Integrating AI Orchestration and Lifecycle Management in 6G Networks: A Pipeline Approach

Julian Jimenez*, Andreas Gavrielides†, Nina Slamnik-Kriještorac†,
Steven Latré*, Johann M. Marquez-Barja†, Miguel Camelo*
*University of Antwerp - imec, IDLab - Department of Computer Science, Belgium
†University of Antwerp - imec, IDLab - Faculty of Applied Engineering, Belgium

***Abstract*—On the threshold of a new technological era, Sixth Generation (6G) networks promise to revolutionize global connectivity, bringing mobile communications to data speeds in the terabits per second range and ultra-low latency. These networks will enhance the user experience enable a wide range of advanced applications and emerging services. Artificial Intelligence (AI)-powered network functions and services, also known as Network Intelligence Functions (NIF) and Network Intelligence Service (NIS), are essential to achieve this vision. In this study, we present the design and development of an end-to-end framework for orchestrating AI-based functions. Utilizing Kubernetes (K8s) and Prefect, we showcase its implementation through an AI-driven Traffic Classification (TC) use case. Our results confirm the feasibility of the proposed framework, offering valuable insights in the lifecycle management design, such as data collection, decision-making, and critical performance metrics, including deployment time and model performance in terms of accuracy and inference times among three different Machine Learning (ML)-based TC models.**

***Index Terms*—Network Intelligence, Network Orchestration, Machine Learning, Lifecycle Management, MLOps, ML pipeline, Artificial Intelligence.**

I. Introduction

To manage next-generation networks with higher speeds, lower latency, enhanced user experiences, and the flexibility to adapt to emerging services, AI is required as a key enabler for automating both physical and logical resources [1]. Realizing this vision will depend on the Network Intelligence Stratum (NI Stratum) framework [2], which will be fundamental in enabling the native management and lifecycle handling of novel AI-based Network Functions (NFs) and Network Services (NSs), also referred to as NIFs and NISs. NISs are hierarchical services composed of multiple NIFs and NIF Components (NIF-Cs), with NIFs serving as the foundational blocks that perform tasks such as data collection and data-driven decision-making. These NIFs are managed through a NIF Manager and NIFs Component Manager, responsible for lower-level functions like container management for specific intelligence components. The orchestration of these building blocks, primarily handled by the Network Intelligence Orchestrator (NIO) [3], is crucial for integrating and managing intelligence within the network.

In order to achieve the extremely high data rates and ultra-low latency expected in 6G networks, technological enablers such as edge computing, Network Function Virtualization (NFV), Software Defined Networking (SDN), and Service-Based Architecture (SBA) are fundamental. his approach offers the flexibility and scalability needed to manage large volumes of real-time data on adaptable network infrastructure [4]. It is in this context where technologies such as K8s are playing an essential role to provide cloud-native deployments anywhere from cloud to edge. K8s provides a flexible and efficient environment for container management, enabling the automatic orchestration of containerized applications, such as traditional NF/NS and modern ones such as the one empowered by AI, i.e., NIF/NIS, which is a requirement to realize a full AI-native architecture.

These AI-native architectures rely on seamlessly integrating AI/ML algorithms into the network infrastructure. One way to achieve this is through ML pipelines [5], which prepare data-driven models that enable various functionalities. Proper design and implementation of these pipelines are crucial for managing the lifecycle of NIF/NIS, working in conjunction with NIF Managers, the NIF Component Manager, and key modules like AI/ML analytics and monitoring blocks. This approach enables the evaluation of the Network Intelligence (NI) algorithm's behavior and the implementation of corrective measures, such as performing specialized orchestration operations for NI [6]–[8], in response to performance degradation, which traditional Management and Orchestration (MANO) systems are unable to handle effectively [9].

In the literature, various works propose different architectures for orchestrating intelligence [6]–[8], [10], generally defining the necessary components to equip 6G networks with AI. However, the designing and developing of the ML pipelines has not been addressed. In this paper, we present an implementation of a NI Stratum that also includes the ML pipeline functional block and its interactions with the rest of the NI Stratum functionalities to achieve a complete NIS/NIF workflow management and control. More precisely, we design and implement the enablers that are required to integrate the ML pipeline framework, which is based on Prefect[1], the NIF manager and component manager, based on K8s, and custom modules to realize the NIO. Figure 1 presents an architecture of the NI Stratum [3], highlighting key functional blocks involved in this research.

The proposed framework allows the efficient and scalable workflow orchestration for data and ML model pipelines to

[1]https://www.prefect.io/cloud

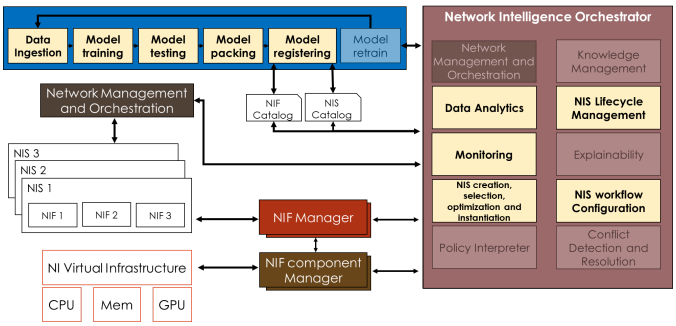

Fig. 1. The NI Stratum and its functional blocks. The highlighted blocks are the ones studied in this paper to support our approach.

fully realize the NI Stratum. As as results, this combination allows intelligent services to be deployed and managed effectively, ensuring smooth integration with the underlying network infrastructure. To demosntrate the capabilities of the proposed frameworks, we use TC [11]–[13], a key NF for current and future networks, which allows us to identify the main elements required to define the ML pipelines in a unified way. These pipelines facilitate the efficient training and deployment of models and ensure that models can be continuously updated and improved based on real-time data.

The paper is organized as follows: Section II reviews related works, and Section III introduces the proposed framework for AI lifecycle management. Section IV discusses intelligence orchestration for the traffic classification use case and system operation, followed by results in Section V. Finally, Section VI presents conclusions and future work.

II. Related works

A. Orchestrating Intelligence

The work of [7] presents the NIO framework for 6G networks (Figure 1), as the architecture component responsible for managing and coordinating NI instances across a network infrastructure efficiently and effectively. It involves the use of AI and ML techniques to automate and optimize network operations, from resource management to service configuration and maintenance. The authors also propose an architectural model with an NIO layer to ensure efficient end-to-end coordination of NI instances across the network. An specific application of such framework is presented in [14], where the authors discuss the integration of NI within Virtualised Radio Access Network (vRAN). It emphasizes the need for effective orchestration and control mechanisms to handle the complex state spaces and numerous parameters in vRAN environments.

More recently, an updated version of the architecture is presented in [3], [15], where the Network Intelligence Plane (NIP) is introduced to natively integrate NI into the operations, management, and orchestration of the networks. Among the evolution, this paper introduces and defines the NI interfaces for communication between NI instances and NIO to manage the lifecycle of multiple NI instances such as NIS and NIF.

To manage AI functions automatically, the work of [16] proposes different ways to fully manage the lifecycle of AI projects, to cover not only ML but also other areas of AI. This approach focuses on managing each stage of the development and deployment of AI solutions, from problem definition and goal setting, data collection and preparation, model development and training, model evaluation and validation, to deployment, monitoring, maintenance, and eventual retirement of models when they are no longer relevant or effective.

Also standarization bodies has started to define how to integrate and perform the AIML lifecycle management in their studies. An example of this is Open RAN (O-RAN), which has a dedicated working group for this and has provided initial guidelines for it [17]. However, notice that although O-RAN has already begun to define mechanisms for AIML lifecycle management, the details regarding their mechanism and its realization remain unexplored.

B. MLOps frameworks

As discussed above, most of recent academic and industry works are mainly focusing on the architectural design and guidelines for AI/ML lifecycle management. To bridge this, it is essential to further study on how to integrate network orchestration mechanism with AI/ML lifecycle management. On the lifecycle management, Machine Learning Operations (MLOps) plays a fundamental role since it covers all phases of the ML solution development, from idea conception to removal of such solutions. MLOps [18] is a practice derived from DevOps that focuses on collaboration and communication between data scientists, data engineers, and operations developers to automate and optimize the process of developing, deploying, and maintaining ML models.

In the recent years, several tools have been developed to address the challenges of MLOps, each with unique features. Metaflow[2] simplifies data science workflows with strong Python integration and basic K8s support, making it ideal for moderate-scale environments. Prefect provides flexible workflow management with robust K8s integration and minimal computational requirements. Kedro[3] supports building modular pipelines but lacks native K8s support, which can limit its scalability in cloud-native environments. Truera[4] specializes in model explainability and validation with low resource requirements, but it also lacks K8s integration, limiting its use in large-scale deployments. Kubeflow[5], on the other hand, offers a comprehensive suite for end-to-end ML lifecycle management with strong K8s integration but demands significant resources and is complex to set up.

Acumos[6] is another platform focusing on the full ML lifecycle, supporting Docker and K8s deployments. It provides components for model deployment, microservice generation, and orchestration, making it suitable for scalable and production-ready environments. For example, in [11], Acumos is used in the RAN Intelligent Controller (RIC) to train ML models for

[2]https://www.metaflow.org/
[3]https://www.kedro.org/
[4]https://truera.com/
[5]https://www.kubeflow.org/
[6]https://docs.acumos.org/en/elpis/

TABLE I
COMPARISON AMONG DIFFERENT MLOPS TOOLS

Tool	Integration with Python	Kubernetes Support	Computational Resources Required
Metaflow	Yes	Yes Limited	Moderate
Prefect	Yes	Yes	Low
Kedro	Yes	No	Moderate
Truera	Yes	No	Low
Kubeflow	Yes	Yes	High
Acumos	Yes	Yes	High

user traffic prediction at the LTE access network level. Similarly, [19] demonstrates how an AI engine can use Acumos to encapsulate multiple AI algorithms for managing RAN slices, automating the training and deployment of models based on user-defined service requirements. Table I summarizes the key attributes of these tools, comparing their integration with Python, K8s support, and resource requirements. This comparative overview helps in identifying the appropriate tool, considering the infrastructure and scalability needs.

Notice that current research predominantly focuses on using AI to enhance network performance or on developing new MLOps tools to streamline the ML workflow. However, there is a significant gap in integrating these aspects into a cohesive framework for lifecycle management of NIF/NIS. Most existing works overlook how to natively incorporate MLOps into the network for orchestrating NI, resulting in fragmented solutions that do not fully realize the potential of AI-driven network management.

III. GENERAL FRAMEWORK FOR THE LIFECYCLE OF AI-BASED NETWORK FUNCTIONS AND SERVICES

A. Lifecycle management of AI-based functions

As identified in the previous section, there is a need to provide a complete approach that seamlessly integrates MLOps tools with the NIO in order to enable automated, scalable, and efficient deployment and management of NI. In order to achieve a holistic approach for orchestration of intelligence, we need to ensure that the practices and tools for developing, deploying, and maintaining ML models are well integrated into network orchestration frameworks so there is no more differences between creating a NF/NS and NIF/NIS.

Let us start by describing a general approach for lifecycle management of AI-based functions that can serve as baseline for NI orchestration. The general framework is presented in Figure 2 and is composed of the following seven phases:

1) **Planning and Design:** In this initial phase, specific requirements and issues that AI must address. The scope of the AI model is defined as well, including objectives, architecture, and necessary data.
2) **Data Preparation:** AI models require training data to learn and adapt. This phase involves collecting the required amount of data from the data sources that are associated to the problem. The data must be cleaned, normalized, and processed to ensure its utility in model training.

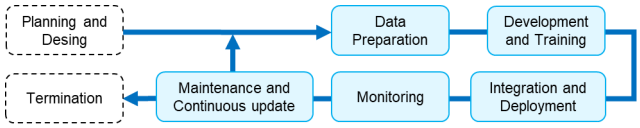

Fig. 2. Main steps to achieve a lifecycle management of AI-based functions

3) **Development and Training:** AI models are developed using the prepared data. ML techniques are selected in this phase, and models are trained to solve the specific tasks defined in phase 1.
4) **Integration and Deployment:** Once the AI model is trained, it is integrated with the system and deployed. At this point, the trained model will make decisions within the system.
5) **Monitoring:** The deployed model is monitored to verify its performance and effectiveness. This phase is crucial as model performance may change over time due to different circumstances, e.g., data drifting. In such cases, model monitoring will allow adjustments or retraining of the model to maintain its performance and efficiency.
6) **Maintenance and Continuous Update:** AI models require regular maintenance as system conditions and data patterns evolve. This phase includes retraining models with new data, adjusting algorithms as necessary, and updating systems to incorporate technological advances and mitigate new threats or vulnerabilities.
7) **Termination:** When a model becomes obsolete or is replaced by a more efficient solution, it is removed from the system. This phase ensures that the removal does not negatively affect the system.

B. Lifecycle management of NIF and NIS

Although the previous phases are very generic to cover the full lifecycle management of AI-based functions, it is important to adapt it to the specific details of networking problems. To implement the phases presented in Figure 2, we need to integrate them into the NI Stratum architecture as presented in Figure 1.

1) **Planning and Design.** In this phase, the networking problem that needs to be solved is analyzed and described as a NI problem. It means that scope and objectives of the networking problem are defined together with the data collection strategy, model selection criteria and performance indicators. Moreover, security and privacy considerations together with closed loop control strategies for monitoring are also defined.
2) **Data Preparation.** This phase is responsible for preprocessing and preparing the necessary data for training and validating the NI algorithm. At the NIO level, this is carried out by the first step of the ***ML pipeline's*** functional block, which is activated when new model training or retraining is required. Access to network monitoring tools to capture data and storage is key in this step.

3) **Development and Training.** This phase involves key steps including model training, testing, packaging, and registering in the NIF/NIS. A critical aspect is ensuring the resulting model is packaged in a format suitable for deployment by orchestrators (e.g., Virtual Network Function (VNF) packages) or controllers (e.g., xApp), allowing seamless integration and operation within the network environment. It is important to note that the NIO will execute all these steps through ***workflow pipelines***, necessitating interaction between functional blocks, including ***NIS workflow configuration*** and ***NIS lifecycle management***.
4) **Integration and Deployment:** Once the NI algorithm has been trained and registered in the NIF/NIS catalog, the NIO will provide the necessary pipeline workflows to facilitate the operation of the NIF/NIS, in coordination with the network orchestrator/controller. For instance, depending on the hardware constraints of the network environment, provided by the network controller, the functional block ***NIS Creation, Selection, Optimization, and Instantiation*** may trigger a ML pipeline workflows to perform model optimization to better suit the target environment. Furthermore, if the optimization is insufficient, it may lead to the creation of new NISs, triggering again the pipelines of the previous phase.
5) **Monitoring:** With the same name, the ***Monitoring*** functional block in the NIO supervises the performance and status of NIFs and NISs in real-time. It collects metrics related to the learning and network metrics in order to detect any performance degradation, allowing for proactive corrective actions, if required.
6) **Maintenance and Continuous Update:** NI functions and services require continuous maintenance as network conditions and traffic patterns evolve. This phase involves retraining models with updated network data, refining algorithms to adapt to changing performance demands, and updating systems to leverage technological advancements.

To implement the previous phases, excluding the first one which does not involve actual implementation, we utilize ML pipeline workflows. These workflows represent the sequence of processes designed to manage various stages of the lifecycle of NI functions and services. Executed in a structured and automated manner, they ensure that all necessary software components (e.g., libraries) and hardware resources (e.g., accelerators) are available. The ***NIS Lifecycle Management*** functional block dynamically controls how the NIO transitions through the different phases to ensure smooth execution.

IV. Orchestrating intelligence for traffic classification

A. Traffic classification use case and its lifecycle management

In the previous section, we introduced a complete framework for managing the entire lifecycle of NIF and NIS. In this section, we will demonstrate how this framework can be applied in a practical networking use case as illustrated in Figure 3. The system begins with **Step 0**, where the gNodeB (gNB) starts processing data packets from the User Equipment (UE). As the packets pass through the gNB in **Step 1**, they are forwarded to both a) a data lake for future use in training, and b) the xApp[7] responsible for the TC task. The initial TC can range from a simple rule-based algorithm to more advanced approaches, such as those based on Deep Learning (DL) architectures. In the case of ML algorithms, it is crucial to ensure that the deployed algorithm has already been trained, as specified by O-RAN [17].

The xApp performs the TC task and sends the results of its traffic classification predictions to the NIO in **Step 3**. This step is crucial for monitoring the model's performance, enabling the NIO to support maintenance and continuous improvement of the process. If the model's performance is inadequate, the system proceeds to **Step 4**, where a more suitable ML model is selected, triggering the training and deployment ***pipeline workflows*** to create a new xApp. Before creating a new xApp, the NIO first checks the xApp catalog in **Step 5** to determine if a similar xApp already exists. If no such xApp is found, the process to create a new one begins in **Step 5.1**, using the data stored in the data lake in **Step 5.2**. Once the new xApp is created, it is stored in the xApp catalog in **Step 5.3** and deployed to replace the previous xApp in **Step 6**, which will then be turned off and removed from the system in **Step 7**. The newly deployed xApp will now carry out the TC task as described in **Step 3**. This closed-loop process continuously retrains models as needed, adapting to changes in the network and model performance.

B. Framework implementation

To build this use case using the framework for NI lifecycle management, each phase described below was implemented using Prefect. Prefect was chosen as the pipeline tool to manage all the steps required to achieve comprehensive ***lifecycle management*** for NIFs/NISs. In essence, the NIO processes are structured as a series of pipeline workflows within this tool. We implemented three pipelines to support the training, creation, and deployment of new xApps utilizing different ML models, including Linear Regression (LR), Decision Tree Regression (DTR), and Deep Neural Network (DNN). These models were pre-defined during the planning and design phase for the TC use case. Moreover, these pipelines contains the code that is required to interface with the NIF manager, in our case K8s. Additionally, three more pipelines were established to facilitate maintenance and continuous updates, ensuring the ongoing optimization and performance of the deployed xApps. These pipelines also support tasks such as the creation of NIFs/NISs based on containers already available in the catalog, streamlining the deployment and management process.

[7]An xApp is a specialized application from the O-RAN ecosystem that optimizes and manages functions operating within the Radio Access Network (RAN) environment. In this paper, an xApp can be seen as a NIF, or if it is composed of several of them, then as a NIS.

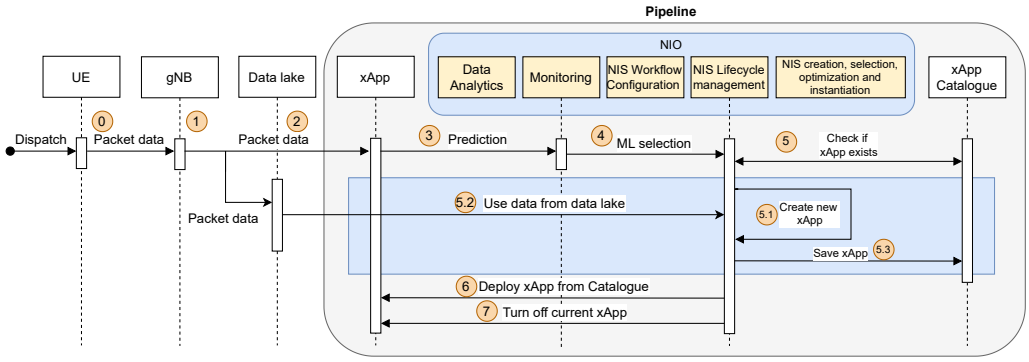

Fig. 3. Sequence diagram of the process of deploying a NIF for Traffic Classification (TC)

- **Data Preparation**: This phase begins with the collection of data packets from the UEs, typically captured at layer two of the gNB. The packets are stored in the pcap[8] format. Once stored, the data is made available either in a bucket or a data lake, depending on the deployment infrastructure.
  After data collection, we proceed with the pre-processing step, which involves extracting and transforming parameters from the pcap files into a set of selected features for training the ML models. We chose the CSV format to store these features due to its ease of manipulation during the training phase. In this use case, the key features extracted for training include: frame length, source IP, destination IP, data length, source port, and destination port. It is important to note that the protocol type, extracted from layer 4 information, will serve as the label for supervised training.
- **Development and Training**: To create an ML model for performing TC, the pre-processed dataset is divided into training and test sets using the scikit-learn[9] libraries. Once the model is trained, it is saved as a serialized file, which can then be embedded into a NIF/NIS as a pod[10] or container[11]. This enables the network controller or orchestrator to easily deploy the model within the infrastructure.
- **Integration and Deployment**: To integrate the serialized trained ML model with the Python code handling the xApp's processing logic, a custom template was designed with placeholders for model name, location, and xApp name. These placeholders are dynamically replaced during deployment for flexible customization. Next, a Dockerfile is created to define the dependencies and configurations, including the model and Python code. Once the image is built, it is added to the xApp Catalog, ready for deployment by the controller.
  To deploy the new xApp, a K8s manifest is utilized, which is invoked by the NIO through the appropriate methods from the K8s library. For the NIO to effectively interact with the cluster that hosts both the Orchestrator and xApps, it must be granted the necessary roles and permissions in line with the Role-Based Access Control (RBAC) system. These permissions are configured using the "ClusterRole" and "ClusterRoleBinding" objects in the manifest, ensuring the NIO has the appropriate access to manage resources within the cluster.
- **Monitoring, Maintenance, and Continuous Update**: To monitor and evaluate the model's performance, we consider both its efficiency, measured by predictions per second, and its accuracy. Predictions per second is a key metric for assessing scalability with traffic loads and users, while accuracy ensures the model performs its intended task. By continuously monitoring these performance indicators, the NIO can dynamically select the most suitable TC model, not only based on accuracy but also considering the available network and computing resources, enabling fully automated lifecycle management of NI functions and services, represented by an xApp in this use case.

[8]https://www.endace.com/learn/what-is-a-pcap-file
[9]https://scikit-learn.org/stable/
[10]https://kubernetes.io/docs/concepts/workloads/pods/
[11]https://www.docker.com/resources/what-container/

V. Evaluation Results

To validate the functionalities of the NIO using pipeline workflows that support various phases of the proposed framework for the TC task, we utilized a dataset from [20], provided in pcap format and containing 1,803,059 packet samples across three traffic classes: TCP (999765 samples), UDP (803087 samples), and MPTCP (207). We used 70% of the dataset for training and developed a custom replay script with the remaining 30% as test data to simulate continuous network traffic injection, enabling real-time processing and pipeline evaluation. The validation setup involved a desktop machine with an AMD Ryzen 9 5900 12-core processor, an Nvidia RTX 3080 graphics card, 64GB of RAM, and Windows 11, while Docker Desktop with K8s v1.29.1 was used for container management.

A. Data Preparation, Development, and Training Times

The first aspect evaluated was the time required to deploy a fully trained TC model using our pipelines, starting from

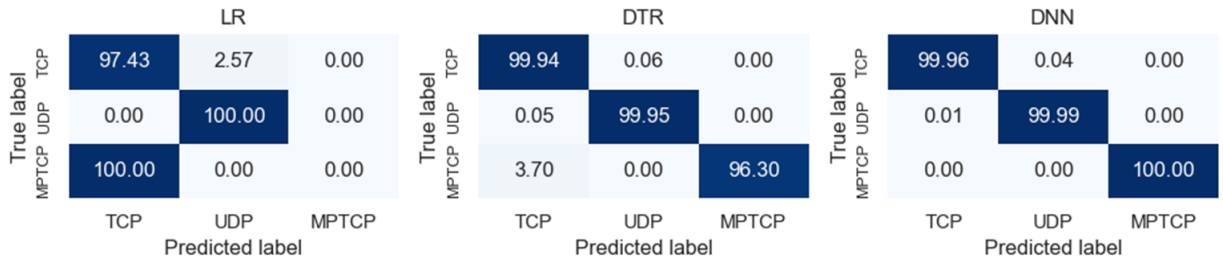

Fig. 4. Confusion matrix of the different ML models. While the accuracy of the DTR is higher overall, the DNN performs better among all classes.

TABLE II
TOTAL EXECUTION TIME OF DIFFERENT PIPELINES

Model	Data Preparation	Development and Training	Integration	Deployment
LR	5	4	112	5
DTR	5	6	108.5	5
DNN	5	1033.5	112.5	5

TABLE III
PREDICTIONS OF A PACKET CLASS PER SECOND

Mean inference time	RL	DTR	DNN
Packets/Seconds	264027.6	264217.8	13829.34

data preparation to xApp deployment. As shown in Table II, deployment times vary depending on the model used during training. While data preparation, integration, and deployment phases are similar across models, training time differs significantly, with more complex models requiring longer training periods.

B. Accuracy Performance

In terms of accuracy, confusion matrices for the three classification models are shown in Figure 4. The DNN model achieves the highest accuracy, even with highly imbalanced classes, followed by the DTR. Both models perform similarly for the larger classes, but the DNN shows a 4% improvement in classifying the MPTCP class compared to the DTR. In contrast, the LR model struggles with MPTCP classification but performs well for the two major classes (TCP and UDP).

C. Inference Time Performance

Model choice has a significant impact on packet processing time, as detailed in Table III. The DTR model strikes the best balance between accuracy and processing speed, leveraging the non-encrypted nature of the data and multi-core processors for efficient training. On the other hand, the DNN model is up to 19 times slower than both LR and DTR due to its complexity, with only marginal accuracy gains, which is aligned with previous research on ML models for TC [21]. For encrypted traffic, simpler models like DTR may not suffice, and more complex models such as DNN may be necessary despite their slower performance [22], [23]. In such cases, hardware acceleration could improve DNN inference speed, offering a balance between accuracy and processing time.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we outlined the requirements for AI lifecycle management of NIF and NIS and introduced a pipeline approach to enable their orchestration. Additionally, we validated and evaluated the performance of the proposed framework by implementing a TC use case, covering everything from data capture to inference, utilizing K8s and Prefect. While K8s is used to manage the interaction between containers, Prefect is employed to create new NIFs for TC throughout the entire pipeline lifecycle. Although we focus on a single use case, the proposed framework is designed to be generic, allowing it to be applied and extended to other NIF/NIS use cases as needed.

In future work, we aim to explore advanced AI-based algorithms for orchestration of intelligence that enables end-to-end lifecycle management and orchestration of NIF and NIS, where the interactions between these functionalities are more complex (e.g., conflict resolution, knowledge sharing, etc.). In addition, we will also explore how to design the ML pipelines for training and inference in isolated (e.g., cloud for training and edge for inference) and shared (training and inference at the edge) environments. For other use cases, we will investigate the orchestration of intelligence related to Network Digital Twins (how to orchestrate intelligence inside and outside of it) and those related to deeply embedded AI for networking (e.g., inference at the physical layer of intelligent radios or intelligent switches).

ACKNOWLEDGMENT

The architectural design of the framework was funded through the imec.icon project 5GECO. 5GECO is co-financed by imec and receives financial support from Flanders Innovation and Entrepreneurship (project nr. HBC.2021.0673) and/or Innoviris. The rest of this research, including implementation and experimentation, has been funded by the 6G-TWIN project, which has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation program under Grant Agreement No 101136314. However, views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or Smart Networks and Services Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] M. Polese, M. Dohler, F. Dressler, M. Erol-Kantarci, R. Jana, R. Knopp, and T. Melodia, "Empowering the 6g cellular architecture with open ran," *IEEE Journal on Selected Areas in Communications*, 2023.

[2] P. Soto, M. Camelo, G. Garcia-Aviles, E. Municio, M. Gramaglia, E. Kosmatos, N. Slamnik-Krieštorac, D. D. Vleeschauwer, A. Bazco-Nogueras, L. Fuentes, J. Ballesteros, A. Lutu, L. Cominardi, I. Paez, S. Alcalá-Marín, L. E. Chatzieleftheriou, A. Garcia-Saavedra, and M. Fiore, "Designing the network intelligence stratum for 6g networks," 2024.

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

[4] X. You, C.-X. Wang, J. Huang, X. Gao, Z. Zhang, M. Wang, Y. Huang, C. Zhang, Y. Jiang, J. Wang, *et al.*, "Towards 6g wireless communication networks: Vision, enabling technologies, and new paradigm shifts," *Science China Information Sciences*, vol. 64, pp. 1–74, 2021.

[5] Y. Zhou, Y. Yu, and B. Ding, "Towards mlops: A case study of ml pipeline platform," in *2020 International conference on artificial intelligence and computer engineering (ICAICE)*, pp. 494–500, IEEE, 2020.

[6] K. Kassai, S. Ghosh, and A. Dagiuklas, "Evolution of orchestration towards 5g," *Journal of Communication*, vol. 14, no. 12, 2019.

[7] M. Camelo, L. Cominardi, M. Gramaglia, M. Fiore, A. Garcia-Saavedra, L. Fuentes, D. De Vleeschauwer, P. Soto-Arenas, N. Slamnik-Krijestorac, J. Ballesteros, *et al.*, "Requirements and specifications for the orchestration of network intelligence in 6g," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–9, IEEE, 2022.

[8] L. I. Barona López, J. Maestre Vidal, and L. J. García Villalba, "Orchestration of use-case driven analytics in 5g scenarios," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, pp. 1097–1117, 2018.

[9] X. Cai, H. Deng, A. Lingli Deng, S. Gao, A. Nicolas, Y. Nakajima, J. Pieczerak, J. Triay, X. Wang, B. Xie, *et al.*, "Evolving nfv towards the next decade," *ETSI White Paper*, no. 54, 2023.

[10] S. Faye, M. Camelo, J.-S. Sottet, C. Sommer, M. Franke, J. Baudouin, G. Castellanos, R. Decorme, M. P. Fanti, R. Fuladi, G. Kesik, B. Mendes, C. Murphy, S. Parker, S. Pryor, S. M. Senouci, and I. Turcanu, "Integrating network digital twinning into future ai-based 6g systems: The 6g-twin vision," in *2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pp. 883–888, 2024.

[11] H. Lee, J. Cha, D. Kwon, M. Jeong, and I. Park, "Hosting ai/ml workflows on o-ran ric platform," in *2020 IEEE Globecom Workshops (GC Wkshps*, pp. 1–6, 2020.

[12] G. Lu, R. Guo, Y. Zhou, and J. Du, "An accurate and extensible machine learning classifier for flow-level traffic classification," *China Communications*, vol. 15, no. 6, pp. 125–138, 2018.

[13] M. S. Raza, K. Aziz Bhatti, F. M. Malik, and S. Amin Sheikh, "Network traffic classification using deep neural networks," in *2023 International Conference on Frontiers of Information Technology (FIT)*, pp. 85–89, 2023.

[14] M. Gramaglia, M. Camelo, L. Fuentes, J. Ballesteros, G. Baldoni, L. Cominardi, A. Garcia-Saavedra, and M. Fiore, "Network intelligence for virtualized ran orchestration: The daemon approach," in *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pp. 482–487, 2022.

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

[16] D. D. Silva and D. Alahakoon, "An artificial intelligence life cycle: From conception to production," 2021.

[17] O.-R. Alliance, "O-ran ai/ml workflow description and requirements, o-ran.wg2.aiml-v01.03," technical report, O-RAN Alliance, 2021.

[18] Google Cloud, "Mlops: Continuous delivery and automation pipelines in machine learning." https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning, 2024. Accessed: 2024-05-03.

[19] H. Zhang, W. Guan, D. Wang, Q. Song, and A. Nallanathan, "Demo: Ai-engine enabled intelligent management in b5g/6g networks," in *2022 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–2, 2022.

[20] A. Khan, "Network traffic dataset," 2019. Accessed: 2023-10-02.

[21] K. Ismailaj, M. Camelo, and S. Latré, "When deep learning may not be the right tool for traffic classification," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 884–889, 2021.

[22] M. Camelo, P. Soto, and S. Latré, "A general approach for traffic classification in wireless networks using deep learning," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 5044–5063, 2022.

[23] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.