application Server, but can be partially offloaded to the element that compose the PDU session of the user. Having that said, networking elements such as AN and UPF are now actively involved in computing the outcome of a layer of a Deep Neural Network (DNN), or more generally, the execution of a portion of an AI-relevant tasks. To account for this vision, the key element that we introduced as a main entity of the upcoming 6G IUP is the CUPE, which represents any of the mentioned networking elements that are augmented with INC and general-purpose computing capabilities.

We supported this study by developing and presenting an experimental Proof-of-Concept (PoC) simulation testbed, which allowed us to evaluate the benefits of the IUP for both network-related and UE-specific KPIs. Specifically, we demonstrated significant improvements in task completion time, such as the computation of the final classification outcome of a DNN, while simultaneously reducing overall bandwidth consumption. Moreover, our approach substantially alleviated the computational burden on the UE, leading to lower energy consumption. This reduction opens the possibility of simplifying UE designs in terms of computational and energy requirements by offloading a portion of the computation to the network. Such advancements would be particularly advantageous for future Augmented/Virtual Reality (AR/VR) applications, where wearable devices must minimize weight and physical encumbrance to enhance user comfort and usability.

Together, these contributions highlight the transformative potential of embedding AI within networking, moving beyond traditional paradigms to establish a ro-

bust, intelligent, and secure network architecture. By addressing both security and computational efficiency, this thesis has laid the groundwork for an AI-enhanced network infrastructure, demonstrating its applicability to emerging use cases and its alignment with the broader vision of 6G networks.

Looking forward, the integration of AI and networking opens exciting avenues for future research. Key challenges remain, including the optimization of resource allocation, scalability of distributed AI models and ensuring security in increasingly complex networked environments. The work presented in this thesis aimed at providing a strong foundation for addressing these challenges, contributing to the evolution of networks into truly intelligent, adaptive, and secure infrastructures capable of supporting the dynamic demands of the next-generation digital ecosystem.

References

[1] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with P4: Fundamentals, advances, and applied research," *Journal of Network and Computer Applications*, vol. 212, p. 103561, Mar. 2023.

[2] S. Schwarzmann, R. Trivisonno, S. Lange, T. E. Civelek, D. Corujo, R. Guerzoni, T. Zinner, and T. Mahmoodi, "An Intelligent User Plane to Support In-Network Computing in 6G Networks," in *ICC 2023-IEEE International Conference on Communications*, 2023, pp. 1100–1105.

[3] M. Spina *et al.*, "In-Network Computing and Split-AI in 6G: Enablers and Proof-of-Concept Studies," pp. 1–6, 2024.

[4] S. Schwarzmann, T. E. Civelek, A. Iera, D. Corujo, G. T. Karetsos, R. Guerzoni, O. Abboud, A. M. Valenzuela, R. Trivisonno, M. G. Spina, T. Zinner, and T. Mahmoodi, "Native Support of AI Applications in 6G Mobile Networks Via an Intelligent User Plane," in *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, 2024, pp. 1–6.

[5] M. G. Spina, F. De Rango, E. Scalzo, F. Guerriero, and A. Iera, "Distributing Intelligence in 6G Programmable Data Planes for Effective In-Network Deployment of an Active Intrusion Detection System," *arXiv*, Oct. 2024.

[6] M. G. Spina, E. Scalzo, F. De Rango, F. Guerriero, and A. Iera, "Optimal In-Network Distribution of Learning Functions for a Secure-by-Design Programmable Data Plane of Next-Generation Networks," *arXiv*, Nov. 2024.

[7] "one6G – Taking communications to the next level," Nov. 2024, [Online; accessed 3. Nov. 2024]. [Online]. Available: https://one6g.org

[8] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 1, pp. 27–51, Jun. 2014.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[10] Y. Li and M. Chen, "Software-Defined Network Function Virtualization: A Survey," *IEEE Access*, vol. 3, pp. 2542–2553, Dec. 2015.

[11] H. Farhady, H. Lee, and A. Nakao, "Software-Defined Networking: A survey," *Comput. Networks*, vol. 81, pp. 79–95, Apr. 2015.

[12] M. B. Jiménez, D. Fernández, J. E. Rivadeneira, L. Bellido, and A. Cárdenas, "A Survey of the Main Security Issues and Solutions for the SDN Architecture," *IEEE Access*, vol. 9, pp. 122 016–122 038, Sep. 2021.

[13] W. Quan, Z. Xu, M. Liu, N. Cheng, G. Liu, D. Gao, H. Zhang, X. Shen, and W. Zhuang, "AI-Driven Packet Forwarding With Programmable Data Plane: A Survey," *IEEE Commun. Surv. Tutorials*, vol. 25, no. 1, pp. 762–790, Oct. 2022.

[14] S. Kianpisheh and T. Taleb, "A Survey on In-Network Computing: Programmable Data Plane and Technology Specific Applications," *IEEE Commun. Surv. Tutorials*, vol. 25, no. 1, pp. 701–761, Oct. 2022.

[15] "In-Network Computing," Apr. 2019, [Online; accessed 3. Sep. 2024]. [Online]. Available: https://www.sigarch.org/in-network-computing-draft

[16] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-Network Computation is a Dumb Idea Whose Time Has Come," in *ACM Conferences*. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 150–156.

[17] D. R. K. Ports and J. Nelson, "When Should The Network Be The Computer?" in *ACM Conferences*. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 209–215.

[18] M. Dimolianis, A. Pavlidis, and V. Maglaris, "SYN Flood Attack Detection and Mitigation using Machine Learning Traffic Classification and Programmable Data Plane Filtering," in *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, pp. 01–04.

[19] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing Key-Value Stores with Fast In-Network Caching," in *ACM Conferences*. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 121–136.

[20] Z. Xiong and N. Zilberman, "Do Switches Dream of Machine Learning? Toward In-Network Classification," in *ACM Conferences*. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 25–33.

[21] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, N. Zilberman, H. Weatherspoon, M. Canini, F. Pedone, and R. Soulé, "P4xos: Consensus as a Network Service," *IEEE/ACM Trans. Networking*, vol. 28, no. 4, pp. 1726–1738, Aug. 2020.

[22] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "In-Network Machine Learning Using Programmable Network Devices: A Survey," *IEEE Commun. Surv. Tutorials*, vol. 26, no. 2, pp. 1171–1200, Dec. 2023.

[23] "P4 – Language Consortium," Sep. 2024, [Online; accessed 5. Sep. 2024]. [Online]. Available: https://p4.org

[24] The P4 Language Consortium, "P4~16~ Language Specification," May 2023, [Online; accessed 5. Sep. 2024]. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.4.html

[25] "behavioral-model," Sep. 2024, [Online; accessed 5. Sep. 2024]. [Online]. Available: https://github.com/p4lang/behavioral-model

[26] "Target Architecture • Vitis Networking P4 User Guide (UG1308) • Reader • AMD Technical Information Portal," Sep. 2024, [Online; accessed 5. Sep. 2024]. [Online]. Available: https://docs.amd.com/r/2023.1-English/ug1308-vitis-p4-user-guide/Target-Architecture

[27] "Open-Tofino," Sep. 2024, [Online; accessed 5. Sep. 2024]. [Online]. Available: https://github.com/barefootnetworks/Open-Tofino

[28] J. Pan, L. Cai, S. Yan, and X. S. Shen, "Network for AI and AI for Network: Challenges and Opportunities for Learning-Oriented Networks," *IEEE Network*, vol. 35, no. 6, pp. 270–277, Sep. 2021.

[29] C. Park, K. Park, J. Song, and J. Kim, "Distributed Learning-Based Intrusion Detection in 5G and Beyond Networks," in *Proceedings of the 2023 European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 2023, pp. 490–495.

[30] C. Kim, "Programming the network dataplane," *ACM SIGCOMM: Florianopolis, Brazil*, 2016.

[31] G. Siracusano and R. Bifulco, "In-network Neural Networks," *arXiv preprint arXiv:1801.05731*, 2018.

[32] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the AI accelerator?" in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, 2018, pp. 20–25.

[33] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: a data plane architecture for per-packet ML," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 1099–1114.

[34] T. Swamy, A. Zulfiqar, L. Nardi, M. Shahbaz, and K. Olukotun, "Homunculus: Auto-Generating Efficient Data-Plane ML Pipelines for Datacenter Networks," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 329–342.

[35] J.-H. Lee and K. Singh, "SwitchTree: in-network computing and traffic analyses with Random Forests," *Neural Computing and Applications*, pp. 1–12, 2020.

[36] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pForest: In-Network Inference with Random Forests," *arXiv preprint arXiv:1909.05680*, 2019.

[37] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, 2021, pp. 12–14.

[38] G. Xie, Q. Li, Y. Dong, G. Duan, Y. Jiang, and J. Duan, "Mousika: Enable General In-Network Intelligence in Programmable Switches by Knowledge Distillation," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1938–1947.

[39] C. Zheng, Z. Xiong, T. T. Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "IIsy: Hybrid In-Network Classification Using Programmable Switches," *IEEE/ACM Transactions on Networking*, 2024.

[40] G. Xie, Q. Li, G. Duan, J. Lin, Y. Dong, Y. Jiang, D. Zhao, and Y. Yang, "Empowering In-Network Classification in Programmable Switches by Binary Decision Tree and Knowledge Distillation," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 382–395, 2024.

[41] J. Gallego-Madrid, A. Molina-Zarca, R. Sanchez-Iborra, J. Ortiz, and A. F. Skarmeta, "Fast traffic processing in multi-tenant 5G environments: A comparative performance evaluation of P4 and eBPF technologies," *Engineering Science and Technology*, vol. 52, 2024.

[42] J. Gallego-Madrid, I. Bru-Santa, A. Ruiz-Rodenas, R. Sanchez-Iborra, and A. Skarmeta, "Machine learning-powered traffic processing in commodity hardware with eBPF," *Computer Networks*, vol. 243, 2024.

[43] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[44] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-split: A general framework of collaborative edge-cloud AI," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2543–2553.

[45] M. Saquetti, R. Canofre, A. F. Lorenzon, F. D. Rossi, J. R. Azambuja, W. Cordeiro, and M. C. Luizelli, "Toward In-Network Intelligence: Running Distributed Artificial Neural Networks in the Data Plane," *IEEE Communications Letters*, vol. 25, no. 11, pp. 3551–3555, 2021.

[46] J.-H. Lee and K. Singh, "SwitchTree: In-network Computing and Traffic Analyses with Random Forests," *Neural Comput. & Applic.*, 2020.

[47] "A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018)," accessed: 2023-04-04. [Online]. Available: https://registry.opendata.aws/cse-cic-ids2018

[48] R. Cerulli, F. Guerriero, E. Scalzo, and C. Sorgente, "Shortest paths with exclusive-disjunction arc pairs conflicts," *Computers & Operations Research*, vol. 152, p. 106158, 2023.

[49] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[50] E. Nuutila and E. Soisalon-Soininen, "On finding the strongly connected components in a directed graph," *Information processing letters*, vol. 49, no. 1, pp. 9–14, 1994.

[51] C. E. Andrade, R. F. Toso, J. F. Gonçalves, and M. G. Resende, "The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications," *European Journal of Operational Research*, vol. 289, no. 1, pp. 17–30, 2021.

[52] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016. [Online]. Available: https://doi.org/10.1016/j.orp.2016.09.002

[53] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2019, pp. 01–03.

[54] K. Doshi, Y. Yilmaz, and S. Uludag, "Timely Detection and Mitigation of Stealthy DDoS Attacks Via IoT Networks," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2164–2176, Jan. 2021.

[55] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[56] M. W. Nadeem, H. G. Goh, Y. Aun, and V. Ponnusamy, "Detecting and Mitigating Botnet Attacks in Software-Defined Networks Using Deep Learning Techniques," *IEEE Access*, vol. 11, pp. 49 153–49 171, 2023.

[57] "In-network-Distributed-IDS," Aug. 2024, [Online; accessed 19. Aug. 2024]. [Online]. Available: https://github.com/mattiagiovanni/In-network-Distributed-IDS

[58] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The Road Towards 6G: A Comprehensive Survey," *IEEE Open J. Commun. Soc.*, vol. 2, pp. 334–366, Feb. 2021.

[59] C.-X. Wang *et al.*, "On the Road to 6G: Visions, Requirements, Key Technologies, and Testbeds," *IEEE Commun. Surv. Tutorials*, vol. 25, no. 2, pp. 905–974, Feb. 2023.

[60] S. Yang, Z. Zhang, C. Zhao, X. Song, S. Guo, and H. Li, "CNNPC: End-edge-cloud collaborative CNN inference with joint model partition and compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4039–4056, 2022.

[61] J. He, H. Wu, X. Xiao, R. Bassoli, and F. H. P. Fitzek, "Functional Split of In-Network Deep Learning for 6G: A Feasibility Study," *IEEE Wireless Commun.*, vol. 29, no. 5, pp. 36–42, Oct. 2022.

[62] 3GPP, "5G System (5GS); Study on traffic characteristics and performance requirements for AI/ML model transfer," 3rd Generation Partnership Project, Technical Report 22.874, 12 2021, version 18.2.0.

[63] T. A. Benson, "In-Network Compute: Considered Armed and Dangerous," in *HotOS '19: Proceedings of the Workshop on Hot Topics in Operating Systems*. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 216–224.

[64] Y. Zhou, L. Liu, L. Wang, N. Hui, X. Cui, J. Wu, Y. Peng, Y. Qi, and C. Xing, "Service-aware 6G: An intelligent and open network based on the convergence of communication, computing and caching," *Digital Communications and Networks*, vol. 6, no. 3, pp. 253–260, 2020.

[65] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, "The Programmable Data Plane: Abstractions, Architectures, Algorithms, and Applications," *ACM Comput. Surv.*, vol. 54, no. 4, may 2021. [Online]. Available: https://doi.org/10.1145/3447868

[66] S. Kianpisheh and T. Taleb, "A Survey on In-Network Computing: Programmable Data Plane and Technology Specific Applications," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 701–761, 2023.

[67] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, pp. 150–156.

[68] D. R. K. Ports and J. Nelson, "When Should The Network Be The Computer?" in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 209–215. [Online]. Available: https://doi.org/10.1145/3317550.3321439

[69] W. da Costa Cordeiro and et.al., "Data Plane Programmability Beyond OpenFlow: Opportunities and Challenges for Network and Service Operations and Management," *J Netw Syst Manage*, vol. 25, p. 784–818, 2017.

[70] N. Hu, Z. Tian, X. Du, and M. Guizani, "An energy-efficient in-network computing paradigm for 6G," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 4, pp. 1722–1733, 2021.

[71] J. A. Brito, J. I. Moreno, L. M. Contreras, M. Alvarez-Campana, and M. Blanco Caamaño, "Programmable Data Plane Applications in 5G and Beyond Architectures: A Systematic Review," *Sensors*, vol. 23, no. 15, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/15/6955

[72] R. A. Cooke and S. A. Fahmy, "Quantifying the latency benefits of near-edge and in-network FPGA acceleration," in *ACM Conferences*. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 7–12.

[73] NTT Network Service Systems Laboratories. (2023) Inclusive Core: Integrative and Cooperative Network Architecture for the 6G/IOWN Era. [Online]. Available: https://www.rd.ntt/e/ns/inclusivecore/whitepaper_ver1.html

[74] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "Caltech-UCSD Birds-200-2011," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011.

Acknowledgments

The work presented in Chapter 3 was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on “Telecommunications of the Future” (PE00000001 - program “RESTART”).

The research described in Chapter 4 was partly conducted in the framework of the Working Item (WI) 207 on ”Intelligent User Plane and In-Network Computing” of the one6G association. Thanks therefore go to all active members of the WI for the fruitful discussions.