future internet

MDPI

Article

Multi-Agent-Based Traffic Prediction and Traffic Classification for Autonomic Network Management Systems for Future Networks

Sisay Tadesse Arzo [1,*,†], Zeinab Akhavan [1,†], Mona Esmaeili [1,†], Michael Devetsikiotis [1,†] and Fabrizio Granelli [2,†]

[1] Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87106, USA; zakhavan@unm.edu (Z.A.); mesmaeili@unm.edu (M.E.); mdevets@unm.edu (M.D.)
[2] Department of Information Engineering and Computer Science (DISI), University of Trento, 38123 Trento, Italy; fabrizio.granelli@unitn.it
* Correspondence: sarzo@unm.edu
† These authors contributed equally to this work.

Abstract: Recently, a multi-agent based network automation architecture has been proposed. The architecture is named multi-agent based network automation of the network management system (MANA-NMS). The architectural framework introduced atomized network functions (ANFs). ANFs should be autonomous, atomic, and intelligent agents. Such agents should be implemented as an independent decision element, using machine/deep learning (ML/DL) as an internal cognitive and reasoning part. Using these atomic and intelligent agents as a building block, a MANA-NMS can be composed using the appropriate functions. As a continuation toward implementation of the architecture MANA-NMS, this paper presents a network traffic prediction agent (NTPA) and a network traffic classification agent (NTCA) for a network traffic management system. First, an NTPA is designed and implemented using DL algorithms, i.e., long short-term memory (LSTM), gated recurrent unit (GRU), multilayer perceptrons (MLPs), and convolutional neural network (CNN) algorithms as a reasoning and cognitive part of the agent. Similarly, an NTCA is designed using decision tree (DT), K-nearest neighbors (K-NN), support vector machine (SVM), and naive Bayes (NB) as a cognitive component in the agent design. We then measure the NTPA prediction accuracy, training latency, prediction latency, and computational resource consumption. The results indicate that the LSTM-based NTPA outperforms compared to GRU, MLP, and CNN-based NTPA in terms of prediction accuracy, and prediction latency. We also evaluate the accuracy of the classifier, training latency, classification latency, and computational resource consumption of NTCA using the ML models. The performance evaluation shows that the DT-based NTCA performs the best.

Keywords: network management automation; multi-agent system; network traffic classifier; network traffic predictor; machine learning; deep learning; 5G/6G; future network architecture; service-oriented architecture

Citation: Arzo, S.T.; Akhavan, Z.; Esmaeili, M.; Devetsikiotis, M.; Granelli, F. Multi-Agent Based Traffic Prediction and Traffic Classification for Autonomic Network Management Systems for Future Networks. *Future Internet* **2022**, *14*, 230. https://doi.org/10.3390/fi14080230

Academic Editor: Ali Tizghadam

Received: 23 June 2022
Accepted: 25 July 2022
Published: 28 July 2022

1. Introduction

Traditional networks consist of several devices, including switches, routers, servers, deep packet inspection, firewalls, and hosts. These devices are traditionally implemented using hardware. A human administrator manages such small networks [1,2], through configuration and management interfaces. Typically, an administrator is necessary to configure and make the required changes. This is feasible only for relatively small size networks, albeit with limited performance and flexibility in terms of resource utilization, energy efficiency, service admission, latency, reliability, etc.

Nevertheless, within the last two decades, communication networks have tremendously evolved. It is now possible to interconnect a massive number of devices and

applications including traditional users, IoT devices, and machines. For instance, in the wireless domain, 4G and 5G have shown a multi-fold increase in the network size. The advent of IoT and other machine-to-machine communication paradigms also hugely increases the network size. The legacy network implementation has an enormous capital expenditure (CAPEX) since such networks are hardware-based and manually operated, which is inefficient and inflexible. As the network grows larger, it is increasingly difficult and expensive to manually administer its complexity, implying an increase in operational expenditure (OPEX). This is mainly because a massive number of technical personnel and experts is needed and sometimes humanly challenging and inefficient to perform, resulting in degrading network performance.

Moreover, it is expected that Beyond 5G (B5G) and 6G will increase in size and become more complex. Furthermore, as indicated in 6G deliverable [3,4], intelligence and autonomy are expected to be the core principles that drive the future networks, B5G and 6G. In this direction, an approach for in-network intelligence and automation is required to address the expected network growth as well as stringent requirements of various services. Currently, software-defined networking (SDN) and network function virtualization (NFV) are enabling network softwarization, subsequently increasing network flexibility and opening the door to autonomic networking. SDN provides a programmable network, while NFV softwarizes network functions, decoupling the hardware-based implementation to general-purpose and cloud-based deployment [5].

Autonomic networking is the process of enabling the network system to have a self-X property; X can be managing, configuring, healing, and protecting the network [6]. This was originally aimed at creating autonomic computing by IBM [1,7]. One of the reasons for having automated network management systems is the idea of dynamic network resource allocation [8,9] based on the service traffic arrival statistics. A self-managing network should be able to autonomously predict, diagnose, reconfigure, monitor, and circumvent failures while providing service processing as per the user traffic demands. This means, an autonomous network management system (ANMS) can collect, analyze, act, while adapting to the network and user dynamic and their evolving behavior [10]. Using ANMS, networks are capable of self-management and autonomous decisions both in unsupervised and supervised manner [11]. This guarantees a simplified control, management, and operation of complex and vast networks. These research works are a classic approach for network automation that we developed before the introduction of the softwarized networking, which transformed the network management, design and configurations.

Recently, the MANA-NMS architectural framework was proposed in [12,13], using a multi-agent system (MAS). Defining atomic decision-making units, the authors discussed the concept of network function atomization. Such atomic units could replace the virtual network functions (VNFS) with atomized intelligent agents. The approach is a service-oriented architectural (SOA) design for a flexible, loosely coupled, and intelligent system development for future generation networks. Moreover, it is a competing approach for microservice-based design with proactive and reactive intelligence. MANA-NMS used a MAS due to its advantage over microservices in terms of autonomy and intelligence. In the proposed ANF, the agents are autonomous with adaptive functionality in performing a given task. An ANMS can be built composing these service agents. Alternatively, the MANA-NMS framework can be taken as a splitting of a monolithic system into agents and recomposing them to obtain an autonomous system: e.g., SDN controller evolved packet core (EPC), and network management system (NMS). Currently, network function such as EPC, SDN, and NMS are monolithic systems that can be split into the smallest possible functions [14,15]. The decomposed units can be implemented as microservices [14] or agents [12]. This is an appealing paradigm in the current trend of softwarization, service function virtualization or containerization, cloudification, and orchestration. This helps deploy network functions (possibly autonomous agents) as loosely coupled, elegant, and lightweight applications in a virtualized/containerized, distributed, and cloud/edge environment with maximum re-composition freedom. Both refs. [12,13] only discussed

a theoretical model, and the performance evaluation is performed using mathematical evaluation.

A similar approach with a focus on an agent-based algorithm was proposed in [16]. The authors presented a software agent to execute a set of computation loads in parallel, which enables a bottom-up systematic organization of the overall system. The paper discussed IoT objects to be an agent that is autonomous and able to collaborate with other peer agents. However, the approach is focused on high-level objects and does not consider protocol level and function level functionalities. Moreover, it is needed to consider service design approaches in comparison with microservices showing how to incorporate ML/DL algorithms in the agent design.

In this paper, using the design guideline from [12], we implement the internal composition of an NTPA and NTCA. We briefly introduce the ML and DL algorithms that are used as the cognitive component of NTCA and NTPA. We also propose and implement possible communication interface technologies. In addition to the legacy agent communication language (ACL), we suggest Restful-API, gRPC, and Websocket. This paper could be considered a continuation toward the implementation of MANA-NMS, the automation architecture proposed in [12] and an extended version of the work in [17]. The authors in [17] only considered classification agent design with limited evaluation that does not consider the resource consumption of agents. This is crucial in that agents are mostly deployed in a containerized environment with a resource-constraint environment, especially as the complete system may need a number of agents to be instantiated.

In [12], the authors used mathematical models and evaluation and left the implementation as future work. Therefore, this article tries to implement and simulate the ML/DL-based NTPA and NTCA and evaluate critical measures, such as accuracy and latency. The benefit and advantage of designing NTPA and NTCA agents are to use these agents as a service unit in the multi-agent system. In the simulation section, an example of the complete MANA-NMS system architectures is provided, incorporating the traffic classifier agent and traffic predictor as well as their role in the overall system as an example. The contributions of this work are as follows:

- Design the internal architecture of NTPA and NTCA.
- Implement and simulate both agents.
- Evaluate the performance of NTPA and NTCA using accuracy prediction and classification, training latency, prediction and classification latency, and resource consumption, respectively.

The rest of the paper is organized as follows. Section 2 presents an overview of the network automation and enabling technologies. The prediction and classification agents' design architecture along with the agent communication technologies are discussed in Section 3.1. The implementation scenario for the proposed conceptual framework and dataset description is presented in Section 4. Section 5 presents the performance evaluation of the proposed NTPA and NTCA. Lastly, Section 6 concludes this work.

2. Overview of Network Automation and Enabling Technologies

This section provides a review of important concepts for network automation, including the applied ML/DL algorithms and multi-agent systems.

2.1. Overview of Network Management Automation

Autonomic network management is proposed as one of the solutions for the management of large and complex networks. As indicated above, it comes from autonomic computing, a manifesto started with IBM in 2001 [1,6,13]. IBM introduced the concept of self-management and self-adaptation of networks. Self-management implies that a network can make decisions on its own, while self-adaptation means that the network is aware of its environment and can adapt the performance according to the changes in the environment. The autonomic systems described by IBM are equipped with four self-management prop-

erties [13], such as self-configuration, self-optimization, self-protection, and self-healing. These properties are discussed in detail below.

- Self-configuration offers the opportunity to discover new or evolving devices on a network, and to automatically establish routes and other configurations required to seamlessly connect the device.
- Self-optimization is the ability of the network to change its settings to match its actions better with the system's user-defined purpose.
- Self-protection requires a network to protect itself from a possible attack, such as denial-of-service (DoS) attacks or the modification of firewall rules based on alleged malicious traffic.
- Self-healing is the capacity to correct, by itself, any issues that may occur, whether or not from the attempted behavior of the device.

2.2. Overview of the Multi-Agent System

Problem solving in complex environments requires distributed approaches. As a solution, *distributed artificial intelligence* (DAI) is proposed, where the system consists of many intelligent agents interacting with each other to achieve the goal [18,19]. DAI, traditionally, suggests two types of solutions. The first one is to break a problem into sub-problems. Then, micro solutions are provided to solve each sub-problem individually. The second solution is MAS, where autonomous agents cooperate with each other to provide a service.

Figure 1 illustrates the general architecture of the MAS. Similar to distributed problem solving using microservice solutions, the agent works with other agents or individually to perform a given task. The agent also measures the environment continuously to obtain updated information about the environment [20]. The core characteristics of the MAS are discussed in Section 3.1 in the context of our agent definition.

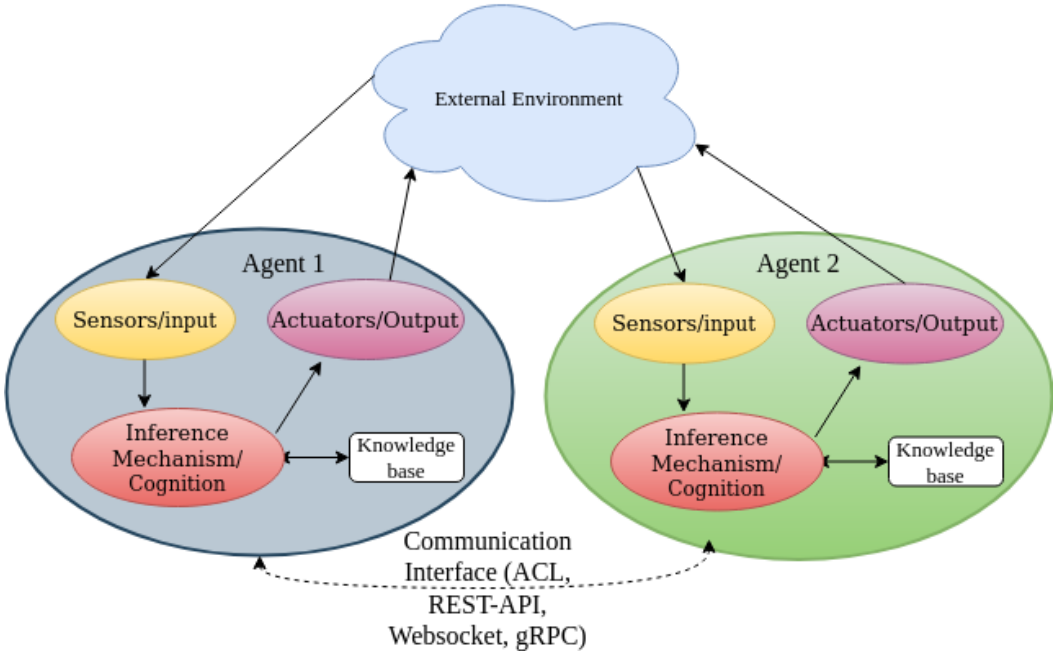

Figure 1. A simple MAS architecture.

Agent as a Decision-Making Element in Autonomic Networking

Agents have different definitions based on their applications in different environments [20]. Before defining network agents, we present a general definition incorporating all characteristics of the agents. An agent is an object that observes an environment and

uses various parameters to execute an appropriate action at each step and achieve the goal. This definition is made of four keywords that are further explained below.

1. *Object*: This refers to the kind of agent in different contexts, for example, a software agent, a hardware agent, etc.
2. *Environment*: The environment receives the agent's actions and updates itself accordingly. As a result, the agent observes the changes and adjusts its actions.
3. *Parameters*: These are the information that agents need to know to take the appropriate action.
4. *Action*: The agent executes actions that result in changes in the environment. The action space can be continuous or discrete.

Each agent decides to perform its action with regard to time and resource constraints [20]. In order to perform a good action, the agent needs to interact with the environment as well as communicate with other agents through ACL, REST-API, gRPC, Websocket, as illustrated in Figure 1 to learn about the information. This information contains the previous agent's experience from interacting with the environment. This helps the agent with its decision-making process [21].

In this work, we present the MAS as an automated network management system, where the agents are known as the network decision elements (DEs).

2.3. Application of Traditional Machine Learning and Deep Learning as a Cognitive Component of Agents

As indicated in the Introduction section, and will be further discussed in the next section, one of the internal elements of an agent is the cognitive or reasoning component which enables the agent to perceive, reason, and decide. This would enable designing agents to be the smallest atomic element with autonomous and intelligent capability in performing a given task. Using such elements, we build the complete system. This is in line with the future service-based network design in a softwarized, containerized, and intelligent network environment. Incorporating in-network intelligence in the network functions is the prime target in 6G [3,4,22,23]. Hence, we require to incorporate ML/DL to create an intelligent agent, such as NTPA and NTCA.

In the next paragraph, we discuss the DL definition and its advantages and disadvantages over ML methods.

DL is a sub-field of ML that enables learning and decision capability in the agent, training the agent to learn like a human brain. DL statistics and predictive modeling enable agents to process massive datasets. It uses a layered structure and applies a nonlinear transformation to its input iteratively to learn the features and create a statistical model as output. There are several DL techniques that help with creating accurate predictive models from a large amount of unlabeled and unstructured data. We describe some of the techniques as follows:

- Learning rate decay: a hyperparameter in DL settings controlling the learning process of the model every time the weights are updated to avoid unstable training and a long training process.
- Transfer learning: this method uses an existing network trained previously. It feeds the new data along with previously unknown samples to the existing network, and once the adjustments are made, it feeds the new samples to be classified. This method requires much less data, thus reducing the computation time.
- Training from scratch: This method requires a large amount of labeled data to train the network. However, this technique may be impractical for some applications, as it needs a tremendous amount of data to be collected and makes the training process last too long.
- Dropout: Training a neural network on supervised learning tasks may cause overfitting. To avoid this problem, the dropout method drops units and their connection randomly from the neural network.

Although various neural network architectures have been proposed, such as RNN, CNN, and feedforward neural networks, they all function in similar ways. They benefit from a layered architecture, where input is given and the model figures out itself whether it has made the right decision about the data element by a trial-and-error process. Next, we list some advantages and disadvantages of the DL methods and compare them with ML algorithms.

Unlike the general ML, DL does not need to be given the feature set to make decisions. It learns the features by interacting with the environment and builds the feature set incrementally. Therefore, DL methods do not require the programmer to specify the features that the computer should be looking for. In other words, DL is able to learn without human intervention. While this makes the DL methods take much time to train, they are much quicker in testing than the ML algorithms. The ML test time increases as the data size grows. However, there are some issues with DL models, making data scientists choose traditional ML over DL. Its neural networks rely on the trial-and-error process. In any case, we used different ML and DL models for the NTCA and NTPA design and tested their performance.

2.3.1. LSTM-Based Network Traffic Predictor Agent

LSTM is an improved form of the traditional recurrent neural networks (RNNs) addressing the vanishing and exploding gradients problems using connected memory blocks in the layers. Each block consists of multiple cells containing three units: the input, output, and forget gates. Unlike the standard RNNs, this block design enables LSTM to capture long-term temporal dependencies in sequence learning problems. The network interacts with the cells through their gates. The role of the input and output gates is to achieve a constant error flow and avoid the irrelevant memory contact, respectively. However, an unstable error flow is observed through the backpropagation in existing RNNs [24,25].

2.3.2. GRU-Based Network Traffic Predictor Agent

GRU is a variant of LSTM and the improved version of standard RNN [24]. A GRU cell consists of two gates: update gate and reset gate to solve the vanishing gradient problem in recurrent neural networks. First, the reset gate is used to filter out the irrelevant information from the past time steps and create the memory content at the current time step. Next, the update gate is used to create the final content of the memory by deciding how much information still needs to be kept and passed to the output layer [24,25]. We describe the usage of the reset and update gates with mathematical notation below: first, the reset gate (similar to the forget gate in the LSTM setup) filters out the irrelevant past information as shown in Equation (1).

$$m'(t) = tanh(Wx_t + r_t \odot W'm_{t-1}) \quad (1)$$

where $m'(t)$ represents the current memory content. The element-wise product between the reset gate and weighted previous memory content ($r_t \odot W'm_{t-1}$) determines what information needs to be removed from the past time steps.

Next, the update gate determines what information from the current memory content and the previous time steps need to be passed through the network for computing the final result as shown in Equation (2), where u_t is the update gate and controls the memory content.

$$m_t = u_t \odot m_{t-1} + (1 - u_t) \odot m'_t \quad (2)$$

2.3.3. MLP-Based Network Traffic Predictor Agent

MLP is a fully connected feedforward artificial neural network composed of many perceptrons (neurons) that are organized into at least three layers: an input layer, a hidden layer, and an output layer [26,27]. Each neuron in the layer except the input layer utilizes a non-linear activation function to learn the non-linear data. The learning process in an MLP is described as, when the data are given to the MLP, they go through their layers. Each node

in the layer computes its error by comparing its output and the expected result. The goal is to minimize the error and then adjust the node weights [26]. The error is calculated by the following equation:

$$e_j(n) = d_j(n) - s_j(n) \tag{3}$$

where d_j is the target value and s_j is the value generated by the node j. The error for processing data point n by the nodes is obtained by the equation below:

$$E(n) = \frac{1}{2}\sum_j e_j^2(n) \tag{4}$$

Next, the weight adjustment is calculated using the gradient descent:

$$\Delta w_{ji}(n) = -\alpha \frac{\partial E(n)}{\partial a_j(n)} s_i(n) \tag{5}$$

where s_i is the output of the previous node, α is the learning rate to ensure that the weight corrections converge quickly, and a_j is the induced local field or activation potential.

2.3.4. CNN-Based Network Traffic Predictor Agent

CNN is a kind of feed-forward neural network that has major application in image classification [27]. It has also applications in time series, where we can utilize it for network traffic prediction. It consists of an input layer, convolution layer, pooling layer, fully connected layer (as hidden layers), and an output layer. The convolution layer is a good substitute for a fully connected layer, as it is scalable to massive datasets. The role of the convolution layer is to reduce the dimension of the input images without losing critical features, allowing the network to be deeper and learn more easily.

A part of a convolution layer is the kernel/filter that has smaller dimensions than the input image and moves over the image and performs matrix multiplication until the entire image is traversed. This process produces a convoluted feature output, extracting high-level features of the image, such as edges. The next layer is called the pooling layer. The pooling layer is in charge of reducing the spatial size of the convoluted feature output even more, using two operations: max pooling and average pooling. Max pooling returns the maximum value of the calculated convoluted features corresponding to a portion of the image. Average pooling returns the average of all values generated by the kernel/filter. Finally, a fully connected layer (multilayer perceptrons) is added to classify the images using the softmax classification technique.

2.3.5. Decision-Tree-Based Network Traffic Classifier Agent

We presented four possible ML solutions for the NTCA design in [14] in more detail. For the readers' convenience, we briefly discuss these traffic classifier ML algorithms.

A non-parametric-based supervised learning approach utilized for classification is a DT. DTs learn from knowledge to approximate a sine curve using a sequence of if-then-else decisions. The longer the tree, the more complex the rule of decision in fitting a given model [28,29]. Figure 2 illustrates the DT diagram.

2.3.6. Naive Bayes Based Network Traffic Classifier Agent

NB is a classification technique that, given a finite set of class labels, assigns the labels to the data samples using a probability distribution [30]. It assumes each feature value independently builds the probability specifying the data sample's class regardless of the correlations between the features.

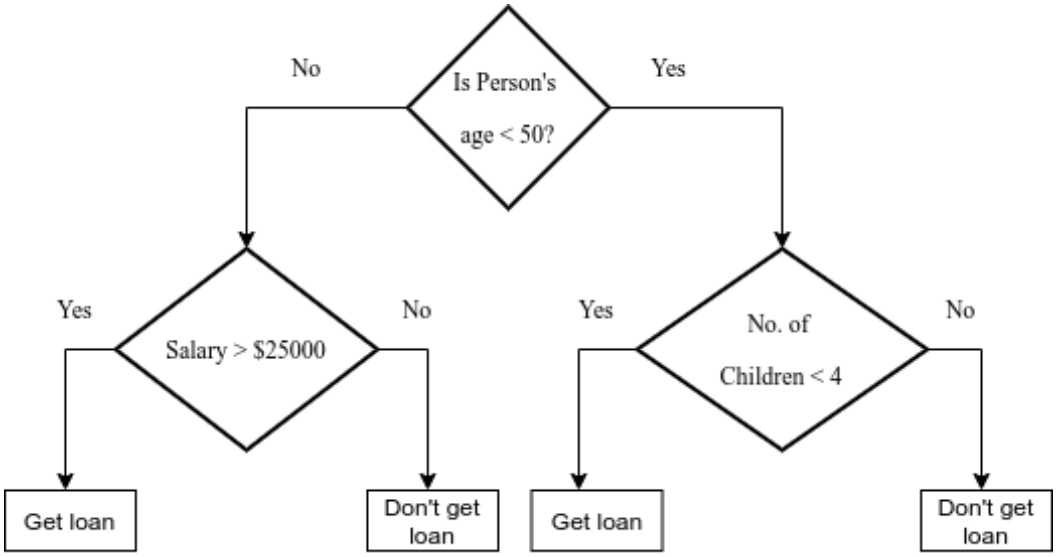

Figure 2. An example of a DT.

2.3.7. Support Vector Machine Based Network Traffic Classifier Agent

SVM attempts to find a hyperplane (decision area), having the maximum distance between the two categories of data points in high or infinite dimensional space [29–31]. Figure 3 illustrates the SVM diagram.

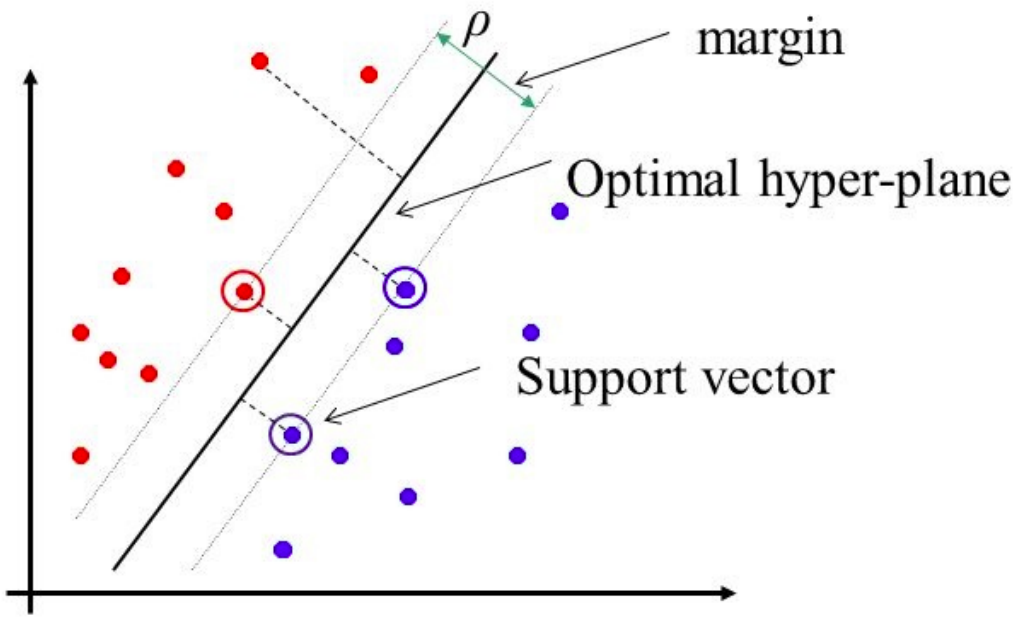

Figure 3. An Illustration of a hyperplane and support vectors.

2.3.8. K-Nearest Neighbors Based Network Traffic Classifier Agent

K-NN classifies the data point based on the most common class among its k-nearest neighbors [29]. Figure 4 shows how K-NN can help one to identify the class of a new data point.

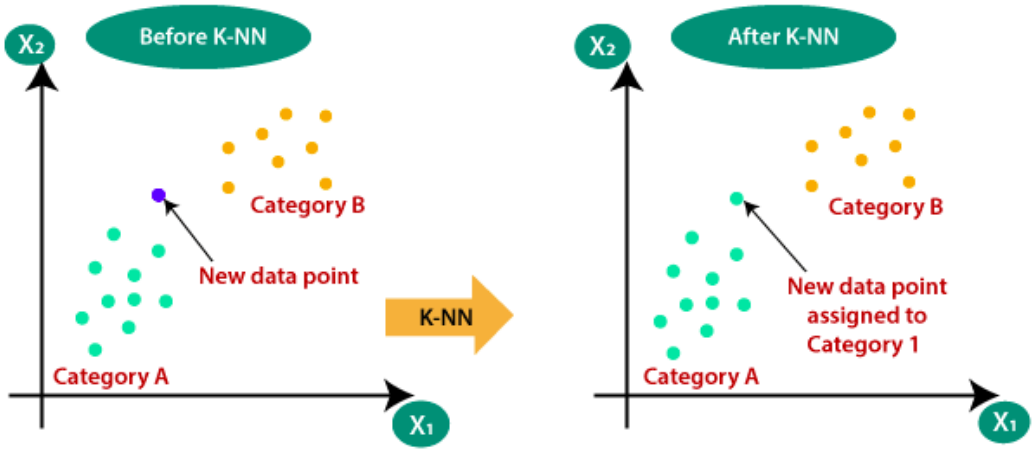

Figure 4. An illustration of the K-NN classifier algorithm in action.

3. Proposed Network Traffic Predictor and Traffic Classification Agents

In this section, we discuss the proposed NTPA and NTCA frameworks. First, let us redefine and contextualize the agents. The agent has input, output, and communication tools to observe the environment, provide the output or execute the action, and communicate with other existing agents in the MAS. Once an agent is employed in a given environment and it observes the environmental variables, it can make decisions. The decision is based on the agent's final goals, which could be to predict the incoming traffic and classify them accordingly. By measuring the incoming traffic over a given link, it could analyze and use it for load balancing, resource slicing, or resource allocation. The important characteristics of an agent are as follows:

- Situatedness: that means agents use sensors or perception capabilities as an interaction with the environment, and it uses actuators to perform actions on the environment. In the NTPA and NTCA contexts, it means taking incoming data as input and providing prediction and classification results as output, respectively.
- Autonomy: the agents' internal state is protected from any external disturbance from other agents in the MAS. NTPA and NTCA are capable of performing their respective function autonomously without any external support.
- Inferential capability: this enables agents to work on an exact goal until it is achieved. Agents can analyze the available data for decision making, such as network traffic prediction and traffic class determination.
- Responsiveness: defined as the ability of an agent to perceive the environment and perform on it with minimal latency, which is important in real-time processes. Agents should be designed such that the task execution (prediction and classification in case of NTPA and NTCA) must be performed with a strict time delay.
- Pro-activeness: means agents should take advantage of particular opportunities that aim to improve their action performance to dynamically adapt to the changes in the surrounding environment of the network. By analyzing the historical data, it could predict the amount of traffic that comes in a given link. Using this knowledge, it pro-actively allocates the required resources, such as bandwidth.
- Social behavior: the agent's decision should not be affected by any external interference, either from human interactions or other agents in the system. NTPA and NTCA should be capable of performing independently but also could communicate and collaborate to perform a sequence of tasks, such as classification followed by prediction or vice versa. For instance, NTCA could classify traffic into various classes and provide the output to the predictor. Based on the classes, the predictor could predict the amount of incoming traffic of a given time in the coming hours or days.

3.1. Proposed Network Traffic Predictor Agent Architectural Framework

A given NTPA is equipped with an "input unit" that enables it to receive a network traffic processing request and network state sampling/measuring points. Furthermore, NTPA is equipped with several facts that are model features that could be the amount of incoming network traffic, the available bandwidth in a given link, IP addresses, source and destination ports, and labels (protocols) [20]. The main target to focus on is the "cognitive/reasoning unit". It is the brain of NTPA that enables it to have reasoning capability, which is typically a DL model. The "planning strategy" component of NTPA is the steps for the action to be performed in accomplishing the required processing. Another component of NTPA is a validation unit. It is composed of validation rules for the final decision, for example, knowing the amount of arriving network traffic of given traffic classes that helps make decisions. Finally, we have an "output or action" unit. This is the final outcome that could be a prediction result to be sent to the other agents. Figure 5 (bottom) shows the building blocks of an NTPA.

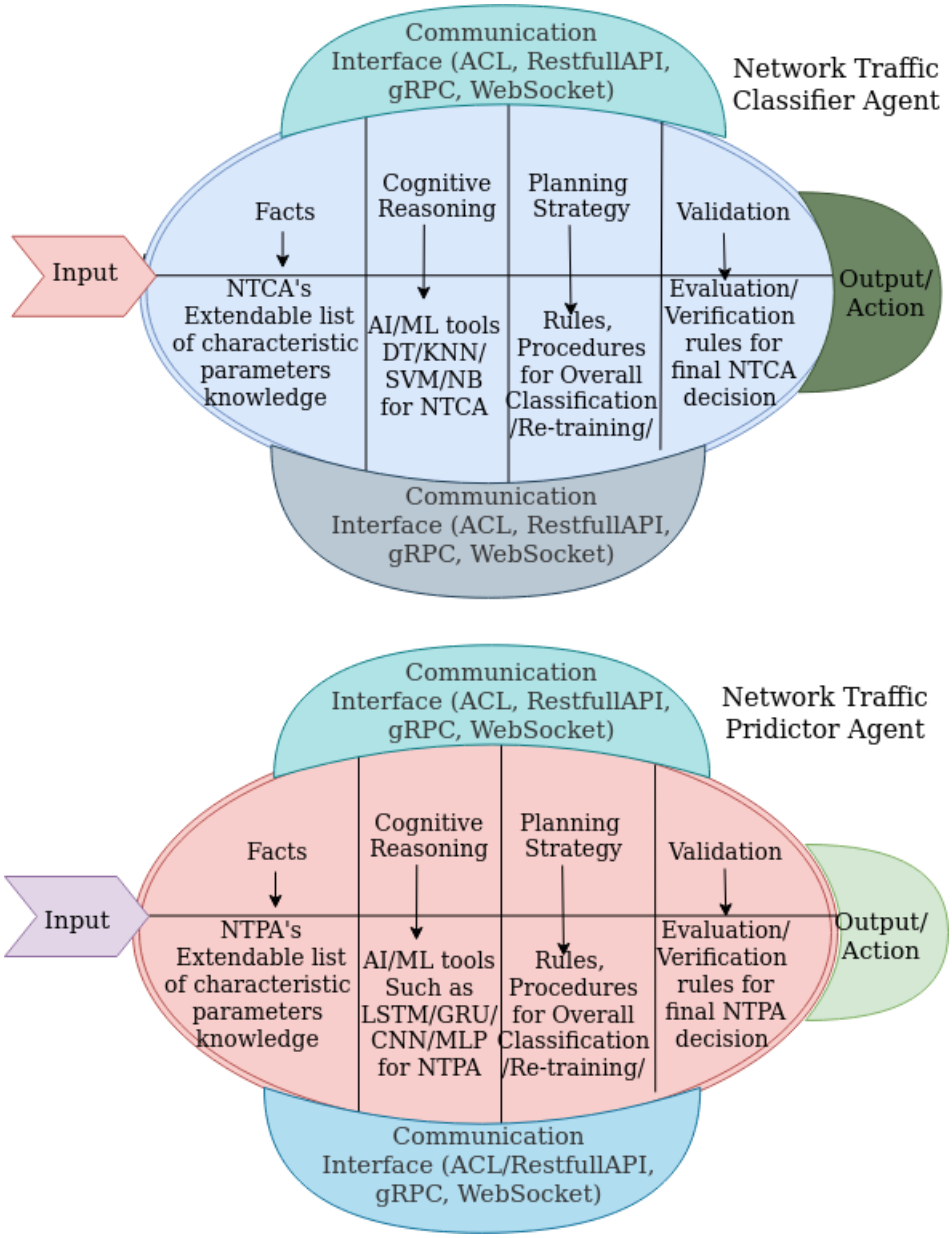

Figure 5. Network traffic classifier agent (**top**) and network traffic predictor (**bottom**) agent architecture.

3.2. Proposed Network Traffic Classifier Agent Architectural Framework

The general architecture of NTCA is similar to NTPA. However, the actual values, parameters, and algorithms used are different. That means the input parameter, knowledge base, cognition algorithms, validation technique and the output for NTCA are different from NTPA. In general, taking the network traffic as input, NTCA classifies them into the classes to which the traffic belongs. NTCA helps other network agents in making network decisions, including QoS provisioning and monitoring based on SLA, resource provisioning, and network slicing. The NTCA design and implementation framework is depicted in Figure 5 (top).

In particular, simpler than NTPA, NTCA has an "input-unit" that can be used in receiving a service request and network state sampling/measuring points. In addition, NTCA has facts, i.e., incoming traffic features, which could be the size of a packet, source and destination IP addresses, protocols, etc. The "cognition/reasoning unit" is the component that gives the central intelligence for NTCA. It uses it to identify and classify traffic. The "planning strategy unit" is the steps/procedures in providing the class of the traffic. Another component of NTCA is the validation unit. It is composed of validation rules for the final classification decision, i.e., knowing the amount and type of arriving network traffic of given traffic classes that helps make decisions. Finally, the remaining NTCA component is an "output unit", which could be a classification result to be sent to other agents.

3.3. Communication Interface between Agents

The communication interface enables the creation of an agent chain for collaboration and cooperation to create a complete MANA-NMS. Agents could be deployed in a containerized and distributed environment that communicate based on open-source communication interfaces. The standard communication interface for agents is ACL [32]. ACL enables a complex knowledge and information exchange mechanism which includes facts and a knowledge base, the agent's goal, strategy, plans, and outputs. We used ACL as a communication interface in our simulation. However, other generic communication interfaces were tested in our work in [14]. These are REST, WebSocket, and gRPC. Each of them has its pros and cons in terms of latency for web-based communicating intelligent agents.

4. Implementation Scenario and Conceptual Framework

We used the osBrain [33] library for the design and implementation of NTPA and NTCA. osBrain is an agent simulation environment implemented using Python.

We provide the hardware configuration information shown in Table 1, as it affects the prediction results. Therefore, the same hardware was used to run the LSTM, GRU, CNN, and MLP models.

Agents within the MAS are running independently as system processes (uses multiprocessing). Since osBrain uses ZeroMQ [34], agents communicate using zeroMQ. zeroMQ is a messaging library for flexible and efficient communication between agents. It is an open-source universal messaging library that uses asynchronous communication between agents in the MAS. The basic communication patterns used by osBrain are *Push–Pull*, *Request–Reply* and *Publish–Subscribe*.

Table 1. Hardware configuration.

Hardware	Capacity
Memory	16 GB 2133 MHz LPDDR3
Processor	2.7 GHz Intel Core i7
Storage	2 Terabytes

4.1. Implementation Conceptual Framework for MANA-NMS Using NTPA

NTPA was implemented in a MANA-NMS framework as a proof of concept. The MANA-NMS architecture is depicted in Figure 6, showing the integration of NTPA along with the dispatcher, the audit log agents, and other service processing agents. NTPA mainly uses LSTM. However, we also used GRU, CNN, and MLP algorithms to compare the performance against LSTM-based NTPA. MANA-NMS is assumed to be deployed in an edge/cloud data center, where the incoming traffic is collected at the gateway. We assumed a Poisson arrival process for the service arrival into the system. The Poisson arrival process is considered with a 1st degree approximation of the traffic arrival. However, other models could be used, such as fractal process arrival.

We collected the amount of incoming traffic to observe the service request distribution in time. The data were collected for 1-year period. To store and manage the history of traffic data for future retraining of agents, a centralized database can be used. Alternatively, it could be possible to provide the agent with database management capability. In the agent interaction dynamics, agents are responsible for pulling results from other agents and logging their actions. NTPA uses the collected dataset to train or retrain its' internal cognitive reasoning algorithms. The training of the DL component of NTPA could be performed centrally (remotely) or locally (at the agent). The training process is performed periodically.

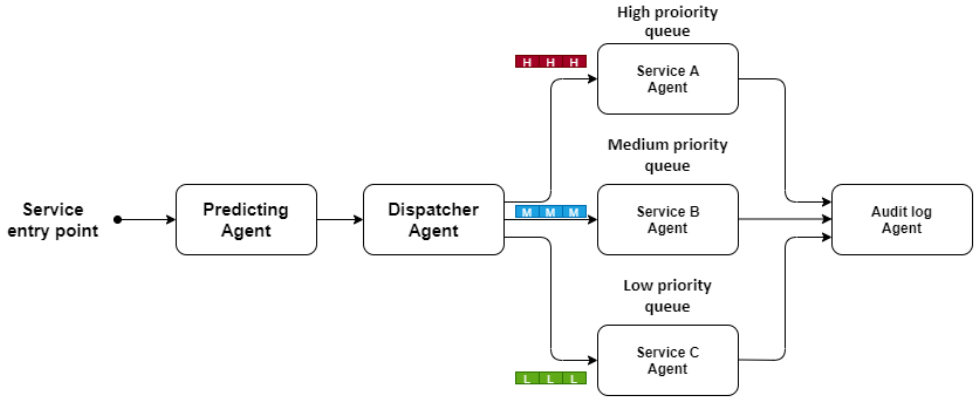

Figure 6. Simple architecture for the MANA-NMS.

After training is complete, NTPA predicts the incoming traffic for the next hours/days. The predicted data indicate the real incoming traffic demands at a given time.

By using the NTPA prediction traffic values, the resources/agents are dynamically allocated, creating an agent chain for that particular period. In other words, the predicted value is used to instantiate the number of service agents by the orchestration agent through a request communication protocol channel. Then the orchestration agent creates the required agent chain that is considered the network resources to perform the required service execution. This can save a lot of unused resources at some time of the day. By doing so, we optimize the performance of the network, reduce the running cost, and increase the network efficiency.

Moreover, we have a dispatcher agent (event handler agent), whose goals are to add the incoming daily traffic data/services in a FIFO queue, divide the traffic into three different queues consulting the NTCA and based on the task/service priority, whether high, medium, or low, and communicate through three different communication channels to the service agents (A, B, C).

The scheduling agent schedules the incoming tasks using the first-in-first-out (FIFO) queuing algorithm while using the classifying agent to classify the tasks into high, medium, or low priority classes. This represents three main application areas in 5G network ultra-reliable machine-type communication (uMTC), massive machine-type communication (mMTC) or extreme mobile broadband (xMBB), respectively. The service agents execute tasks scheduled in three different queues depending on the task priority. They represent the complementary service agents in the MANA-NMS. Three types of service agents are

assumed to serve 5G/B5G/6G traffic, which are (a) service agent type A serving the high priority queue (uMTC), (b) service agent type B serving the medium priority queue (mMTC), and (c) service agent type C serving the low priority queue (xMBB). Finally, once the service agents finish their processes, the audit-log agent will pull the results through three separate communication channels and save them.

4.2. Implementation Conceptual Framework for MANA-NMS Using NTCA

Similar to the NTPA, we used the NTCA design guideline presented above, as depicted in Figure 5, for the agent implementation. Using the input traffic, NTCA classifies the incoming network traffic into the appropriate classes. NTCA helps with important decisions in network management, for instance, for QoS provisioning, resource provisioning, dynamic network slicing, etc. In performing autonomous service management and provisioning, such as resources provisioning and task scheduling, it is necessary to classify all incoming network traffic as per the protocol to which it belongs to. The protocols indicate the application or category that the traffic/user or service belongs to. The NTCA uses an ML algorithm as an internal cognitive part to perform the required traffic classification.

Supervised ML algorithms are employed for the NTCA implementation. The supervised ML uses a labeled (protocols or port numbers) network traffic dataset. The supervised learning models used are K-NN, DT, SVM, and NB, and their performance is compared. We also assume that the NTCA can collect incoming network traffic and store it as a historic dataset. This dataset will be used by the agent to label and utilize it in training and re-training the internal cognitive component. The assumption is that the dataset will be stored in a centralized/distributed database: it could be internal or through the use of other database management agents. In our case, NTCA can handle the dataset and enable the dataset to retrieve it when needed.

In the NTCA operation, the traffic dataset is collected using Wireshark in real time. About 100,000 entries are collected while surfing the internet for about an hour.

The data are saved in CSV format and used in the feature extraction and feature selection stages. The extracted features are the source and destination IP address, packet size, and protocol. The main goal is to demonstrate the implementation of an NTCA before using it to recompose a MANA-NMS system.

NTCA is a building block in the MANA-NMS framework as depicted in Figure 7. The NTCA collects the incoming network traffic and classifies them into three different classes. Similar to the NTPA implementation, we assumed high priority (HP), for uMTC service types, medium priority (MP) for mMTC service types, and low priority (LP) for xMBB service types. After the network traffic is classified, it is sent to an event handler (task manager/dispatcher). The event handler then decides where to schedule the traffic relaying based on the scheduling policy. A FIFO policy is used in the case of traffic belonging to the same classes. Then a task is scheduled onto the appropriate service agent to be executed.

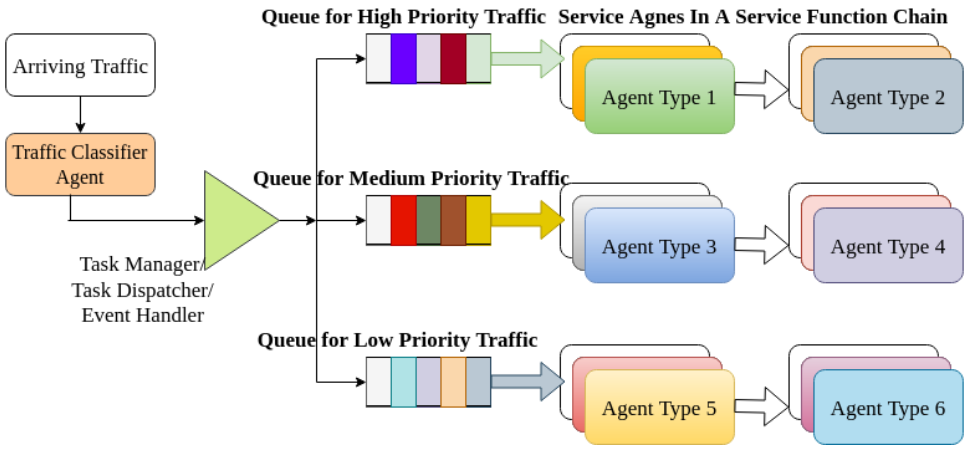

Figure 7. Implemented MANA-NMS scenario.

4.2.1. Job/Task/Event Handler and Queuing Scheduling

A task (event) manager agent is another type of agent in a given MANA-NMS architecture. It receives the classified network traffic from NTCA. We assumed every class of the network traffic as a task requiring execution in the service processing agent. The task manager decides which service processing agent should handle the service. This decision is performed using a scheduling policy or intelligent scheduling agent.

4.2.2. Scheduling Policy

"Join Class-Related Queue" is used for scheduling. Scheduling is a policy that sends tasks to the queues of the appropriate service processing agent. The agent should be subscribed to that particular task execution. For instance, when an HP class job is sent by the event manager, it will be placed in the queue of a particular service agent that is allocated to process an HP task.

4.2.3. Queuing Model Employed

We assumed an M/M/1 Markovian model for the queue. The queuing policy is assumed to be FIFO similar to the NTPA scheduling policy. In general, M/M/1 means that the system has a Poisson arrival process with an arrival rate of λ, an exponential service time distribution with service time μ, and one server. Jobs arrive at the server's queue at independent exponentially distributed time intervals with mean $1/\lambda$, they join the queue, and once in the head of the queue, they get served by the server whose service time per job is independent and exponentially distributed with parameter $1/\mu$. A task is scheduled into the appropriate queue when the task manager/event handler releases it. We assumed three different queues that are associated with a particular service agent chain, which indicates the three classes of services to be scheduled in the appropriate agent chain.

5. Performance Evaluation

This section presents the performance evaluation results of NTPA and NTCA equipped with different DL and ML models, respectively. We used metrics such as prediction and classification accuracy, training and predicting latency, and memory and CPU utilization to evaluate the performance. Therefore, at the end of this section, we will able to identify appropriate DL/ML-based NTPA and NTCA for different prediction/classification network applications with respect to their time and resource constraints.

Table 2 shows the DL models parameters used in the experiments.

Table 2. Configuration parameters.

Parameter Name	Deep Learning Models			
	LSTM	**GRU**	**CNN**	**MLP**
Learning Rate	0.1	0.3	0.2	0.01
Validation-split	0.1	0.1	0.1	0.1
Epochs	20	20	20	20
Dropout	0.2	0.1	0.2	0.2
Momentum	0.9	0.8	0.7	0.9
Hidden-layer	1	1	1	1

5.1. Accuracy of NTPA Using Different DL Models

We used the sliding window technique to train our DL models and cross validation to avoid overfitting with a validation split of 0.1 as mentioned in Table 2 along with other training parameters. The validation split determines the fraction of training data used for validation. The validation data are not used for training but evaluate the loss at each epoch.

Figure 8 illustrates the performance of the different Deep learning models, where the x-axis is day and the y-axis is traffic (terabit per second (Tbps)). In our experiment, we predicted traffic over different days with different DL models (GRU, CNN, MLP and LSTM) and took the average of the traffic and compared it with real traffic. Figure 8 shows the LSTM, GRU, MLP, and CNN based NTPA prediction results against the actual network traffic. As shown in the figure, there is no significant difference between their prediction results, while MLP predicts the network traffic differently. Overall, LSTM outperforms other models.

To measure the accuracy for different models, we used two methods: root mean square error (RMSE) and mean absolute percentage error (MAPE). RMSE is a measure that evaluates how good or bad the prediction is, using the Euclidean distance by calculating the difference between the true value and prediction. Additionally, MAPE is a different method that measures the prediction accuracy as a ratio. RMSE and MAPE are expressed as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \parallel p(i) - \hat{p}(i) \parallel^2}{N}} \tag{6}$$

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{V_i - P_i}{V_i} \right| \tag{7}$$

where in Equation (6), N is the number of data points, $p(i)$ is the measured true value, and $\hat{p}(i)$ is the predicted value. In Equation (7), n is the number of fitted points, V_i is the actual value, and P_i is the predicted value.

Figures 9 and 10 demonstrate the accuracy for DL models with respect to a different dataset size. We calculated the averages for one week, one month, three months, six months, and one year for the MSE and MAPE errors and compared different models. As shown in the figures, LSTM is the most accurate model to predict the network traffic. Additionally, as the training dataset size grows, the error value decreases. In other words, training the NTPA with larger datasets provides higher prediction accuracy. The RMSE error values for 1 year long dataset as shown in Figure 9, for LSTM, GRU, CNN, and MLP are 0.03, 0.06, 0.2, and 5.7, respectively. Moreover, the MAPE error values for 1 year long dataset as shown in Figure 10, for LSTM, GRU, CNN, and MLP are 0.24, 0.44, 0.77, and 7.17, respectively. Both figures indicate that LSTM has the most accurate prediction performance.

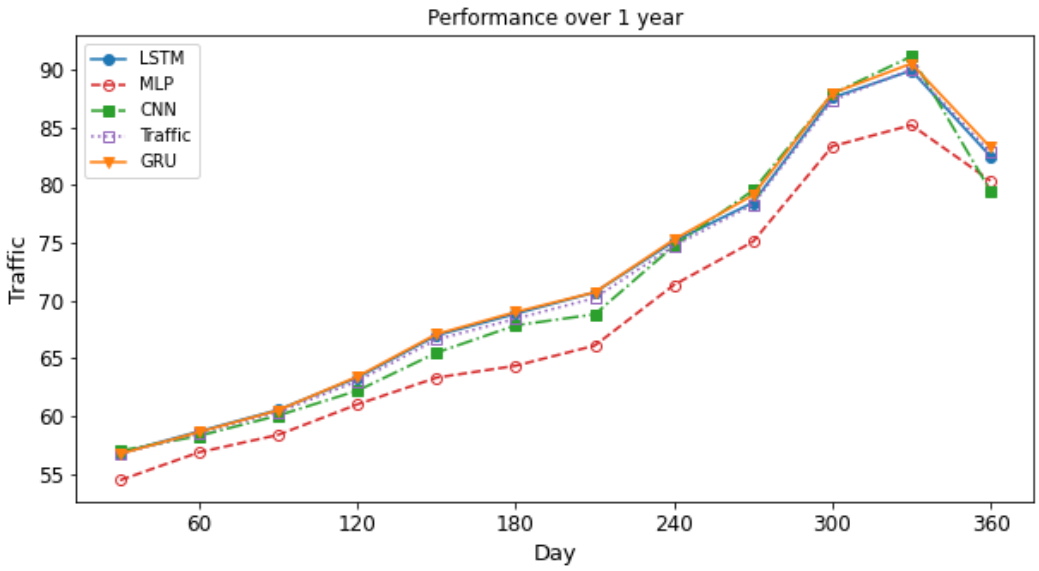

Figure 8. Different DL models performance comparison.

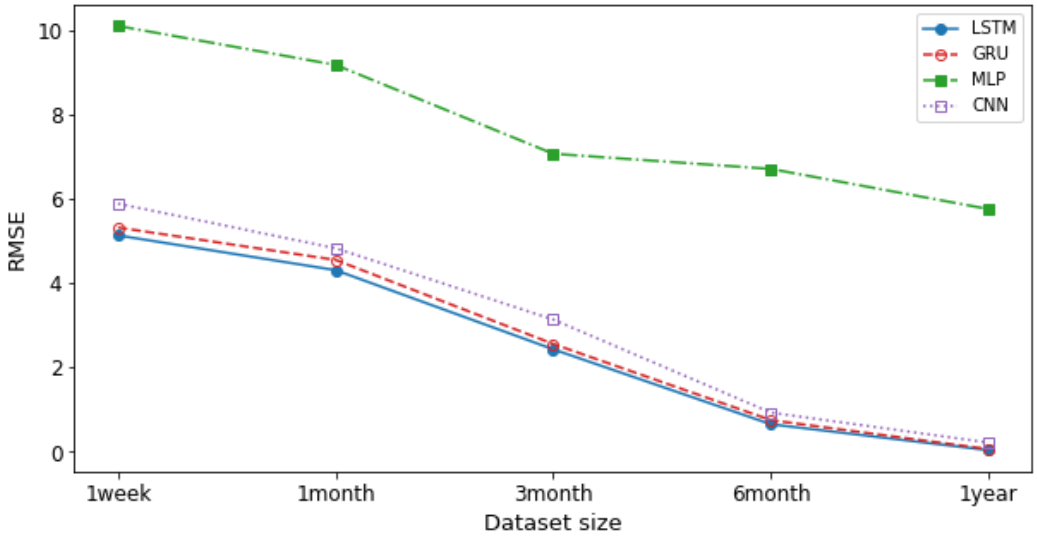

Figure 9. RMSE for different dataset size.

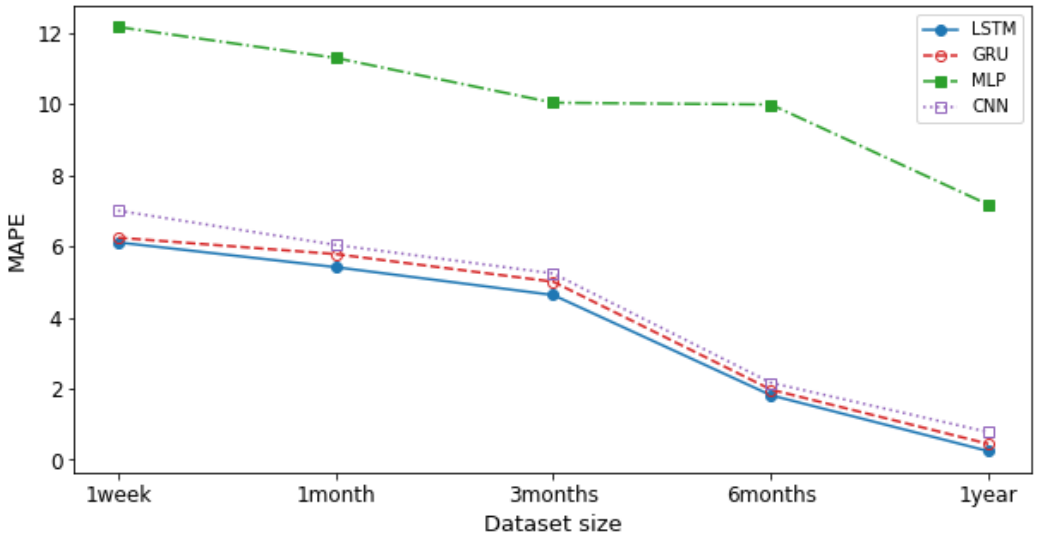

Figure 10. MAPE for different dataset size.

5.2. Training Latency and Predicting Latency of NTPA Using Different DL Models

Figure 11 shows the comparison of training latency for the LSTM, GRU, CNN, MLP models. Training latency is represented as the amount of time taken in training an NTPA. The training latency was measured over different dataset size. According to the figure, we see that CNN has the lowest training latency compared to other models. However, GRU has the highest training latency. The average training latency values for 1 year long dataset over 20 training trials for the CNN, MLP, LSTM, and GRU models are 74.036 s, 120.873 s, 269.647 s, and 291.274 s, respectively. In addition to the training latency measure, we assessed the predicting latency. Predicting latency is defined as the amount of time taken to predict the traffic at a previously unknown time slot. As shown in Figure 12, the predicting latency was evaluated over a simulation period of one hour. On average, the LSTM, MLP, CNN, and GRU NTPA designs have predicting latency values of 0.769 µs, 0.834 µs, 1.025 µs, and 1.523 µs, respectively.

The result shows that NTPA takes longer to be trained; however, it takes a shorter amount of time to predict, compared to NTCA. This confirms the advantage of the DL methods over ML methods, where a DL model is much quicker in testing, while the ML model's testing time will increase as the dataset size grows as mentioned in Section 2.3.

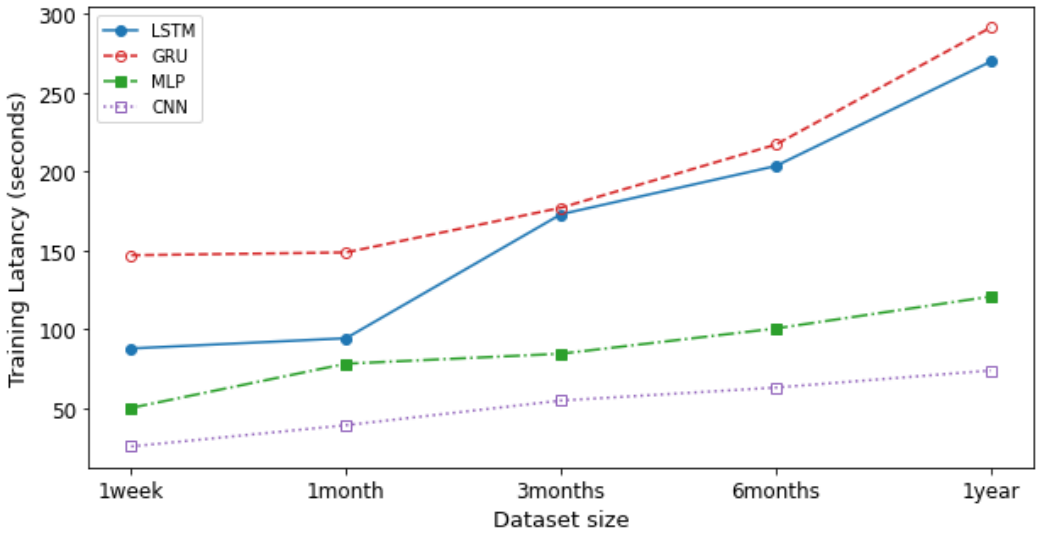

Figure 11. Comparison of NTPA training latency.

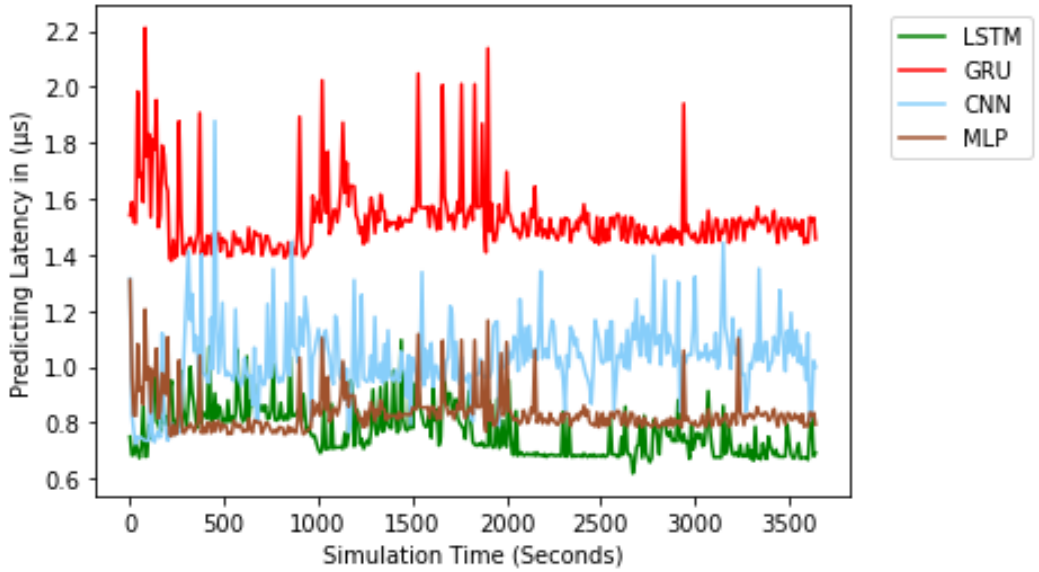

Figure 12. Comparison of NTPA predicting latency.

5.3. Computing Resource Utilization of NTPA Using Different DL Models

5.3.1. Memory Usage

We examined the memory usage (in MB) of DL models. As shown in Figure 13, we observe that the GRU-based NTPA has the highest and the CNN-based NTPA has the lowest memory usage compared to other models.

5.3.2. CPU Usage

We also examined the CPU usage of DL models. The result is shown in Figure 14. We observe that LSTM has the lowest CPU usage compared to other models. Additionally, as seen in the figure, the CPU usage of GRU is the highest.

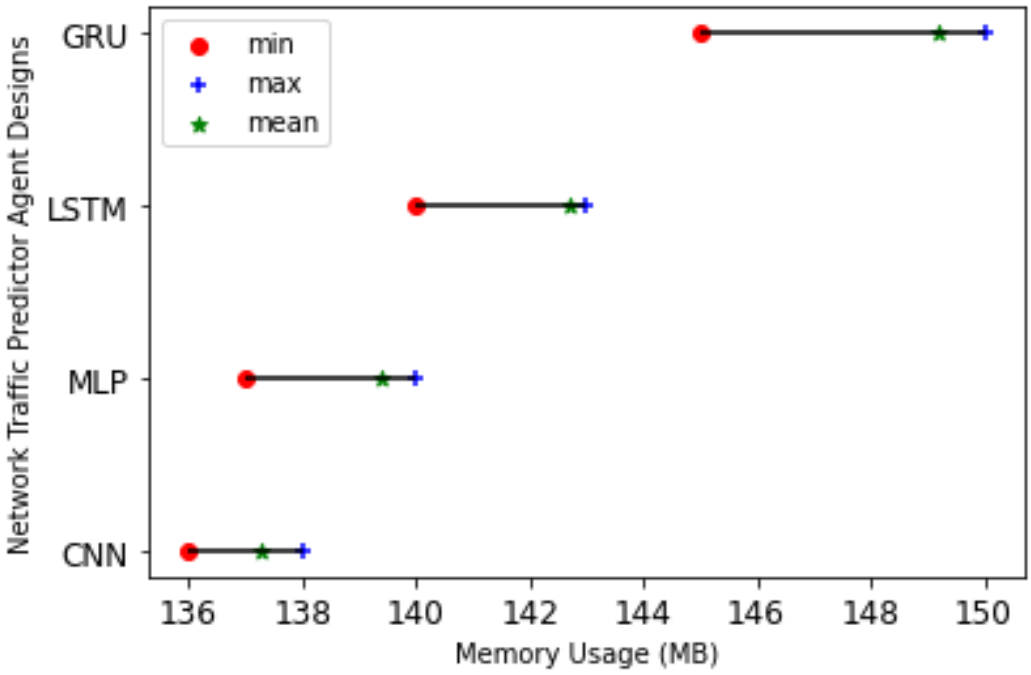

Figure 13. Comparison of NTPA memory usage.

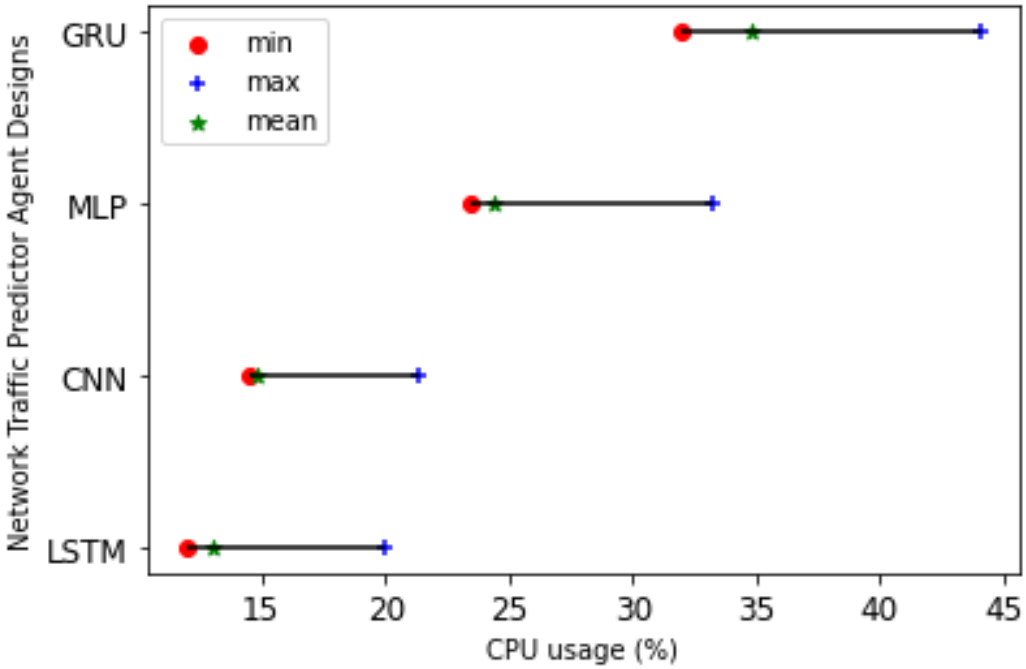

Figure 14. Comparison of NTPA CPU usage.

5.4. Accuracy of NTCA Using Different ML Models

We evaluated our ML algorithm for the NTCA designs. We split the collected data into training (80%) and test (20%) sets. The classification accuracy is considered by various factors, such as the dataset size and the number of features.

For the sake of generating a generic result that is consistent across time and multiple trials in a given environment for a given model, we ran the NTCA algorithms several times, running our models 10 times. We calculated the percentage of error made by the classifier agent for a number of classification trials. We evaluated the mean and variance in terms of accuracy for different NTCA using various ML models. The result is presented in Table 3.

According to the table, we can understand that the DT-based NTCA is the most robust and accurate of all the classifier agents. This is because it has the smallest variance while the algorithm is run multiple times.

Table 3. NTCA classification accuracy.

k_{th} Run of Algorithm	Classifier Agent Designs			
	K-NN (%)	Decision Tree (%)	SVM (%)	Naive Bayes (%)
1	98.95	99.6	94.77	88.01
2	98.87	99.44	94.45	88.01
3	98.79	99.52	94.53	89.14
4	98.79	99.6	94.53	87.61
5	98.63	99.44	95.17	88.42
6	98.63	99.52	95.09	88.09
7	98.63	99.36	94.21	87.77
8	98.87	99.68	94.85	88.9
9	98.87	99.44	94.93	87.93
10	98.95	99.44	94.93	88.74
Mean	98.798	99.504	94.746	88.262
Variance	0.01628	0.00967	0.09394	0.26282

5.5. Classification and Training Latency of NTCA Using Different ML Models

Training latency refers to the delay incurred in training a given ML model.

The mean training latencies of 154.3, 154.9, 155.1, and 162.2 ms were measured for the DT-, K-NN-, NB-, and SVM-based NTCAs, respectively. It is worth mentioning that the DT-based NTCA design exhibits the lowest average training latency with 154.3 ms, as shown in Figure 15. We observe that NB, DT, and K-NN models train faster than the SVM-based NTCA model.

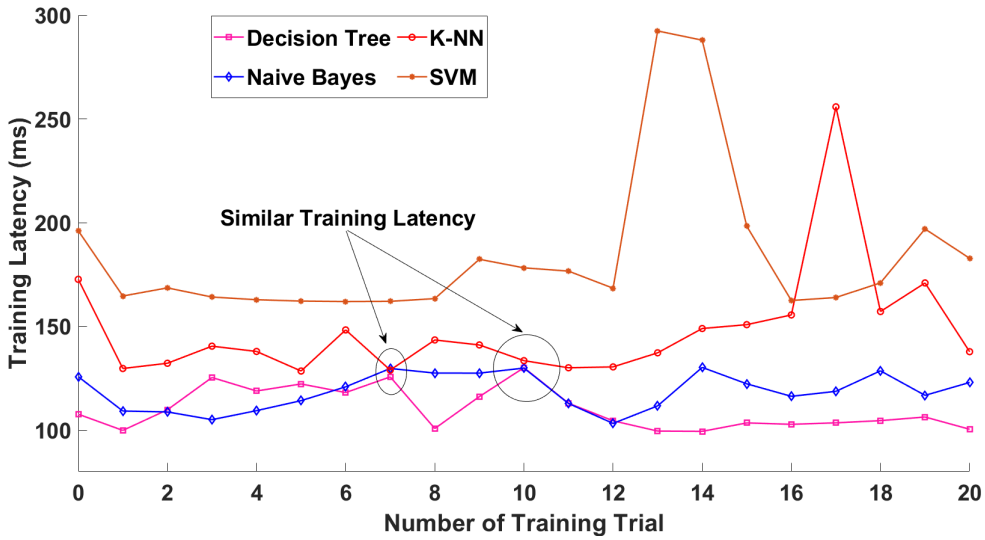

Figure 15. Comparison of NTCA training latency

Classification latency is how long it takes to classify new data. In Figure 16, we compare the NTCA classification accuracy, simulating over a one-hour simulation period.

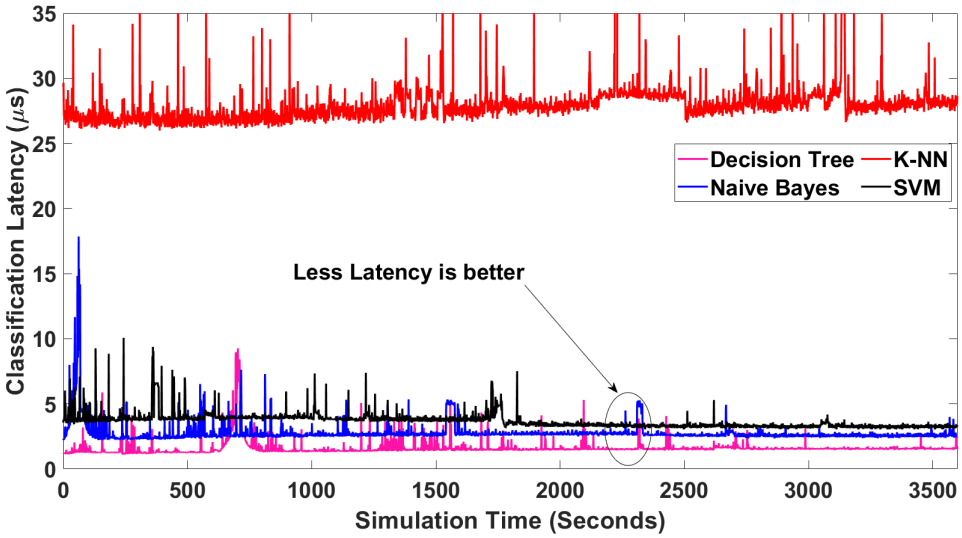

Figure 16. Comparison of NTCA classification latency.

The average latencies of K-NN-, SVM-, NB-, and DT-based NTCAs are 28.01 µs, 3.668 µs, 2.773 µs, and 1.593 µs, respectively. Relatively, SVM, NB, and DT NTCAs impose the least mean latency, which is less than 4 µs. Additionally, the DT-based NTCA has the least average latency, which is 1.593 µs.

5.6. Computing Resource Utilization of NTCA Using Different ML Models

5.6.1. Memory Usage

Figure 17 shows that the NB traffic classifier agent design uses the least average memory of 141.1 megabytes (MB) to train the model and classify new data instances. Despite the fact that the DT-based NTCA design offers the best classification accuracy of 99.504%, it uses the most memory resources amounting to 150 MB. This gives the system designer a chance to make a trade-off between the classification accuracy and memory usage while trying to choose a traffic classifier agent design.

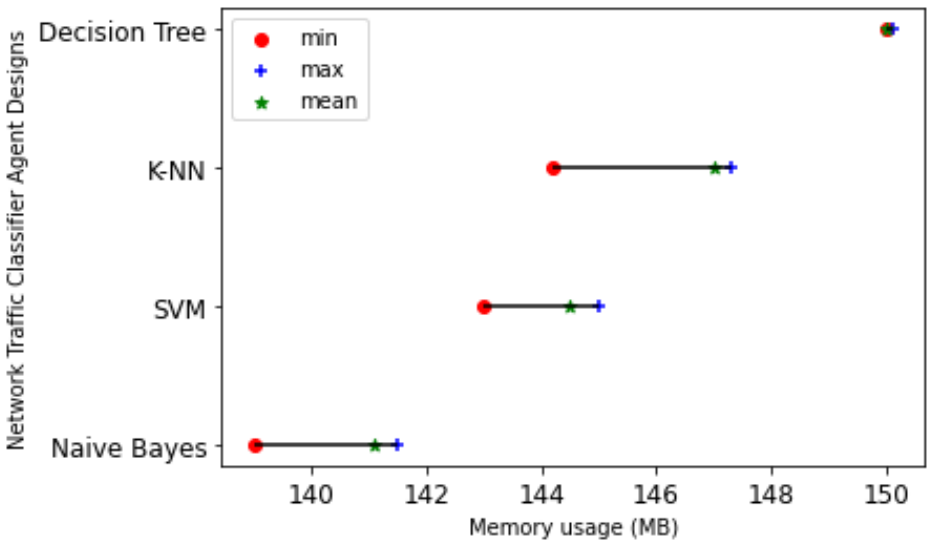

Figure 17. Comparison of NTCA memory usage.

5.6.2. CPU Usage

Figure 18 shows the statistical differences of each traffic classifier performance, including the minimum, mean, and maximum CPU usage. The SVM traffic classifier has the least maximum CPU utilization of 31.2%, while the DT-based NTCA has the largest maximum CPU utilization of 79.2%.

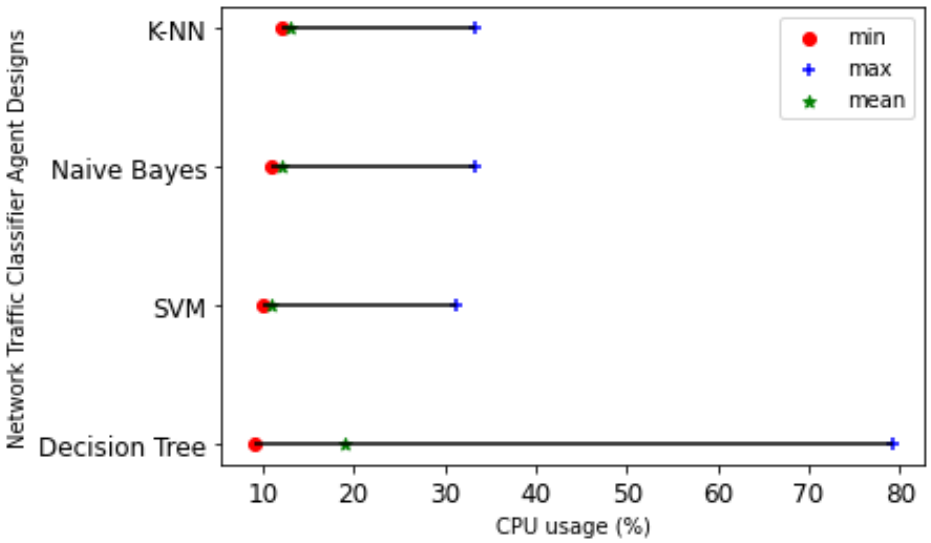

Figure 18. Comparison of NTCA CPU usage.

6. Conclusions and Future Work

Traffic classification and prediction are essential for network resource management, QoS provisioning and monitoring, security, and network failure detection. For example, proactive-based network management, resource allocation, and traffic scheduling can be performed using the prediction result. An organized introduction of intelligence in the network is required to have automated network control and management. Using multi-agent-based service-orientation, a MANA-NMS architecture was presented in our previous work.

In this work, we designed and implemented traffic classifier and predictor agents. Both NTCA and NTPA leverage ML/DL as an internal cognitive component of the agents to use in their decision process. The agents are autonomous in performing their respective decision. Using different cognition elements in the design and implementation, the performance of both agents is evaluated in terms of classification and prediction accuracy, classification and prediction latency, training latency, and computation resource utilization. The result suggests that LSTM-based NTPA and DT-based NTCA perform better than the rest in terms of prediction and classification accuracy and latency, respectively.

As a future work, a realistic testbed implementation is required. Moreover, a complete system using the autonomous and atomic agents as a building block is necessary. However, we implemented only NTPA and NTCA in this work. Therefore, it is necessary to have other services, such as routing, QoS monitoring, 5G core access and mobility management function (AMF), session management function (SMF), firewall, deep packet inspection, and other novel functions to have a complete MANA-NMS model for network systems, such as 5G wireless network, autonomic network management, etc. Thus, it is an open challenge to implement such functions as intelligent agents. Furthermore, the agent is expected to be deployed in a container or virtual environment. The experimental evaluation of such deployment is also another open challenge that we will be working on.

Author Contributions: Conceptualization, methodology, formal analysis, investigation, supervision, original draft preparation, writing—review and editing, S.T.A.; methodology, formal analysis, writing—review and editing, original draft preparation, Z.A. and M.E.; resources, supervision, project administration, writing—review and editing, and funding acquisition, M.D.; writing—review and editing, supervision and funding acquisition, F.G.; All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially funded by NATO Science for Peace and Security (SPS) Program in the framework of the project SPS G5428 "Dynamic Architecture based on UAVs Monitoring for Border Security and Safety", and partially by the US National Science Foundation under the New Mexico SMART Grid Center-EPSCoR cooperative agreement Grant OIA-1757207.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Samaan, N.; Karmouch, A. Towards autonomic network management: An analysis of current and future research directions. *IEEE Commun. Surv. Tutor.* **2009**, *11*, 22–36. [CrossRef]
2. Long, X.; Gong, X.; Que, X.; Wang, W.; Liu, B.; Jiang, S.; Kong, N. Autonomic networking: Architecture design and standardization. *IEEE Internet Comput.* **2017**, *21*, 48–53. [CrossRef]
3. Hexa-X. D5.1—A Flagship for B5G/6G Vision and Intelligent Fabric of Technology Enablers Connecting Human, Physical, and Digital Worlds. 2021. Available online: https://hexa-x.eu/wp-content/uploads/2022/01/Hexa-X_D5.1_full_version_v1.0.pdf (accessed on 22 June 2022).
4. D5.1—AI-dRiven Communication Computation Co-Design: Gap Analysis and Blueprint. 2021. Available online: https://hexa-x.eu/wp-content/uploads/2021/09/Hexa-X_D4.1_slideset.pdf (accessed on 22 June 2022).
5. ETSI, Network Functions Virtualisation (nfv); Terminology for Main Concepts in nfv., ETSI Industry Specification Group (ISG), Gs Nfv 003-V1.4.1, Volume 1. 2018. Available online: http://www.etsi.org/standards-search (accessed on 22 June 2022)
6. Movahedi, Z.; Ayari, M.; Langar, R.; Pujolle, G. A survey of autonomic network architectures and evaluation criteria. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 464–490. [CrossRef]
7. Behringer, M.; Dutta, A.; Hertoghs, Y.; Bjarnason, S. Autonomic networking-from theory to practice. In Proceedings of the IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–17.
8. Herrera, J.G.; Botero, J.F. Resource allocation in nfv: A comprehensive survey. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 518–532. [CrossRef]
9. Fossati, F.; Moretti, S.; Perny, P.; Secci, S. Multi-resource allocation for network slicing. *IEEE/ACM Trans. Netw.* **2020**, *28*, 1311–1324 [CrossRef]
10. Berrayana, W.; Youssef, H.; Pujolle, G. A generic cross-layer architecture for autonomic network management with network wide knowledge. In Proceedings of the 8th International Wireless Communications and Mobile Computing Conference (IWCMC), Limassol, Cyprus, 27–31 August 2012; pp. 82–87.
11. Tsagkaris, K.; Logothetis, M.; Foteinos, V.; Poulios, G.; Michaloliakos, M.; Demestichas, P. Customizable autonomic network management: Integrating autonomic network management and software-defined networking. *IEEE Veh. Technol. Mag.* **2015**, *10*, 61–68. [CrossRef]
12. Arzo, S.T.; Bassoli, R.; Granelli, F.; Fitzek, F.H.P. Multi-agent based autonomic network management architecture. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 3595–3618. [CrossRef]
13. Arzo, S.T.; Naiga, C.; Granelli, F.; Bassoli, R.; Devetsikiotis, M.; Fitzek, F.H.P. A theoretical discussion and survey of network automation for iot: Challenges and opportunity. *IEEE Internet Things J.* **2021**, *8*, 12 021–12 045. [CrossRef]
14. Arzo, S.T.; Scotece, D.; Bassoli, R.; Barattini, D.; Granelli, F.; Foschini, L.; Fitzek, F.H. Msn: A playground framework for design and evaluation of microservices-based sdn controller. *J. Netw. Syst. Manag.* **2021**, *30*, 1573–7705. [CrossRef]
15. Foundation, O.N. Open Network Operating System (onos). Available online: https://docs.onosproject.org/ (accessed on 22 June 2022).
16. Forestiero, A.; Papuzzo, G. Agents-Based Algorithm for a Distributed Information System in Internet of Things. *IEEE Internet Things J.* **2021**, *8*, 16548–16558. [CrossRef]
17. Serugunda, C.N.; Arzo, S.T.; Granelli, F.; Bassoli, R.; Devetsikiotis, M.; Fitzek, F.H. Autonomous network traffic classifier agent for autonomic network management system. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–6.
18. Dowell, M.L.; Bonnell, R.D. Learning for distributed artificial intelligence systems. In Proceedings of the The Twenty-Third Southeastern Symposium on System Theory, Columbia, SC, USA, 10–12 March 1991; pp. 218–221.

19. Shaw, M.J.; Harrow, B.; Herman, S. Distributed artificial intelligence for multiagent problem solving and group learning. In Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences, Kauai, HI, USA, 8–11 January 1991; Volume iv, pp. 13–26.
20. Dorri, A.; Kanhere, S.S.; Jurdak, R. Multi-agent systems: A survey. *IEEE Access* **2018**, *6*, 28573–28593. [CrossRef]
21. Rizk, Y.; Awad, M.; Tunstel, E.W. Decision making in multi agent systems: A survey. *IEEE Trans. Cogn. Dev. Syst.* **2018**, *10*, 514–529 [CrossRef]
22. Hexa-X. D1.2—Expanded 6G Vision, Use Cases and Societal Values—Including Aspects of Sustainability, Security and Spectrum. 2021. Available online: https://hexa-x.eu/wp-content/uploads/2021/05/Hexa-X_D1.2.pdf (accessed on 22 June 2022).
23. Kim, H.; Lee, D.; Jeong, S.; Choi, H.; Yoo, J.; Hong, J.W. Machine learning-based method for prediction of virtual network function resource demands. In Proceedings of the IEEE Conference on Network Softwarization (NetSoft), Paris, France, 24–28 June 2019; pp. 405–413.
24. Yang, S.; Yu, X.; Zhou, Y. Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example. In Proceedings of the International Workshop on Electronic Communication and Artificial Intelligence (IWECAI), Shanghai, China, 12–14 June 2020; pp. 98–101.
25. Ramakrishnan, N.; Soni, T. Network traffic prediction using recurrent neural networks. In Proceedings of the IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 17–20 December 2018; pp. 187–193.
26. Wang, W.; Bai, Y.; Yu, C.; Gu, Y.; Feng, P.; Wang, X.; Wang, R. A network traffic flow prediction with deep learning approach for large-scale metropolitan area network. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–9.
27. Neagoe, V.-E.; Ciotec, A.-D.; Cucu, G.-S. Deep convolutional neural networks versus multilayer perceptron for financial prediction. In Proceedings of the International Conference on Communications (COMM), Bucharest, Romania, 14–16 June 2018; pp. 201–206.
28. Navada, A.; Ansari, A.N.; Patil, S.; Sonkamble, B.A. Overview of Use of Decision Tree Algorithms in Machine Learning. In Proceedings of the 2011 IEEE Control and System Graduate Research Colloquium, Shah Alam, Malaysia, 27–28 June 2011; pp. 37–42. [CrossRef]
29. Ray, S. A Quick Review of Machine Learning Algorithms. *Int. Conf. Mach. Learn.* **2019**, 35–39. [CrossRef]
30. Huang, J.; Lu, J.; Ling, C.X. Comparing naive Bayes, decision trees, and SVM with AUC and accuracy. In Proceedings of the Third IEEE International Conference on Data Mining, Melbourne, FL, USA, 22–22 November 2003; pp. 553–556. [CrossRef]
31. Support Vector Machine-Mitosis Technologies. 2021. Available online: https://www.mitosistech.com/support-vector-machine/ (accessed on 5 January 2021).
32. Soon, G.K.; On, C.K.; Anthony, P.; Hamdan, A.R. A review on agent communication language. In *Computational Science and Technology*; Alfred, R., Lim, Y., Ibrahim, A.A.A., Anthony, P., Eds.; Springer: Singapore, 2019; pp. 481–491.
33. O. S. de Informacíon Internet S.L. Revision d0b241de., "osbrain-0.6.5". Available online: https://osbrain.readthedocs.io/en/stable/ (accessed on 23 June 2022).
34. ZeroMQ. An Open-Source Universal Messaging Library. Available online: https://zeromq.org (accessed on 22 June 2022).