Security Threats Against Communication and Computing Task Classification for Edge Computing

Younes Salmi
Institute of Radiocommunications
Poznan University of Technology
Poznan, Poland
email: younes.salmi@student.put.poznan.pl

Hanna Bogucka
Institute of Radiocommunications
Poznan University of Technology
RIMEDO Labs
Poznan, Poland
ORCID: 0000-0002-1709-4862

Abstract—**This paper addresses machine learning-based task allocation in edge-computing networks with diverse end-user devices (IoT devices) and diverse edge-computing servers. The task allocation problem is solved by generated-requests classification that can be mapped to servers with enough computational power and proximity to devices that translates to service latency and reliability. Moreover, the impact of cyberattacks on the classification algorithm is also studied.**
This paper was originally presented at the NATO Science and Technology Organization Symposium (ICMCIS) organized by the Information Systems Technology (IST) Panel, IST-200RSY – the ICMCIS, held in Skopje, North Macedonia, 16-17 May 2023

Keywords—edge computing, classification, machine learning, cyberattacks

I. Introduction

According to the recent Ericsson Mobility Report [1], there will be 5 billion of 5G mobile subscriptions by 2028. Moreover, wireless communication of 34,7 billion machines and devices is expected by that year to comprise the Internet of Things (IoT). This predicted massive communication of humans and machines stimulates the research and development of new generations of mobile communication systems. While 5G standards are still on the way (3GPP Release 18 to be frozen in 2024), 6G networking has already been proposed [2]-[5]. One of the key paradigms of 6G is edge computing and edge intelligence.

In the prospective 5G/6G architecture, a mobile edge host runs a mobile edge platform that facilitates the execution of applications and services at the edge. From the data analytics perspective, edge intelligence refers to data analysis and the development of solutions at or near the site where the data is generated and further utilized. Edge intelligence thus allows the reduction of latency, costs, and security risks, making the associated business more efficient. From the network perspective, edge intelligence mainly refers to intelligent services and functions deployed at the network's edge [6], [7]. The focus of the research is on self-learning networks and systems that can autonomously manage resources and control functions. Here, by resources, communication (radio spectrum) and computing (computational power) resources are meant, as well as the relevant energy both in communication and computing. It is impractical to transmit a massive amount of local data to the centralized cloud for training and inference. This calls for new architectures and associated communication-efficient training algorithms over wireless links while making real-time and reliable inferences at the network edge. Such architectures also pose new challenges: limited access to training data, low inference accuracy, lack of generalization, and limitations of processing power and memory for edge devices [8].

In this paper, we consider edge computing network architecture with diverse servers (with diverse computing and storage capabilities) and diverse IoT devices (generating diverse communication and computing requests). This architecture is presented in Fig. 1.

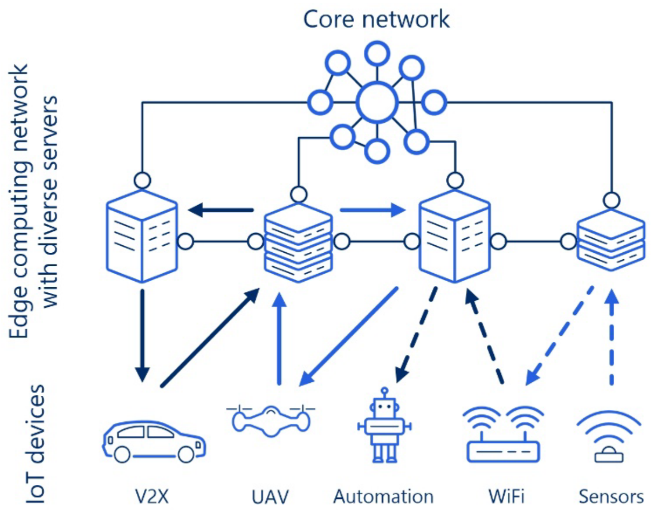

Fig. 1. Edge computing network architecture. Arrows represent diverse communication and computing tasks flow.

The optimal communication and computing (2C) task allocation to the appropriate servers is a challenging problem. It has been addressed in [9][10]. However, such optimization is not always possible given the limited knowledge of the network components at the edge. Therefore, here, we consider using Machine Learning (ML) to classify generated 2C requests. This classification is intended to support the near-optimal delegation of a request to an appropriate server at the edge of a network.

The architecture of the 5G/6G network will be almost entirely virtualized and based on software functionalities. Consequently, it is vulnerable to being used, attacked, and disrupted by hackers. Attacks on ML algorithms used for traffic steering may lead to fatal errors in the case of sensitive applications, such as mission-critical ones that require ultra-low latency and ultra-high reliability (URLLC) [11]-[13]. In our considered edge-computing network, 2C tasks should be classified based on service requirements (for example, the end-to-end latency, the packet error rate PER, the computational complexity of a delegated task…etc.) The classification criteria may change, but considering, for example, the multi-class classification with defined sensitivity levels (starting from noncritical tasks to highly critical ones), any miss-classification may cause a fatal classification error, especially when the classes are close to the classifier.

Here below, we investigate how these attacks impact the classification of the 2C tasks and their offloading to edge computing servers. In Section II, we describe selected ML-based tasks classification methods and in Section III we consider security attacks on these algorithms. In Section IV we define critical use-cases selected for security evaluation. Section V presents simulation results. In Section VI we conclude our work and discuss further research directions.

II. ML-BASED CLASSIFICATION OF COMMUNICATION AND COMPUTING TASKS

Supervised or unsupervised ML algorithms can handle the binary classification issue, for example, to describe the spectrum sensing process, which requires the classifier to distinguish between the availability and unavailability of the two potential states of the radio frequency channel. Energy vectors or probability vectors, also known as feature vectors in the context of ML, are used by the classifier to determine whether the spectrum is available [14]. The classification of the 6G services traffics might be classified using multi-classifications algorithms from both supervised and unsupervised learning, such as the *softmax* algorithm at the output layer of a neural network (NN) as a classifier. Transfer learning might also be helpful, borrowing some already existing NN algorithms from 5G [15], and then either retraining the output layer or retraining the entire layers using the existing model's parameters as initial ones for the desired model. Other standard classification algorithms that can be applied are the following: Naive Bayes Classifier, Logistic Regression, Decision Tree, Random Forests, Support Vector Machines, *k*-Nearest Neighbors, and *k*-Means Clustering [12].

In our considered scenario, 2C task categorization is used to allocate end-user requests to edge-computing servers. Data representing 2C tasks have features reflecting the required Quality of Service (QoS) of communication and computing. For 2C services, these are end-to-end latency (including propagation delay, queuing delay, and time of computations), reliability (or reversely bit- or packet error rate), data rate (including transmission bit-rate and computing speed in terms of floating-point operations per second), and the energy consumption (including energy spent on wireless and wired communication and computations). Two algorithms will be discussed and evaluated in the remainder of this paper: one is the unsupervised machine learning algorithm, namely *k*-means clustering [16], and the other is the supervised machine learning algorithm called *k* nearest neighbors (*k*-NN) [17].

A. k-means algorithm for task allocation

In *k*-means clustering, the examples are divided into *k* clusters, with each example belonging to the cluster with the closest mean (cluster centers or cluster centroids), which serves as the cluster's prototype. The variance within a cluster is minimized. The most common algorithm uses an iterative refinement technique as follows: *k* clusters are created by associating every observation with the nearest mean, and then the centroid of each of the *k* clusters becomes the new mean.

The advantages of the *k*-means classification algorithm include applicability to large datasets and guaranteed convergence for locating clusters. Disadvantages include providing the initial value for *k* and sensitivity to outliers. *k*-means clustering performance is usually not as competitive as the other sophisticated clustering techniques because slight variations in the data could lead to high variance. Furthermore, clusters are assumed to be spherical and evenly sized, which may reduce the accuracy of this algorithm for data representing generated 2C tasks. Therefore this kind of data requires some preprocessing.

B. k-NN algorithm for task allocation and prediction

k-NN-based classification is a supervised algorithm, which means that in the training phase, it requires full knowledge of the output corresponding to the training input data. In other words, the examples (training data) are labeled. Based on the training data set, new input data can be classified into one of the output categories by calculating distances (usually Euclidean distance in multidimensional space) to *k* closest neighbors in used features space. For example, if $k = 1$ only one closest data point is considered, and its category is assigned as the label of that point. In the case of $k > 1$, the most numerous group of neighboring points of one category determines the result. Thus, the prediction of a new example is based on the voting of the *k* nearest neighbors, and the predicted label is assigned to the majority vote.

The advantage of this algorithm is that it can be applied to datasets of any distribution. Disadvantages that are usually noted are that: it is easily affected by outliers, it is biased towards a class that has more instances in the dataset, and it is challenging to find the optimal number for *k*. In our scenario, the disadvantage of *k*-NN being affected by outliers can be considered an advantage for security algorithms since it can detect outliers.

Finally, note that the unsupervised learning classification is not enough to handle the newly generated tasks and forward them to the appropriate server at the edge of the network, hence, algorithms are needed to predict the new tasks based on the results obtained using unsupervised classification. One example scenario is to use the classes obtained by *k*-means classification to feed the supervised *k*-NN, so the *k*-NN voting is based on the distance between the centers of the classes from k-means and the newly generated task.

III. CLASSIFICATION SECURITY ISSUES

ML-based systems suffer from a subset of vulnerabilities caused by the inherent limitations of the learning algorithms. Moreover, ML deployment for wireless communication makes it more subjected to attacks due to the open nature of the transmission medium in the case of wireless channels. An adversary can use conventional wireless attacks and intelligent ML-based techniques known in adversarial machine learning (AML). Recent studies have shown that these types of attacks are possible [18][19][20].

Most machine learning approaches are designed to operate on a certain problem known under the field of artificial narrow intelligence (ANI) or weak intelligence, which is a data-set-dependent field, and small changes in the operating environment or operating data affect the performance of ML algorithms [21]. Moreover, studies of AML, although in the advanced stage for computer vision and natural language processing, are still in the early stage for wireless communications [18]. Thus, many ML techniques applied in a wireless environment are borrowed from other domains, which may lead the adversary to learn more about the models and techniques used.

As the Radio Access Network (RAN) provides access to and coordinates the management of resources across the radio environment, an adversary may attack the system differently and target several goals. For example, an adversary may be a source of a jamming signal, he or she may violate the legitimate user's privacy to exploit the user equipment (UE), exhaust the systems by impacting the energy consumption, impede a multi-access algorithm, intercept the information using adversarial demodulation and decoding, use the legitimate user identity as spoofing, poison data and ML models in a system, etc...

All these adversary actions may have diverse results on the system's performance, including the classification of 2C tasks, which is the focus of this paper. The data representing requests generated by the users (and their features) under attack may cause false classification and as a consequence, wrong allocation of 2C tasks to servers, overload of some servers, and inability to serve the users. In what follows, we examine the impact of massive data poisoning on the classification of 2C tasks.

IV. USE CASES

Out of multiple scenarios of requested 2C tasks, we have selected some representative use cases, which are described below. Most of all, they are defined by the datasets and required preprocessing of data for two considered classification algorithms: *k*-means and *k*-NN. In our use cases, the classification using *k*-NN (supervised) is performed with labels obtained from the *k*-means algorithm (unsupervised). Moreover, for the datasets, we consider massive data poisoning. It means that all data representing 2C requests are altered by jamming in a wireless environment.

A. *Data sets*

The dataset consists of a number of examples (representing 2C requests) with three features: the end-to-end latency in [ms], the PER (packet error rate), and the computational complexity in FLOPs (floating operation points per second). The first two features can be obtained according to the QoS of 5G defined by ETSI in "(2022-05)1683GPP TS 23.501 version 17.4.0" Release 17". The following services are selected to build the dataset:

- IP-Multi-Service (IMS) signaling with 100 ms latency and 10^{-6} PER.
- Conversational voice with 100ms latency and 10^{-2} PER.
- Low latency eMBB applications (Augmented reality) with 10ms latency and 10^{-6} PER.
- Discrete automation with 10ms latency and 10^{-4} PER.
- Intelligent transportation system with 30ms latency and 10^{-5} PER.
- V2X (Collision avoidance) with 5ms latency and 10^{-4} PER.
- Electricity distribution-high voltage with 5ms latency and 10^{-4} PER.
- Non-conversational video (buffered streaming) with 300 ms latency and 10^{-6} PER.
- Interactive gaming with 100ms latency and 10^{-3} PER.
- Mission critical data with 200ms latency and 10^{-6} PER.

While the computational complexity is generated around 1 kFLOPs (kilo-FLOPs = 10^3 FLOPss), 1 MFLOPS (Mega-FLOPs = 10^6 FLOPs), 1 GFLOPs (Giga-FLOPs = 10^9 FLOPs), 1 TFLOPs (Tera-FLOPs= 10^{12} FLOPs), 1 PFLOPs (Peta-FLOPs = 10^{15} FLOPs). These quantities may differ from one task to another, for example, image and video processing requires higher FLOPs than signaling text messages, while ML training processing may achieve 10^{19} TFLOPs per epoch (10 TFLOPs SP for 1 epoch per hour). The values are considered for FP32, and the servers considered CPU/GPU process the tasks.

The different services' signals are generated using random generators with Gaussian distributions with means equal to the service features values and variances equal to 10%, and 20% of the means. The reasoning behind the choice of Gaussian distributions is to create some of the data points which are near (because they are generated by the same type of IoT devices), and some which are far from each other (because they are generated abnormally by the same type of devices or by other types of devices). The purpose of such a choice is to test the prediction scenario between two far examples and two near examples.

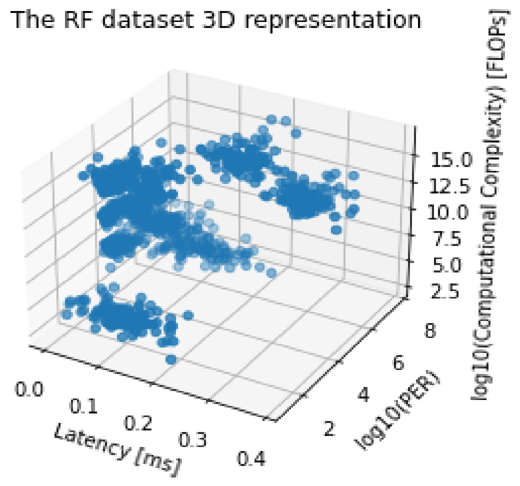

Fig. 2. The visualization of the generated and post-processed dataset.

The dataset is presented in a linear scale when the axes with PER and computational complexity are logarithmically scaled to visualize the examples, in addition to the negative sign introduction for the PER. The dataset consists of 1000 examples, more examples might be used depending on the computation budget of the device running the ML algorithms used later. Other services might also be considered.

B. *Data preprocessing*

And as classification algorithms are mathematically derived, one must convert all input and output data to numerical values. The first step of a classification algorithm is to ensure that the variables, whether input or output, have been encoded correctly. Secondly, because both *k*-means and *k*-NN require uniformly distributed data on all the axes, data preprocessing is necessary to run the algorithms correctly.

For the dataset features: computational complexity and packet error rate, the instances of these features are mapped using logarithm base 10 transformation. Moreover, we can observe that both features' intervals' length after mapping is more or less 10, for this purpose the latency feature instances are also mapped to an interval length of 10 units using linear interpolation function (1):

$$f(x) = c + \left(\frac{d-c}{b-a}\right)(x-a), \tag{1}$$

where $f(x)$ is the post-processed example, function of x which is pre-processed (mapped) example, a and b are the lower and upper x-interval's borders respectively, while c and d are the lower and upper post-processed-interval's borders respectively. For our case, only the latency is mapped using interpolation function, and a is the minimum latency value equal to 5 ms and b is the maximum latency value 300 ms, c equals 1 and d equals 10. As a result, the new latency interval is [0.0, 0.4] ms. The two other axes are mapped logarithmically. Quantile-Quantile (Q-Q) plots for each feature were used to check the similarity of the post-processed features to uniform distributions with means and variances obtained using statistical tools from the *Stats* library on the post-processed features. The obtained results are good enough (approximately straight line). Also Silhouette Score test was performed to check whether the post-processed dataset is optimal, the Silhouette Score results are above 0 for the different k values [2, 12]. The intervals of the post-processed features will be used later for all the plots.

C. *Attacks on the task classification algorithms*

In our scenario, we consider massive data poisoning. It means that all data representing 2C requests are altered by an additive Gaussian-noise-like jamming in a wireless environment. An attacker injects a jamming signal into data sets originating from various users and their locations. Jamming events follow the Poisson distribution. We also assume that the attack has one-hundredth time unit precision (if, for example, the legitimate data signal is sent at time nT, the jammer affects the legitimate data if the jamming event occurs at a moment between $nT - T/100$ and $nT + T/100$, where T is the data interval). An increase of the number of transmitting sources (UEs) increases the attack efficiency by achieving a higher probability of hitting legitimate signals.

The attack is sensing-independent, i.e., the jammer sends the illegitimate signals independently on the legitimate signals. Both legitimate and illegitimate sources generate signals at events' time instances which follow Poisson distribution described by the following probability density function (PDF):

$$f(k,\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{2}$$

where: λ is the mean number of successes that occur during a specific interval and equals 0.5, k is the number of successes, and e is a constant equal to approximately 2.71828. To generate this distribution a new function is defined and tested using Chi-Squared test instead of predefined Poisson distributions from Numpy or other libraries. Both event lists consist of 1000 instances. Moreover, the jammer has 10^{-2} hitting-target time precision, which means that an example is affected if $|T_{\text{jamming-event}} - T_{\text{legitimate-event}}| >= 0.01$ time unit.

V. SIMULATION RESULTS

A. *k-means classification results*

As mentioned above, the k-means algorithm is unsupervised and classifies the dataset examples into some classes which are centered around centroids that are not predefined. The algorithm runs iteratively till convergence. For this purpose, a cost function is defined together with some threshold to stop. For our use cases, the cost function is a simple Mean Square Error (MSE) function, and the stop condition is met when MSE < ε, where $\varepsilon = 10^{-15}$. The MSE here is the sum over all the classes' centroids of the difference between the Euclidean distances of the current-run n-th centroid of the i-class and the preceding (n-1)-th centroid $\sum_{i=1}^{L} |C_n^i - C_{n-1}^i|$, where L is the number of the classes.

The dataset is loaded to the algorithm with a different number of centroids k {3, 6, 9, 12}. The results of this classification are presented in Fig. 3 and 4. The histograms represent the distribution of the examples over the different classes. For different numbers of classes we have different classifications, the classes are not equally dense, after classification some tasks are more probable to belong to a particular class due to the facts that i) the weighing coefficients of axes chosen to preprocess the data, and ii) the dataset is not uniformly distributed over the 3D plane.

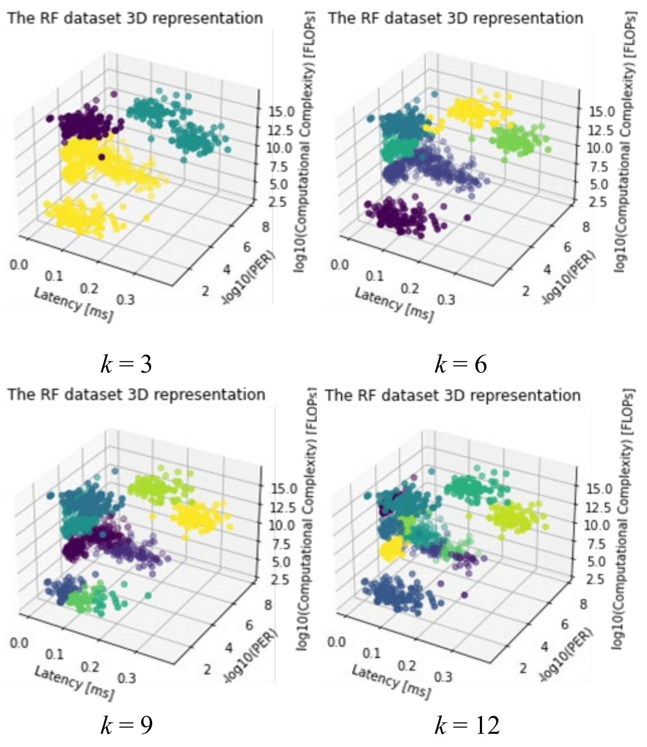

Fig. 3. k-means classification of tasks for different centroids' number.

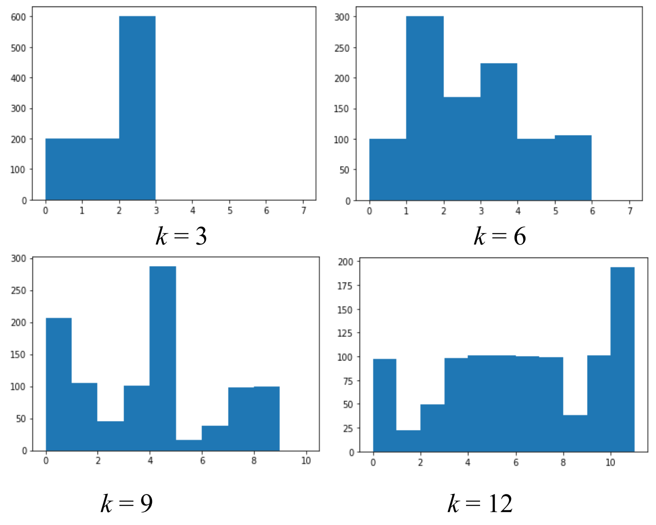

Fig. 4. Histograms of the examples (tasks) assigned to centroids.

The results show that the *k*-means classification is an efficient, accurate, and flexible unsupervised learning algorithm. However, the drawback is that the centroids' number and value define the classification criteria, i.e., for data closely spaced in the 3D plane, different centroids' initialization leads to different classifications. To obtain a desired classification, one should carefully choose centroids numbers and initial positioning based on data distribution (this is more challenging for datasets with higher dimensionality 4D and above).

In Fig. 5 and 6, classification results can be observed when 6 centroids are chosen, with different random initializations. Indeed, the different initializations lead to different classifications. Correctly pre-defined centroids result in better (more accurate) classification, help the algorithm to converge fast. Another factor that may change the classification is the mapping weights, where an axis with a wider unit forces the algorithm to give the priority of classification to that axis. The final centroids are used for the prediction of new examples using for example MMSE between the new examples and the centroids. The final centroids may have different features' values but not too far from the initial centroids.

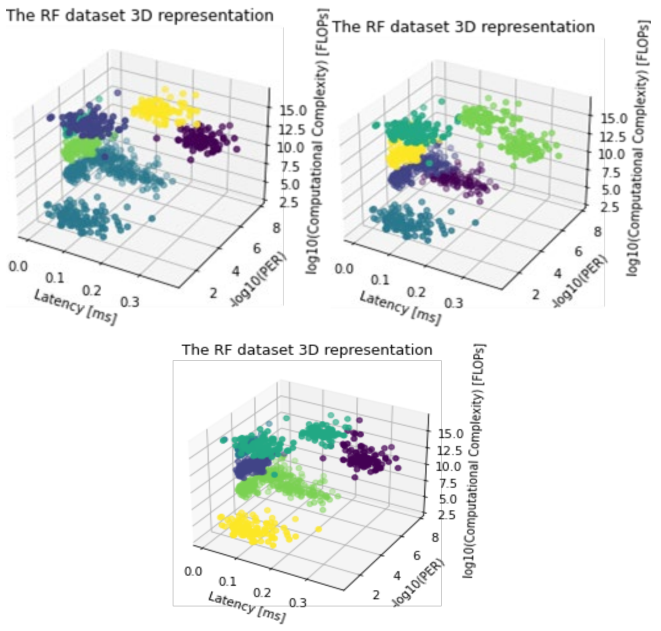

Fig. 5. *k*-means classification of tasks for different 6 centroids' initialization; (a) first run, (b) second run, (c) third run.

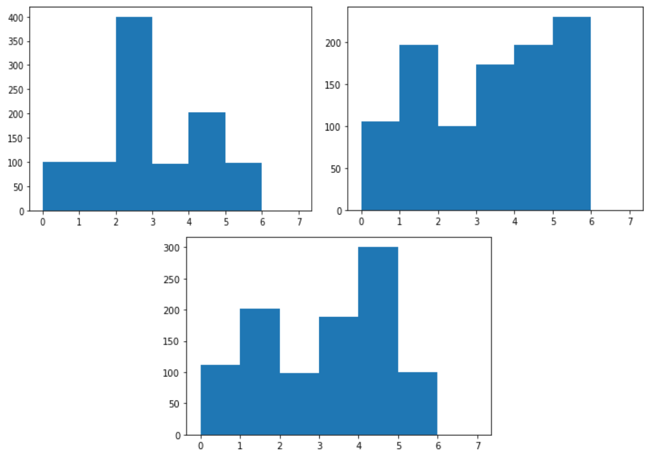

Fig. 6. Histograms of the examples assigned to centroids for the different runs.

Based on Silhouette Score test, the *k*-means algorithm has been examined for the number of centroids with the highest Silhouette Score result (k=9, 8). First to present the algorithm, nine centroids have been selected as follows (in the form of (latency, PER, computation complexity) vector): centroid 1: $(0.05,10^{-6},10^{5})$, centroid 2: $(0.1, 10^{-2},10^{5})$, centroid 3: $(0.25,10^{-5},10^{5})$, centroid 4: $(0.05,10^{-6},10^{9})$, centroid 5: $(0.01,10^{-4},10^{9})$, centroid 6: $(0.1,10^{-2},10^{13})$, centroid 7: $(0.25,10\text{-}5,10^{13})$, centroid 8: $(0.01, 10^{-4},10^{13})$, centroid 9: $(0.1, 10^{-2},10^{15})$. The classification results after 100 iterations are presented in Fig. 7. One can see well-separated clusters and a more evenly distributed histogram.

There is no direct test for *k*-means separately because the algorithm involves a stochastic component, it is exceedingly improbable that we would receive the same result in a test unless we utilize the identical implementation and starting setup. However, we may check whether our results coincide with well-known implementations. To do so, when comparing two series of data, label swapping remains an issue if it is repeated several times. A solution is to use freely accessible data to benchmark our implementation, or we may simulate a specific data set (e.g., using a finite mixture model, as in the MixSim package for "Simulating Data to Study Performance of Clustering Algorithms" in the CRAN repository). Results show 89% average accuracy for our implementation.

B. *k nearest neighbors (k-NN) prediction*

The classification using *k*-NN is performed with labels obtained from the *k*-means algorithm. The prediction of new examples depends on the weights of the features. Let us assume the same weights as before, for example, a new example $(0.4, 10^{-2}, 10^{9})$ we know that the example is most likely assigned to the 5th centroid (obtained from *k*-means) (2.00437371e-02 3.44864990e-05 5.75269952e+09). Running the *k*-NN algorithm to predict this example's class had the result presented in Fig. 8.

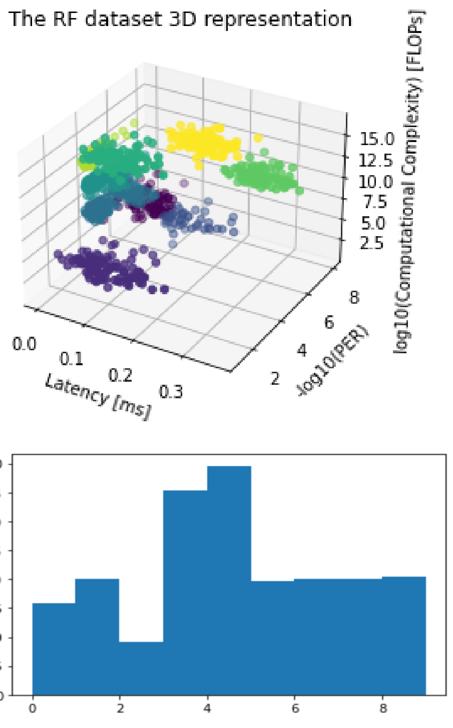

Fig. 7. (a) 3D visualization of the run with 9 pre-defined centroids, (b) The histogram of the examples assigned to the final centroids after running the *k*-means for 100 iterations with pre-defined centroids.

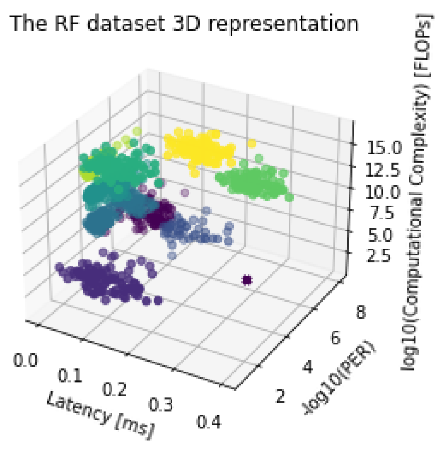

Fig. 8. One example prediction with k-NN

From the plot in Fig. 8, we see that the class of the example is the class with the purple color, which has been obtained with the majority voting of 50 neighbors. Moreover, the example was chosen on purpose, i.e., to be far from the clusters, other scenarios for closer locations will be addressed in the security part. Note that other prediction algorithms might be used, such as the l-2 norm. Our purpose, however, is not to test and compare the ML algorithms for task allocation prediction, but to analyze the impact of security threats on the algorithms selected for our use cases.

The test dataset with 500 examples was created using the same parameters for distributions discussed earlier to test the k-NN separately. The obtained accuracy was 92%.

To check the k-means/k-NN overall algorithm optimality a hybrid Round-Robin item allocation and Round-Robin CPU/GPU scheduling were used as benchmark algorithm. In this algorithm first, tasks are allocated to the task-processing servers based on Round-Robin allocation and on the arrival time. After that, the tasks assigned to each server are scheduled using Round-Robin CPU/GPU scheduling. The servers buffers were assumed unlimited, i.e. no outage of the computational tasks were considered (if the server processing-unit is busy, an arrived task was rescheduled and placed in the server buffer). The results show that the ML based algorithm outperforms the conventional Round-Robin, and the benefits are significantly higher for higher task complexities. (see Fig. 9.)

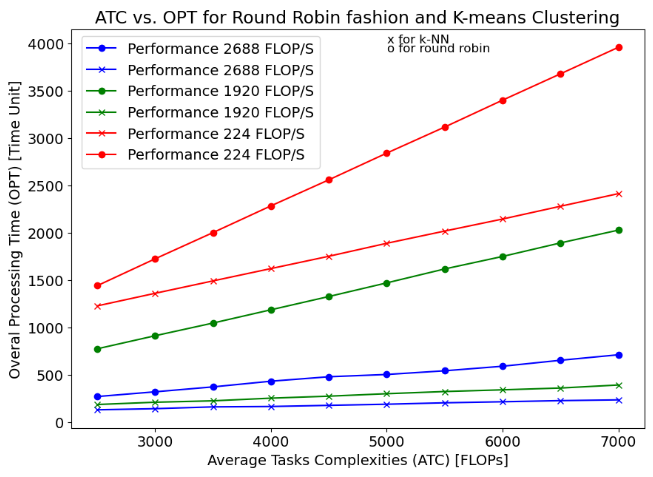

Fig. 9. The overall processing time vs. task complexities and processing performance for *k*-NN and Round-Robin task allocation.

C. *Random jamming results – Attack 1*

The effects of attacks have been tested in the training and prediction phases of task classification. For the training phase, MSE has been used as a measure of the difference between the affected dataset and the original one. For the prediction phase, new examples have been generated deliberately to fall between two close clusters. This approach allowed testing the prediction quality in face of an attack in the most demanding scenario.

The effects of the attack are considered regarding the power budget of an attacker reflected in the generated jamming signal power and in the inter-arrival time between the attack events. In the classification phase, results have been obtained for a reference (legitimate) dataset and a single run of the *k*-means classification algorithm for 8 deliberately initialized centroids. The results in terms of MSE between the classified data in case of the absence of the attacker and in case of the attack are presented in Fig. 10. The MSE is calculated as the average of 1000 runs for different (legitimate) signal power to (jamming) noise power ratio (SNR). Moreover, MSE was calculated after data preprocessing.

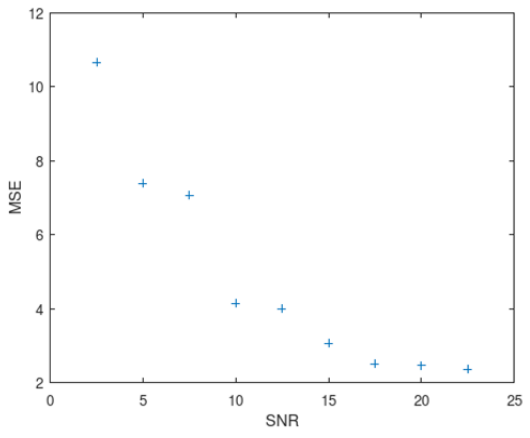

Fig. 10. MSE versus SNR for 8 centroids in *k*-means algorithm.

As expected, the higher SNR, the lower the considered MSE. Moreover, we can observe the difference between the final centroids got by the *k*-means before and after the attack. Moreover, in these simulation experiments, we observed that the attacker hit the legitimate data 60 out of 1000 times. Thus, the dataset was affected by only 6%.

The task classification results in the same scenario without the attack and under attack can be observed in Fig. 11 and 12. Looking at the regions between the classes in the 3D plots, we can observe the change in the allocation of task (only 60 examples are affected), but using the histograms it is not so much obvious because the attack effect may interchange the tasks between the classes.

The following tests evaluate the attack effects on the prediction using *k*-NN. In these tests, the examples are labeled based on the previous centroids obtained by the *k*-means for the same SNR values. The *k*-NN with 50 neighbors used to predict the classes of the following example under no attack. The simple test was made for the latency feature as follows. First, the *k*-NN algorithm was tested with 50 neighbors to predict the classes of the new example [0.4, 1^{-6}, 10^{14}] under no attack. The voting result was 26/50 for Class 5. Next, the voting of the same example under attack was tested by the same method. The results of this *k*-NN prediction are in Table I.

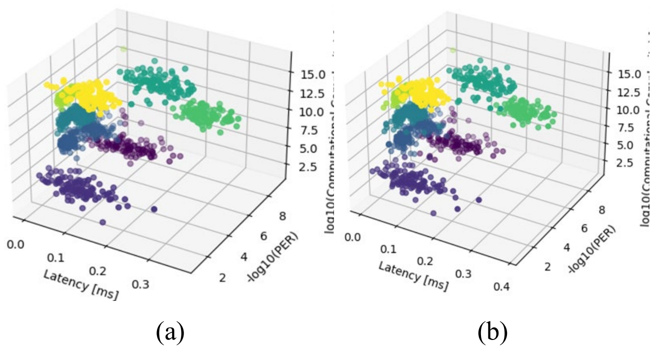

Fig. 11. 3D visualization of the run with 8 pre-defined centroids, (a) no attack scenario (b) under attack.

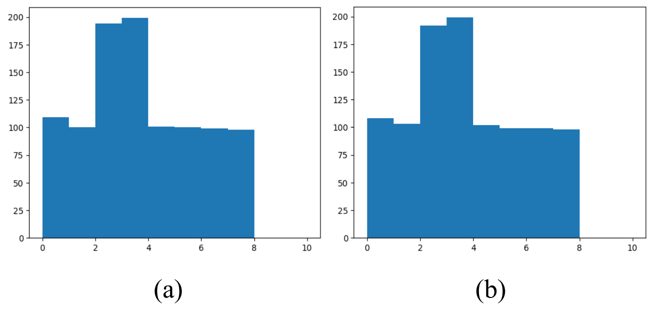

Fig. 12. The histograms of the examples assigned to the final centroids after running the *k*- means, (a) no attack scenario (b) under attack.

Table I. Voting results of 50 neighbors of [0.4,1^{-6},10^{14}] under attack

SNR value	Voting of 50 neighbors
2.5	Class 7
5	Class 7
7.5	Class 7
10	Class 4
12.5	Class 4
15	Class 4
17.5	Class 4
20	Class 4
22.5	Class 4

Results presented in Table I show that even high SNR does not mitigate the attack, where the best vote (the closest to the correct original selection of Class 5) was Class 4.

D. Jamming with 50% of the probability of targetting – Attack 2

Now, let us assume that the attacker senses the medium and the probability of success to target the example is 50% (500 affected examples). Fig. 13 shows the original and affected histograms. The results show that the clustering is significantly fooled, where 8 groups are present, also the Silhouette Score test results in below 0 score for some *k* values including our example (*k* = 9). Moreover, our tests show that the MSE between the original results of random hitting (about 6% of successful hitting) and the results under this attack (50% of successful hitting) increases for all features.

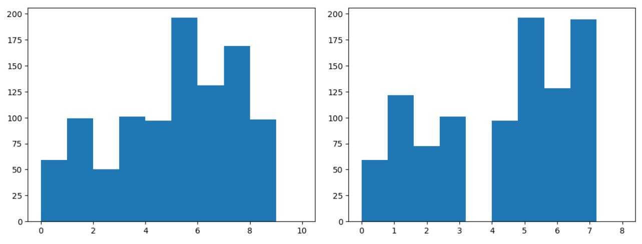

Fig. 13. The histograms of the examples assigned to the final centroids after running the *k*- means, (a) no attack scenario (b) under attack when the probability of success to target the example is 50%.

Now, let us consider the case of the lower number of neighbors participating in voting. The results under both scenarios (6% and 50% affection) for 20 (near and far) examples are listed in Table II below. Attack 1 is the 6% affection attack and Attack 2 is the 50% affection attack. The results are represented in terms of the accuracy of the model voting for the affected dataset compared to the original dataset. Using lower voting neighbors' number or concentrated dataset leads to fatal miss-classification/clustering for far examples from the centroids, but has no impact on the near examples (Around 100%), it may impact the near examples for the higher number of clusters.

Table II. Voting accuracy results of different neighbors under both attacks for the chosen examples for SNR 22.5 dB compared to voting under no attack.

Neighbors *k*	Attack 1	Attack 2
50	63%	42%
40	59%	39%
30	50%	33%
20	39%	24%
15	15%	5%
10	8%	0
5	3%	0

VI. Conclusions

As discussed above in this paper, the ML algorithms are very efficient in terms of their application to task classification and their allocation to executing servers in edge-computing networks. However, the classification algorithms and their results are prone to cyber-attacks. In this work, we have examined the impact of the attack on the *k*-means and *k*-NN classification and prediction. MSE has been used to analyze the attack results for the *k*-means algorithm, and the prediction voting was used to observe the attack effects on the *k*-NN method.

Several issues might be addressed for future work. First, other datasets shall be examined, possibly less dense but more uniformly distributed, by choosing services near to each other in terms of all the features. Data preprocessing should follow new criteria, where the feature coefficients play an important role in the results. Moreover, the selected poisoning attack scenario might be considered where there exist a few malicious users that deliver falsified data and impact the classification algorithm. Other attack types could also be considered. CPU/GPU performance under attack might be studied, where the attacker may use a sophisticated approach to down-perform or/and control the processing unit.

A very important aspect of the wireless environment and edge-computing scenario is the dynamically changing situation changes, e.g. one server may be particularly busy so that its capabilities drop in time. This means that the number of classes decreases and the classification algorithms should adjust to new classes. Research for a dynamically changing number of clusters is planned for the future.

ACKNOWLEDGMENT

This work was funded by the National Centre for Research and Development in Poland within the 5GStar project CYBERSECIDENT/487845/IV/ NCBR/2021 on “Advanced methods and techniques for identification and counteracting cyber-attacks on 5G access network and applications” and by the Polish Ministry of Education and Science subvention.

REFERENCES

[1] The Ericsson Mobility Report, November 2022, https://www.ericsson.com/en/reports-and-papers/mobility-report

[2] M. Latva-aho, K. Leppänen (eds.), “Key drivers and research challenges for 6G ubiquitous wireless intelligence”, 6G Research Visions 1, Sept. 2019, ISBN 978-952-62-2353-7

[3] Alan Gatherer, „What Will 6G Be?”, *IEEE ComSoc Technology News*, June 2018, https://www.comsoc.org/publications/ctn/what-will-6g-be

[4] Li Xuanminand, Xiong Xinyi, “China aims to commercialize 6G by 2030: white paper” *Global Times*, 6th June, 2021, online: https://www.globaltimes.cn/page/202106/1225478.shtml

[5] C. Castro, “US House of Representatives Passes 6G Task Force”, *6G World*, Dec. 2nd, 2021, online: https://www.6gworld.com/exclusives/us-house-of-representatives-passes-6g-task-force/

[6] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, “Edge intelligence: Paving the last mile of artificial intelligence with edge computing,’’ 2019. *Proceedings of the IEEE*.

[7] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, P. Hui, „A Survey on Edge Intelligence”, arXiv print arXiv:2003.12172, 2020

[8] E. Peltonen, et.al., “6G White Paper on Edge Intelligence” [White paper]. (6G Research Visions, No. 8). University of Oulu. http://urn.fi/urn:isbn:9789526226774

[9] B. Kopras, B. Bossy, F. Idzikowski, P. Kryszkiewicz, H. Bogucka, “Task Allocation for Energy Optimization in Fog Computing Networks with Latency Constraints”, *IEEE Transactions on Communications*, Vol. 70, No. 12, Dec. 2022, pp. 8229 – 8243,

[10] B. Kopras, F. Idzikowski, B. Bossy, P. Kryszkiewicz, H. Bogucka, “Communication and Computing Task Allocation for Energy-Efficient Fog Networks”, *Sensors* 2023, 23(2), 997, pp. 1-22,

[11] M. Wasilewska, H. Bogucka, “Machine Learning for LTE Energy Detection Performance Improvement”, *Sensors* 2019, *19*(19), 4348

[12] M. Wasilewska, A. Kliks, H. Bogucka, K. Cichoń, J. Ruseckas, G. Molis, A. Mackutė-Varoneckienė, T. Krilavičius, “Artificial Intelligence for Radio Communication Context-Awareness”, *IEEE Access*, Oct. 2021, vol. 9, pp. 144820 – 144856

[13] M. Wasilewska, H. Bogucka, H. V. Poor, “Secure Federated Learning for Cognitive Radio Sensing”, *IEEE Communications Magazine*, March 2023,

[14] Janu, D., Singh, K., & Kumar, S. (2022). Machine learning for cooperative spectrum sensing and sharing: A survey.*Transactions on Emerging Telecommunications Technologies*, *33*(1), e4352.

[15] Chen, W. E., Fan, X. Y., & Chen, L. X. (2019, August). A CNN-based Packet Classification of eMBB, mMTC and URLLC Applications for 5G. In *2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA)* (pp. 140-145). IEEE.

[16] Shai Shalev-Shwartz, S.B.D. Understanding Machine Learning: From Theory to Algorithms; Cambridge University Press, 2014

[17] Cover, T.; Hart, P. Nearest neighbor pattern classification. IEEE Transactions on Information Theory 1967, 13, 21–27.

[18] Adesina, D., Hsieh, C. C., Sagduyu, Y. E., & Qian, L. (2020). Adversarial machine learning in wireless communications using RF data: A review. *arXiv preprint arXiv:2012.14392.*

[19] Liu, J., Nogueira, M., Fernandes, J., & Kantarci, B. (2021). Adversarial Machine Learning: A Multi-Layer Review of the State-of-the-Art and Challenges for Wireless and Mobile Systems. *IEEE Communications Surveys & Tutorials*.

[20] Sagduyu, Y. E., Shi, Y., Erpek, T., Headley, W., Flowers, B., Stantchev, G., & Lu, Z. (2020). When wireless security meets machine learning: Motivation, challenges, and research directions. arXiv preprint arXiv:2001.08883.

[21] Saghiri, A. M., Vahidipour, S. M., Jabbarpour, M. R., Sookhak, M., & Forestiero, A. (2022). A Survey of Artificial Intelligence Challenges: Analyzing the Definitions, Relationships, and Evolutions.*Applied Sciences*, *12*(8), 4054.

[22] V. Melnykov et.al., “MixSim: Simulating Data to Study Performance of Clustering Algorithms” https://cran.r-project.org/web/packages/MixSim/MixSim.pdf