Computer Communications 243 (2025) 108312

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Beyond performance comparing the costs of applying Deep and Shallow Learning[☆]

Rafael Teixeira[*], Leonardo Almeida, Pedro Rodrigues, Mário Antunes, Diogo Gomes, Rui L. Aguiar

Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Campus Universitário, Aveiro, 3810–193, Aveiro, Portugal
Departamento de Eletrónica Telecomunicações e Informática, Universidade de Aveiro, Aveiro, 3810–193, Aveiro, Portugal

ARTICLE INFO

Dataset link: https://github.com/rgtzths/shallow_vs_deep_learning_extended

Keywords:
Explainability
Shallow learning
Deep learning
Machine learning cost
5G
6G

ABSTRACT

The rapid growth of mobile network traffic and the emergence of complex applications, such as self-driving cars and augmented reality, demand ultra-low latency, high throughput, and massive device connectivity, which traditional network design approaches struggle to meet. These issues were initially addressed in Fifth-Generation (5G) and Beyond-5G (B5G) networks, where Artificial Intelligence (AI), particularly Deep Learning (DL), is proposed to optimize the network and to meet these demanding requirements. However, the resource constraints and time limitations inherent in telecommunication networks raise questions about the practicality of deploying large Deep Neural Networks (DNNs) in these contexts. This paper analyzes the costs of implementing DNNs by comparing them with shallow ML models across multiple datasets and evaluating factors such as execution time and model interpretability. Our findings demonstrate that shallow ML models offer comparable performance to DNNs, with significantly reduced training and inference times, achieving up to 90% acceleration. Moreover, shallow models are more interpretable, as explainability metrics struggle to agree on feature importance values even for high-performing DNNs.

1. Introduction

In recent years, mobile network traffic has experienced tremendous growth, accompanied by a similar rise in complex applications that require time-sensitive or high-throughput capabilities, such as self-driving cars and virtual/augmented reality. To meet the diverse demands of various users, the rigid framework of the 3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE) had to evolve [1]. As a result, the Fifth-Generation Network (5G) was developed as a highly adaptable and virtualized system, offering a cost-effective solution for delivering on-demand services.

In addition to its enhanced flexibility,5G must support data volumes up to 1000 times greater than previous networks, ultra-low latency of 1 millisecond, and massive device connectivity accommodating up to a hundred times more devices. Moreover, these capabilities must be delivered with exceptional reliability of 99.999% [2]. The convergence of these demanding requirements and the increased complexity of the network render traditional approaches to network design, deployment, and optimization impractical.

Artificial Intelligence (AI) is being proposed as a key enabler for meeting the stringent requirements of 5G while also enhancing energy efficiency and security. Among AI techniques, Deep Learning (DL), a specialized subset, has gained significant traction [3]. Its rising prominence stems from its success in fields like Natural Language Processing (NLP) and Computer Vision. The success of DL coupled with the highly heterogeneous data present in telecommunication tasks, makes it the common choice instead of the traditional shallow Machine Learning (ML) models [3].

In Beyond Fifth-Generation Networks (B5G)/Sixth-Generation Network (6G), AI plays an even more pivotal role, serving as a key enabler for meeting stricter performance demands and enhancing user experience [4,5]. Consequently, the effective implementation/deployment of AI/ML models in 5G is paramount, as it will significantly influence the application and evolution of AI/ML in future network technologies.

While larger Deep Neural Networks (DNNs) can be successfully employed in fields like NLP and Computer Vision, where resources and execution time are naturally unbounded. Their application in critical telecommunication tasks is often constrained by the limited resources and time available for training and inference. Furthermore, the scale of some telecommunication networks can prevent a single model from accessing data from all nodes, thus restricting the information available

☆ This article is part of a Special issue entitled: 'IoT-Cloud Continuum' published in Computer Communications.

* Corresponding author at: Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Campus Universitário, Aveiro, 3810–193, Aveiro, Portugal.
E-mail address: rafaelgteixeira@av.it.pt (R. Teixeira).

https://doi.org/10.1016/j.comcom.2025.108312
Received 22 November 2024; Received in revised form 17 June 2025; Accepted 29 August 2025
Available online 22 September 2025

and limiting the feasibility of deeper DNNs. In some cases, shallow ML models have been shown to deliver a performance comparable to DNNs, further questioning the practicality of employing deeper networks in such scenarios [6].

Nonetheless, current research on 5G and B5G typically only considers model performance when proposing a new model and comparing it with others, disregarding other relevant metrics that, when assessed, might highlight limitations for model deployment in real-world scenarios.

Considering that DL models may not meet the rigorous demands of 5G and B5G/6G networks, this paper evaluates current DL approaches against shallow ML models in common 5G tasks. Building on our previous work [7], which addresses these issues, we present an extended analysis that considers more datasets and additional model characteristics, such as explainability and energy costs, to ensure a thorough analysis of the different models.

The findings reveal that most shallow models deliver comparable performance while requiring significantly less training and inference time, often achieving more than 90% acceleration in both processes. Furthermore, aligned with the faster times, the shallow models also exhibit orders of magnitude lower energy consumption, highlighting that their execution times are directly related to less complex and less demanding models.

Regarding model explainability, the shallow ML models are inherently explainable as the simple linear and non-linear data transformations can be explained by looking at the weights attributed to each feature in the final decision boundary formula. While the DNNs explainability metrics often disagree about the importance of features, highlighting high degrees of disagreement.

This extended analysis shows how the costs of DNNs cannot be disregarded and how they might not be suitable for many 5G/B5G tasks, going against the current trend where DNNs are applied for most tasks.

In summary, the main contributions of this work are as follows: (i) a comprehensive evaluation of shallow ML models across various 5G related tasks; (ii) a comparison of state-of-the-art deep ML models with traditional shallow ML models in these tasks; and (iii) an in-depth discussion of the trade-offs between deep and shallow models;

The remainder of the paper is organized as follows. Section 2 briefly introduces network slicing, Quality of Experience (QoE) measurement, Intrusion Detection Systems (IDSs), and ML models and their explainability. Section 3 describes the experiments performed, the datasets used, the collected metrics, and the DL models. In Section 4, the results of the various experiments are presented. Section 5 discusses the results obtained and highlights the problems in ML development and future research directions. Finally, Section 6 concludes the work by giving the essential findings and possible extensions of this work.

2. Background

This section provides the necessary background regarding the tasks considered (Network Slice Attribution, QoE prediction, and IDSs) and ML, highlighting the shallow ML models used and ML models' explainability.

2.1. Network slicing

Initially proposed by the Next Generation Mobile Network Alliance [8], network slicing uses network function virtualization and software-defined networking to establish multiple independent virtual networks on a shared physical infrastructure [9]. These virtual networks, known as slices, can be tailored to provide specific capabilities and features, such as bandwidth, latency, and reliability. This approach is essential for cost-effectively addressing the diverse and heterogeneous needs of users, as building a single network capable of meeting all these requirements would be unfeasible [10].

For 5G, the 3GPP defined three primary service categories for network slicing: enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency Communications (URLLC), and massive Machine Type Communications (mMTC). The eMBB category supports high-throughput applications, such as high-definition video streaming or mobile TV, by providing networks with significant bandwidth. However, this profile assumes that applications are less sensitive to latency. In scenarios requiring connectivity for many devices, such as the Internet of Things (IoT), the mMTC profile is suitable. It prioritizes establishing connections between devices rather than offering high bandwidth or low latency. Finally, when ultra-low latency and high reliability are critical, the URLLC profile is the optimal choice [11].

In network slicing, machine learning models serve two primary purposes: (i) slice management, where they oversee resource allocation for each slice and determine the required number of slices for each type, and (ii) end-user slice attribution, where they decide which slice should handle specific end-user requests.

In this work, the latter task is addressed, where given a set of slice requirements, the ML model attributes the request to one of the existing slices.

Given the various slice types that are now available, Quality of Service (QoS) measurements must be made to ensure that the end-users receive the service they requested. The problem with QoS metrics is that they do not translate to the QoE the end-user perceives [12]. For example, although a video stream is not constantly buffering, it might be reproduced at a lower quality or vice versa. If the video stream traffic is unencrypted, measuring the visual quality through Peak Signal-to-Noise Ratio (PSNR)-based metrics is easy. However, the same cannot be said for buffering and restarting events.

With this in mind, besides slice attribution tasks, in this work, we will also look at QoE prediction based on QoS metrics, as it is a crucial task in 5G and B5G to ensure user satisfaction. Furthermore, this task requires the minimal possible overhead on communications, as one model is expected to monitor hundreds of thousands of communications simultaneously.

2.2. Intrusion detection systems

Although many IoT devices have limited security capabilities due to resource constraints, IoT networks are growing increasingly complex. This complexity poses a significant challenge for security experts, making it more difficult to monitor and safeguard.

A promising method for securing IoT networks is using IDS powered by ML algorithms. These ML-based IDS solutions can analyze network traffic, distinguish between regular and malicious activity, and identify the specific type of attack when malicious traffic is detected.

However, the dynamic nature of IoT networks, with devices frequently added or removed, makes achieving a reliable IDS challenging due to issues like traffic signature drift. To address this, the models must be continuously retrained to adapt to new attack types and evolving benign traffic patterns, minimizing the risk of false positives and negatives.

2.3. Machine learning models

Machine learning models can be classified into two main categories: shallow and deep learning. Shallow models refer to those developed primarily before 2006, including Artificial Neural Networks (ANNs) with only a single hidden layer [13]. A key characteristic of these algorithms is their relatively simple architecture, typically consisting of one or a few processing layers. These models are commonly used for traditional ML tasks and are valued for their interpretability and efficiency. In contrast, deep learning models are ANNs with multiple hidden layers, commonly referred to as DNNs. These models employ ANNs with multiple layers to learn complex patterns from data.

Shallow models often struggle with complex tasks without the aid of feature engineering. However, their primary advantage is faster training and inference when suitable features are provided. In our experiments, we utilized six shallow models: Gaussian Naive Bayes (GaussianNB), K-Nearest Neighbors (k-NN), Decision Tree (DT), Random Forest (RF), AdaBoost, and Gradient Boosting. These models can be categorized into individual models (the first three) and ensemble models (the last three).

The k-NN algorithm classifies data points by examining the closest k neighbors and applying a voting mechanism [14]. The class with the highest number of votes is assigned as the label for the new data point. Its performance relies on the number of neighbors considered and their impact on the classification decision.

The DT algorithm makes class inferences using a tree structure, where nodes assess different features [15]. It is highly interpretable and flexible, allowing adjustments such as limiting tree depth to balance accuracy and inference speed. The quality of the tree is influenced by factors like the splitting criterion and the features considered during the process.

The final individual classifier, GaussianNB, uses Bayes' theorem while assuming independence between features, which is considered a "naive" assumption [15]. In practice, the model calculates and outputs the probability that each example belongs to a particular class based on the given features.

Ensemble classifiers seek to build more robust and comprehensive models by combining multiple individual classifiers. In the experiments, we used one bagging classifier and two boosting classifiers: RF, Gradient Boosting, and AdaBoost.

RF constructs multiple DTs, each trained on different subsets of the data, to mitigate overfitting [14]. The inferences of these trees are aggregated through a weighted voting system based on probability estimates, with the class having the highest average probability being selected as the final inference.

Gradient Boosting is quite similar to RF, but instead of training the DTs independently, they are trained sequentially. The output from the previously combined trees is used to calculate the pseudo-residuals, which serve as the target values for the next tree in the sequence [16].

Lastly, AdaBoost, like Gradient Boosting, trains multiple models sequentially. However, the key difference lies in how the subsequent models are developed. Instead of creating new target values at each iteration, AdaBoost focuses on training models that specialize in correcting the errors of the previous model [16].

2.4. Explainability

Since most shallow models create linear or simple non-linear data transformations, explaining them can usually be done by examining the weights assigned to each feature. For example, in Logistic Regression (LR) and Support Vector Machine (SVM), the model can be easily explained by showing where the decision boundary lies and analyzing the weights the model assigns to each feature.

In other classifiers, the modes are inherently explainable based on how they perform classification. k-NN is based on an intrinsically explained voting system, with the only need to explain how the distance between data points is calculated. Similarly, DT is a set of if and else, which easily explains the model's decisions given any possible input. Finally, since GaussianNB is a probability-based classifier, it is a matter of producing the probabilities of the sample belonging to one of the classes given the pre-calculated probabilities for the features.

The voting classifiers can easily be explained since the models used in them are also easily explained, meaning that the main difference is that to explain the decision of a voting classifier, it is first required to explain the decision of any model it used and then present the results of the votes.

This approach to explainability changed when DNNs emerged, as the number of parameters and non-linearities applied to the data made it extremely hard to understand what input features are relevant and how they are used to obtain the output. Since DNNs are not inherently explainable and usually seen as a black-box model, Explainable Artificial Intelligence (XAI) algorithms gained traction.

XAI algorithms play a crucial role in addressing the transparency and interpretability challenges associated with black-box ML models [17]. These algorithms are typically categorized based on when they deliver explanations, the types of models they are designed to explain, and the aspects they aim to clarify [18], specifically:

- **Timing**: Methods that offer explanations during or before training are referred to as *ante-hoc* methods, while those that explain the model after it has been trained are called *post-hoc* methods.
- **Model Specificity**: Some methods are model-specific, designed for a particular type of model, such as DNNs. Others are model-agnostic, meaning they can be applied to explain any model regardless of its architecture.
- **Scope**: XAI methods can also differ in the focus of their explanations. Local methods provide insights into specific inferences or examples, while global methods aim to explain the model's behavior as a whole.
- **Insights**: XAI methods can be grouped based on the type of insights they provide about models. Currently, there are three main groups of XAI methods: sensitivity analysis, gradient/ backpropagation, and explanation by simplification, with some methods overlapping across these categories.

Focusing on the insights they provide, sensitivity analysis methods such as Local Interpretable Model-agnostic Explanations (LIME) [19] and SHapley Additive exPlanations (SHAP) [20] calculate feature importance scores to reveal how input features influence the model's output. These methods operate in two ways: they can analyze a set of inferences to generate a global explanation of the model or focus on a single example, altering specific feature values and observing the impact on the final classification to provide a local explanation. Understanding feature importance is crucial for auditing model decisions, as it sheds light on the factors driving the results and helps identify whether undesirable features, such as gender or race, play a disproportionately significant role in the outcomes.

Gradient-based methods, in contrast, derive feature importance by analyzing the information flow during backpropagation. These approaches are prevalent for explaining Convolutional Neural Networks (CNNs), as they can produce visual explanations, such as heatmaps of neurons or feature attributions, highlighting the impact of each pixel on the model's final decision [18].

Explanation by simplification seeks to make a complex black-box model understandable by approximating it with a simpler, inherently interpretable model [21]. A common technique involves rule-based approaches that translate the model's decisions into logical rules, making them easier to comprehend. DTs are often favored for this purpose, as they decompose the decision-making process into sequential decision paths based on binary choices.

This study focuses on model-agnostic approaches that provide feature relevance and offer broadly applicable results. Since the experiments involved tabular datasets, two XAI metrics were utilized: Permutation Importance (PI) and Partial Dependence Variance (PDV).

PI is a model-agnostic method introduced in [22] that offers insights into the significance of features in a model through sensitivity analysis. It works by measuring the increase in inference error when the values of a feature are shuffled, disrupting the relationship between that feature and the actual outcome. The more significant the increase in inference error, the more influential the feature is to the model.

Similar to PI, PDV [23] is a model-agnostic technique used to assess global feature importance or the interaction between two features. As the name implies, a single positive value represents both feature importance and feature interactions. At its core, this technique builds on the Partial Dependence method [24], which examines the marginal effect that one or two features have on the predicted outcome of a machine learning model.

We use two explainability metrics in this work to assess the difference between the explanations they provide for the various DNNs used. Since we do not know precisely the importance of each feature when using DNNs, it is essential to assess whether the XAI metrics provide similar results, as this can improve trust in the results. If the XAI metrics yield significantly different results, it is challenging to determine which is correct, indicating that the model remains unexplained.

3. Experiments

Given that most papers on network slicing attribution and IDS focus on DNNs, and many do not consider the training or inference costs, there is a significant gap in the current research. To address this, the experiments evaluate the feasibility of using shallow models for these tasks and their associated costs. This section outlines the datasets used, the hyperparameter optimization process, the evaluation metrics considered, and the state-of-the-art DNNs used for comparison.

The code used to run the experiments is available on GitHub.[1] All experiments were performed on a virtual machine with 64 vCPUs and 256 GB of RAM, running Ubuntu 24.04, inside a Kunpeng server. The DNNs and the shallow models were implemented in Python using the Keras API from Tensorflow and the Scikit-Learn library, respectively.

3.1. Datasets

Ten datasets were selected for three different tasks (network slice attribution, activity classification, and intrusion detection) to ensure generality in the results presented. This section provides a brief description of each dataset, along with its location.

3.1.1. Network slicing

The first dataset used, Slicing5G, is publicly available on IEEE DataPort[2] and has been used in several publications [1,9,25]. The dataset consists of eight input features and one output. The inputs include device and network Key Performance Indicators (KPIs)/Key Quality Indicators (KQIs), while the output corresponds to one of three classes (eMBB, URLLC, or mMTC). These classes represent three distinct network slices, each aligned with a different QoS profile, as discussed in Section 2.1.

The dataset contains eight features and 466,739 examples. The examples were then shuffled and divided into two sets with a ratio of 80:20 for training and testing. The final dataset comprises 373,391 training examples and 93,348 testing examples. A more detailed dataset description can be found in the original paper [1].

The second dataset, NetSlice5G, is publicly available on Kaggle[3] and is already split into training and testing sets. However, the testing dataset lacked the expected labels for the input, so it was discarded as it was impossible to derive accuracy metrics from it. This dataset contains features similar to those of the first, with the distinction that its categorical features have already been preprocessed into one-hot encoding. The only additional step performed was splitting the data using the same ratio as before, resulting in 25,266 training and 6317 testing examples.

The third dataset related to network slicing is KPI-KQI [26] and is available in Zenodo.[4] Similarly to the two previous datasets, this one also focuses on classifying services based on their requirements. However, it differs in the grouping performed, as instead of categorizing the services according to one of the three slice types, it classifies the services into one of 9 service categories (*UHD Video Streaming*, *Immerse Experience*, *Smart Grid*, *ITS*, *Vo5G*, *eHealth*, *Connected Vehicles*, *Industry Automation*, *Surveillance*).

This dataset is the smallest in the considered pool, containing only 165 samples, which, after division, comprise 132 training and 33 testing samples. Regarding the features available, the dataset includes 13 features of network KPIs like latency, jitter, bit rate, and packet loss rate.

The QoS-QoE [12] is publicly available dataset[5] that aims to predict QoE metrics based on QoS values. The dataset focuses on video streaming and aims to predict the perceived video quality based on several QoS metrics from the network.

The dataset comprises 47 features and four possible QoE to measure. Using the same approach as the original paper, we will focus on inferring the *StallLabel*, which is one of three values: *NoStall*, *MildStall*, or *SevereStall*. The dataset comprises 69,129 examples, and after splitting, it contains 55,303 training and 13,826 testing examples.

The last dataset in this section is UNAC [27] and is available at GitHub.[6] In this dataset, the objective is to classify browsing activity based on network traffic, which is crucial for correctly placing users in different slices, thereby reducing the possibility of misplacing users due to incorrect requirements.

The dataset contains three possible activities, 21 network features, and 382 examples. After division, the dataset consists of 305 training and 77 testing examples.

3.1.2. Intrusion detection systems

Several well-known datasets, such as KDD988 [28], KDDCUP99 [29], and NSLKDD [30], are commonly used for training and testing IDS models. However, as noted in [31], while these and other similar datasets are large and widely trusted by the research community, they are outdated and do not adequately represent current network traffic. This limits their relevance and applicability in modern network security studies.

Considering this, we chose to use recent and widely recognized state-of-the-art datasets, UNSW-NB15 [32], IoT-DNL,[7] N_BaIoT [33], IoTID20 [34], RT-IOT [35], which contain network traffic data from IoT devices.

The UNSW-NB15 dataset was synthetically generated by the Australian Centre for Cyber Security (ACCS). It combines real, modern, typical network traffic with synthesized attack activities, comprehensively representing contemporary network behaviors. The dataset contains over 2 million records with 49 features. It is suitable for training binary and multiclass classification models, as each malicious activity is categorized into one of nine attack types: DoS, Analysis, Backdoor, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, and Worms.

To simplify preprocessing, the NF-UNSW-NB15 version of the dataset was used. This dataset, described in [36], is already preprocessed and was published by the University of Queensland and is available on their website.[8] The dataset division followed the same ratio as the network slicing attribution datasets, resulting in 1,912,220 training and 478,055 testing examples.

The IoT-Device-Network-Logs (IoT-DNL) dataset is designed explicitly for network-based IDS. Initially introduced by Sahil Dixit and available on Kaggle, this open-source dataset is intended to evaluate anomaly-based IDS in wireless environments. It is flow-based, labeled, and focuses on IoT devices, with preprocessing already applied to meet the needs of network-based IDS applications.

The network logs for this dataset were collected using ultrasonic sensors, specifically Arduino and NodeMCU devices equipped with the

[1] https://github.com/rgtzths/shallow_vs_deep_learning_extended.

[2] https://ieee-dataport.org/open-access/crawdad-umkcnetworkslicing5g.

[3] https://www.kaggle.com/datasets/amohankumar/network-slicing-in-5g.

[4] https://zenodo.org/records/4779074.

[5] http://jeremie.leguay.free.fr/qoe/.

[6] https://github.com/Montimage/activity-classification.

[7] https://www.kaggle.com/datasets/speedwall10/iot-device-network-logs.

[8] https://espace.library.uq.edu.au/view/UQ:ffbb0c1.

ESP8266 Wi-Fi module. These devices transmit data to a server over Wi-Fi. The dataset contains a total of 477,426 samples, each featuring 14 attributes. The dataset division followed the same ratio as the remaining datasets, ending with 381,940 training and 95,486 testing examples.

The B_BaIoT is an emulated dataset designed to replicate a typical organizational workflow. To create it, the authors deployed nine IoT devices and two botnets that generate malicious attacks in addition to the benign traffic flow. Since the data recorded included traffic before, during, and after the botnets were installed, the dataset comprises typical and malicious flows. The two botnets perform multiple attacks, allowing us to identify malicious traffic and determine the type of attack being performed.

In this scenario, we followed the same approach as the original authors, classifying the traffic as either malicious or benign. The dataset contains 7,062,606 examples and 23 preprocessed features from raw traffic. When divided, it contains 5,650,084 training and 1,412,522 testing examples.

Similarly to B_BaIoT, IoTID20 was also generated in a testbed. However, IoTID20 aims to replicate a typical smart home environment with two IoT devices and various laptops, smartphones, and tablets that are connected to them and act as attackers. This dataset also offers the option to perform either binary classification (detecting either malicious or non-malicious data) or multiclass classification (detecting the type of attack being performed).

Since the authors considered all possible options when training the models in the original approach (with results for both binary and multiclass classification), we opted for multiclass classification in this paper. This means that the dataset comprises 625,783 examples, 80 features, and nine possible classes. After division, the dataset consists of 500,626 training and 125,157 testing examples.

Regarding preprocessing, it is also necessary to mention that only 27 of the 80 features were kept for model training. The other features were removed, as the authors in the original paper also removed them due to their low relevance for model training.

The RT-IOT dataset, available at Kaggle[9] is another example of a dataset based on a real testbed. Similar to the previous datasets, the testbed comprises both malicious and benign users, and the malicious users conduct various attacks.

The dataset used is a preprocessed version of the original paper, which contains 83 features, 12 different types of network traffic attacks, and 123,117 examples. After division, the dataset contained 98,493 training and 24,624 testing examples.

3.1.3. Preprocessing

As a preprocessing step for every dataset, categorical features were label-encoded, and examples with missing values were dropped. Standard normalization was applied to enhance model convergence after dividing the dataset into training and testing sets.

3.2. Shallow models optimization

To ensure a fair comparison between deep and shallow models, some shallow models required optimization, as they were not pre-implemented for these datasets. This optimization was conducted using a simple grid search with five-fold cross-validation. Only the training set was used during this process, focusing on the most critical and common hyperparameters and testing a limited range of values. This streamlined approach was intentional, as the study aimed to compare shallow and deep models rather than performing extensive model optimization.

[9] https://www.kaggle.com/datasets/joebeachcapital/real-time-internet-of-things-rt-iot2022.

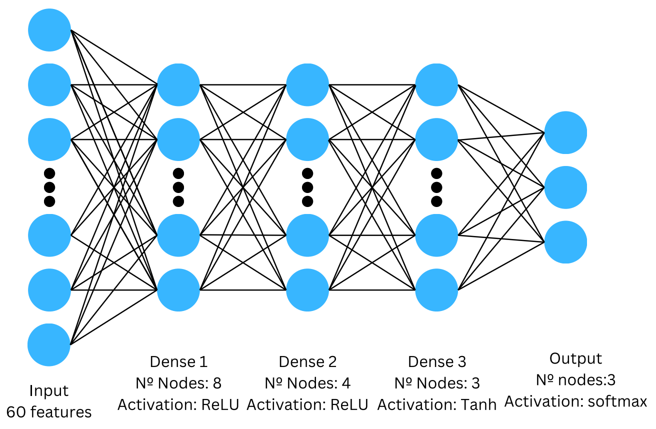


Fig. 1. Architecture of the DEEPSLICE model proposed in [1] and implemeted for the Slicing5G dataset.

The selection of the best hyperparameters followed the same criteria used for model comparison: first, the model performance was considered; if the performance was the same, then training time was considered; finally, when training time was the same, the inference time was considered. If all three values were the same, the first hyperparameter combination was considered.

Nonetheless, after running the hyperparameter optimization, while some models had the same performance, no two models had the same running time. Hence, only the first two selection criteria were necessary.

A complete list of evaluated hyperparameters and tested values is available on the paper's GitHub[1] and presented in Appendix A. Hyperparameters that were not explicitly evaluated retained their default values (refer to the Scikit-learn documentation for details[10]).

3.3. Deep learning models

Ideally, the DNNs models used in the experiments would be those specifically proposed for the respective datasets. However, some datasets lacked models previously proposed for them. To address this issue, we considered the models proposed for the other datasets as a pool of possible architectures and selected the one that performed better for that dataset, taking into account both performance, training time, and inference time.

To use the networks proposed for the other datasets in the datasets without networks, the input and output layers needed to be changed to match the expected input and output of the dataset. Other than that, no other change was performed. Devising and proposing a DNNs model for each dataset is outside the scope of this work.

3.3.1. Network slicing

In the slice attribution problem, most ML models implemented are DL ones [1,9,25]. However, some cases have proposed shallow learning models [9]. The deep learning model implemented for the Slicing5G dataset was explicitly designed for it [1] and is a simple DNN with three dense layers with few nodes each, as depicted in Fig. 1.

For the NetSlice dataset, the model used is presented in our previous work [7] and depicted in Fig. 2. The model is a simple shallow neural network with a single hidden dense layer with 73 nodes, followed by a dropout layer.

The model used in KPI-KQI was introduced in the original paper and is shown in Fig. 3. It consists of a straightforward DNN with a single hidden layer containing 100 nodes, employing the ReLU activation function.

[10] https://scikit-learn.org/stable/

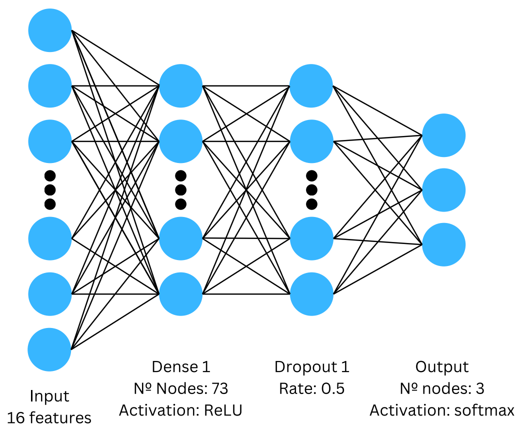


Fig. 2. Architecture of the model implemented for the NetSlice5G dataset.

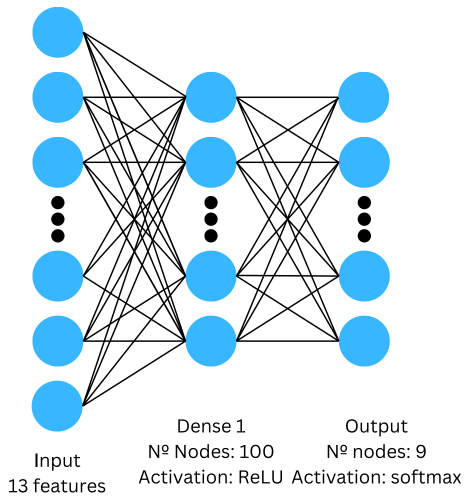


Fig. 3. Architecture of the model implemented for the KPI-KQI dataset.

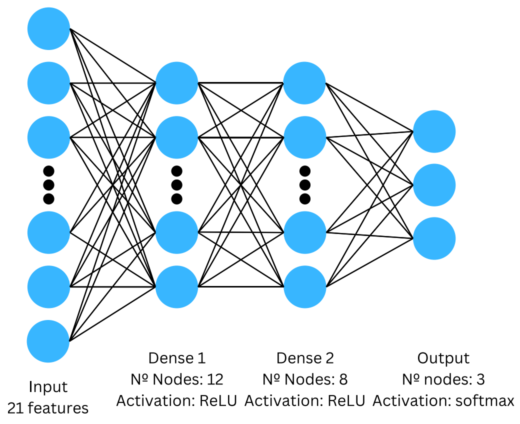


Fig. 4. Architecture of the model implemented for the UNAC dataset.

Similarly to the model proposed in KPI-KQI, the model for the UNAC was proposed in the original paper and is depicted in Fig. 4. This model is also a simple DNN, although it is composed of two hidden layers instead of one, each containing 12 and 8 nodes, respectively. The hidden layers also use the ReLU activation function.

The QoS-QoE dataset is the first dataset for which a model has not been proposed. After running all the other DNNs, the best performing

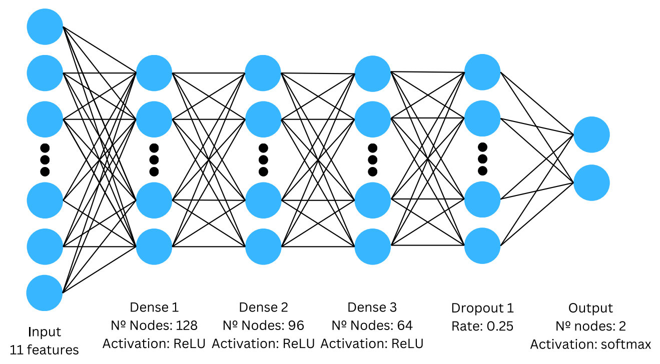


Fig. 5. Architecture of the model proposed in [37] and implemeted for the UNSW-NB15 dataset.

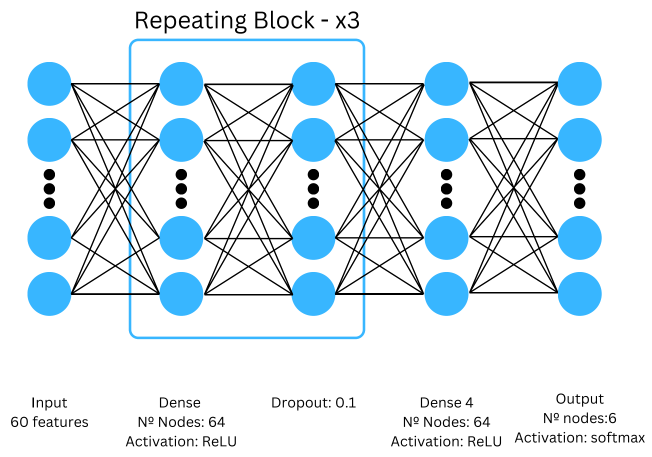


Fig. 6. Architecture of the model proposed in [38] and implemeted for the IoT-DNL dataset.

model was the one proposed for the NetSlice5G dataset depicted in Fig. 2.

3.3.2. *Intrusion detection systems*

Starting with the UNSW-NB15 dataset, the authors in [37] evaluated three different DNNs: Multi-layer Perceptron (MLP), CNN, and Recurrent Neural Network (RNN). Since all models performed comparably well, we opted for the MLP model for its simplicity and faster training time. As shown in Fig. 5, the model features three hidden layers with 128, 96, and 64 neurons, respectively, and includes a dropout layer with a dropout rate of 0.25 following the hidden layers. The Rectified Linear Unit (ReLU) activation function is applied in the hidden layers, and the Adam optimizer is used for training.

In [38], the authors utilized an MLP for the IoT-DNL dataset but did not provide details about the number of neurons in each layer. To replicate their approach, we tested 18 different configurations of hidden layers and neuron counts, selecting the configuration that delivered the best performance. The final model, shown in Fig. 6, comprises four hidden layers with 64 neurons each and dropout layers (dropout rate of 0.1) between each hidden layer. The ReLU activation function was applied in the hidden layers, and the model was trained using the Adam optimizer. The evaluation considered training, validation accuracy, and the model's complexity.

Similarly to the QoS-QoE dataset, the N_BaIoT and IoTID20 did not have a proposed model that applied to this scenario. After running the existing models to understand which performed better, the results indicated that the UNSW model was the superior model for the IoTID20, and the IoT-DNL was the best for the N_BaIoT dataset. The models

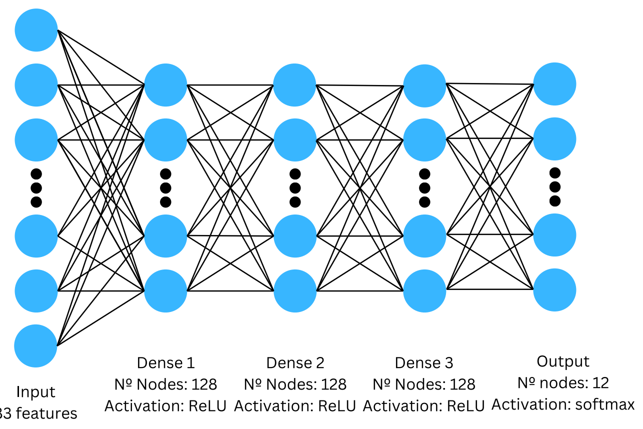


Fig. 7. Architecture of the model proposed in [39] and implemeted for the RT-IoT dataset.

are depicted in Figs. 5 and 6 and were previously addressed when overviewing the models for those datasets.

The only model left to address is the one proposed for the RT-IOT dataset. The model was proposed in [39] and is depicted in Fig. 7. This network is a DNN composed of three hidden layers, each with 128 nodes and the ReLU activation function.

3.4. Model evaluation

Evaluating a model based solely on its performance can create a misleading perception of success. In B5G networks, where strict latency requirements are crucial, models must predict accurately and operate quickly and efficiently to ensure the network can efficiently process requests. Additionally, the dynamic nature of B5G networks necessitates models that can adapt rapidly. Therefore, training and inference times, as well as energy consumption, are critical metrics to assess alongside performance. Furthermore, the new EU AI Act[11] highlights the need for models to be explainable to be applicable, so besides performance, execution time, and energy costs, model explainability was assessed.

Nonetheless, performance remains a critical metric by which the models need to be evaluated. With this in mind, the model's performance was assessed using Mathews Correlation Coefficient (MCC). Given that not all datasets are balanced, we chose this metric because it penalizes misclassifications more heavily than traditional accuracy. Nonetheless, results were also obtained using two other popular metrics, F1-Score and Receiver Operating Characteristic - Area Under de Curve (ROC-AUC) score.

To accurately measure training and inference times, both processes were executed 30 times, and the results were averaged. All random generators were assigned a seed value of 42, ensuring consistency in the models' initial states and training processes. This setup ensured that any variation in training or inference times was attributable solely to the computing environment.

Using the training time of the DNNs as a baseline, the acceleration achieved by the shallow ML models was computed using Eq. (1). In this formula, DNN_{time} represents the time the DNN takes to train or infer, while $model_{time}$ refers to the corresponding time for a shallow model.

$$Acceleration = \left(1 - \frac{model_{time}}{DNN_{time}}\right) \times 100 \tag{1}$$

Regarding energy costs, to ensure reproducibility of the results, the approach presented in [40] was followed. In this approach, energy consumption is estimated based on the machine's raw performance, measured in MIPS using the 7-Zip LZMA benchmark, the time the machine takes to solve the task (either training or inference), and the average CPU utilization during task execution. Considering all these factors, the energy consumption is then calculated as if the task was executed on a benchmark machine, specifically the HP ProLiant ML110 G4 Server. For more details regarding energy costs, please refer to [40].

[11] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689.

Table 1
Accuracy results of the various models for slice attribution datasets using MCC as metric.

Model	Slicing5G	NetSlice5G	KPI-KQI	QoS-QoE	UNAC
DNN	1.00	1.00	0.93	0.99	0.81
k-NN	1.00	1.00	0.84	0.87	0.89
GaussianNB	0.87	1.00	1.00	0.84	0.87
DT	1.00	1.00	0.97	1.00	0.85
RF	1.00	1.00	1.00	1.00	0.93
AdaBoost	0.87	1.00	0.84	1.00	0.93
Grad. Boost.	1.00	1.00	1.00	1.00	0.93

For explainability, since the primary focus of the study is not on the specific XAI results but on the alignment of feature importance between different XAI metrics, we used the Pearson Correlation Coefficient (PCC) [41] to compare the feature importance values provided by PI and PDV.

The PCC quantifies the linear relationship between two variables, X and Y, with values ranging from +1 to −1. A value of +1 indicates perfect positive linear correlation, 0 represents no linear correlation, and −1 signifies perfect negative linear correlation. The formula for PCC is provided in Eq. (2).

$$PCC(x, y) = \frac{\sum (x_i - \bar{x}) \times (y_i - \bar{y})}{\sigma_x \times \sigma_y} \tag{2}$$

where $\bar{x}$ and $\bar{y}$ are the means of X and Y, respectively, and σ_x and σ_y are the standard deviations of X and Y, respectively.

4. Results

Since the paper focuses on comparing shallow and deep learning models, results regarding hyperparameter optimization or model selection for datasets without one are not the focus of this study. Nonetheless, the full scope of results is available in the paper's GitHub. Furthermore, the DNN selection results for datasets without one are presented in Appendix B.

As mentioned in the previous section, analyzing model performance solely based on accuracy metrics creates a biased view of the overall model constraints. This section addresses this issue by analyzing model performance, training and inference times, energy costs, and model explainability.

4.1. Model performance

Starting with the analysis model performance, Tables 1 and 2 provide the results for the network slicing and IDS datasets, respectively. After careful analysis, it is noticeable that each dataset's DNN does not outperform every shallow model, achieving the same performance in most cases.

In the network slicing datasets, it is also possible to see that in the QoE-QoS and KPI-KQI, the DNN is outperformed by some of the shallow ML models. In the QoS-QoE, since the model was not explicitly designed for that dataset, one could argue that the lower performance results from a lack of optimization. Nonetheless, 0.01 MCC is not a critical performance difference. In the KPI-KQI dataset, the result of the DNN is an evident lack of optimization by the original proposal, as the network lacks expressivity when compared with an RF. If the network had one more layer, it could have the non-linearity needed to represent the data distribution in the dataset accurately.

Table 2
Accuracy results of the various models for IDS datasets using MCC as metric.

Model	UNSW	IoT-DNL	B_BaIoT	IoTID20	RT-IOT
DNN	0.85	1.00	1.00	0.73	0.99
k-NN	0.86	1.00	1.00	0.74	0.99
GaussianNB	0.40	1.00	0.96	0.47	0.76
DT	0.81	1.00	1.00	0.62	0.99
RF	0.84	1.00	1.00	0.71	0.99
AdaBoost	0.73	1.00	1.00	0.56	0.83
Grad. Boost.	0.77	1.00	1.00	0.75	1.00

For the IDS datasets, a similar pattern is observed, as in some cases, the DNN is outperformed by shallow ML models, even in situations where the DNN was developed explicitly for that dataset. Nonetheless, the differences in performance are not that significant, not reaching 0.05 MCC.

The results for F1-Score and ROC-AUC score are not presented in this section as they show the same trends as MCC. Nonetheless, the results are available on the paper's GitHub and in Appendix C.

Since some datasets have a small number of examples, DNNs may overfit the training data. With this in mind, visualizing the learning curves might provide a better insight regarding the suitability of applying DNNs in such scenarios.

Analyzing Fig. 8, it is clear that no model significantly overfitted the training data, as the validation and training losses were always close to one another. Furthermore, since a callback saved the best-performing weights based on validation loss, even when the validation loss changed considerably, for example, in the IoTID20 dataset, the weights that achieved the lowest validation loss were the ones selected for the final model evaluation.

Although saving temporary weights during training incurs a higher training cost, it is usually the expected practice when training DNNs, as it helps reduce the effects of overfitting and provides a more well-rounded model.

4.2. Training/inference times

Since, in terms of performance, there is no clear winner between shallow and deep learning models, as most models either have the same performance or are within a margin of error. Other model characteristics must be addressed. This section addresses the training and inference times of the models.

The datasets are divided according to the task group they belong to, which consequently also divides them between larger-size datasets (IDS) and smaller-sized datasets (network slicing).

4.2.1. Network slicing

Considering that the network slicing datasets are smaller, both training and inference times are expected to be considerably smaller overall. Table 3 presents the results for training time and the acceleration obtained for the various shallow ML models. As it is clear, except for Gradient Boosting in the QoS-QoE dataset, the shallow ML models were orders of magnitude faster than the DNN. As we will see in the IDS datasets, Gradient Boosting tends to struggle in datasets with many features and examples.

Regarding the inference times presented in Table 4, the results are not as straightforward as in training. For example, k-NN is always slower than the DNN, and the RF struggles in the KPI-KQI and UNAC datasets. The remaining models exhibit a similar behavior to that observed during training.

The slower inference from k-NN is expected, as a lazy learner calculates the difference between every point in the training dataset with the point to be classified. This is also evident in the slower execution when datasets have many features or examples, as is the case with UNAC and Slicing5G. The RF results highlight that its inference time is directly related to the number of features a dataset has, meaning that to achieve the perfect scores overall, the number of trees created was higher, slowing down the overall inference time.

4.2.2. Intrusion detection systems

In the IDS datasets, the presented trends are very similar. Table 5 presents the training time results where the shallow models considerably outperformed the DNN. Some exceptions are AdaBoost in the B_BaIoT dataset and Gradient Boosting in the IoTID20 and RT-IOT datasets. Considering the performance results of the Gradient Boosting in these datasets, it is possible that the longer training time was the consequence of better fitting models, as those are the datasets where Gradient boosting outperforms the DNN in terms of MCC. The reason why AdaBoost was slower than the DNN might be related to the number of features in the B_BaIoT dataset, as it is the dataset with the most features, and AdaBoost appears to be slower to converge in datasets with more samples.

The inference time for IDS datasets presented in Table 6 highlights a similar trend to the one in slicing datasets, as k-NN is consistently slower than the DNN, and the shallow models generally outperform the DNN.

Gradient Boosting and AdaBoost are exceptions to these rules, which present slower results than the DNN for the IoTID20 (both) and RT-IOT (Gradient Boosting) datasets. This means that Gradient Boosting's slower training times can be due to slower sample processing rates, as the model is slower in both cases. The AdaBoost results show that although it is slower when training in the B_BaIoT (taking longer to converge), it still processes samples at a higher rate, a characteristic not observed in the IoTID20, which presents the opposite behavior. This may be a consequence of the dataset's large number of features, exhibiting a similar behavior to Gradient Boosting.

4.3. Energy consumption

Although the execution times of training and inference provide relevant information regarding the cost of executing a model, they do not give the complete picture. For example, there is no consideration regarding the use of a single or multi-core processing, nor the load posed on the machine. With this in mind, one way to improve the understanding of the cost of executing the different models is to measure the difference in energy consumption when running them. This subsection presents the results of that energy measurement.

4.3.1. Network slicing

Starting with the results regarding network slicing, Tables 7 and 8 show the energy consumption for training and inference, respectively. When comparing the results from these tables with those regarding training and inference times, it is clear that the energy consumption shows an even greater gap, as most models show several orders of magnitude less energy consumption than the DNNs.

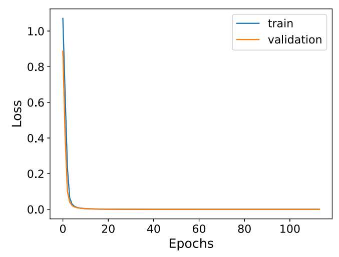
Nonetheless, in cases where the shallow models take longer, for example, the k-NN when inferring for the Slicing5G or NetSlice5G, or the Gradient Boosting when training the QoS-QoE, the energy consumption of these models is also higher, highlighting some of the disadvantages of these specific models.

Nonetheless, this analysis highlights that the reduced training and inference times of the shallow models are not due to better use of multiple cores, as otherwise the energy consumption of the shallow models would be higher. Instead, the reduced training and inference times are a consequence of a model that is less computationally demanding, highlighting one of the most significant advantages of the shallow ML models.

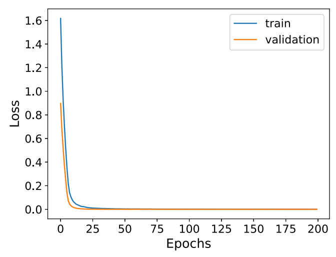
4.3.2. Intrusion detection systems

In IDS datasets as presented in Tables 9 and 10, the trend is similar.

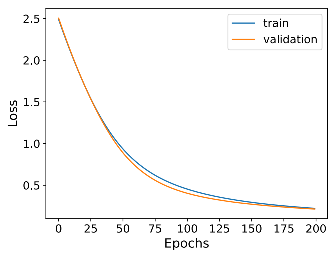
Although the cost per example is not as evident in the training energy costs, given the changes in network architecture between datasets and the fact that the networks required different epochs for training, in inference, it is easier to understand, as all the models process the same number of samples.



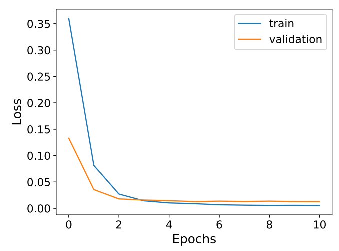
(a) Slicing5G Learning Curve



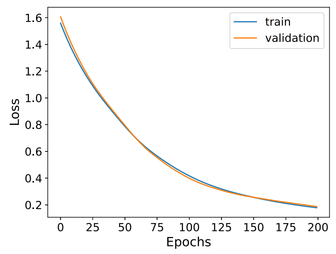
(b) NetSlice5G Learning Curve



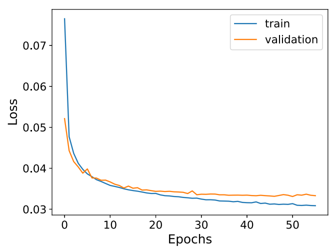
(c) KPI-KQI Learning Curve.



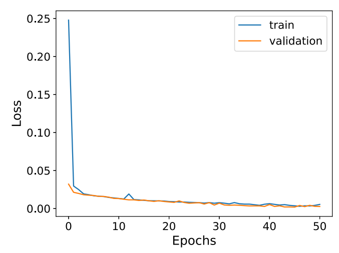
(d) QoS-QoE Learning Curve.



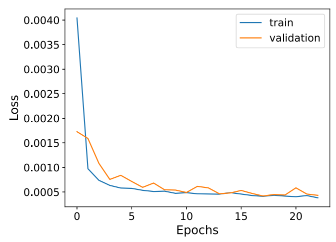
(e) UNAC Learning Curve.



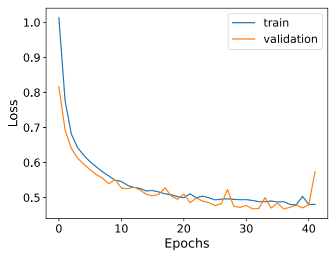
(f) UNSW Learning Curve.



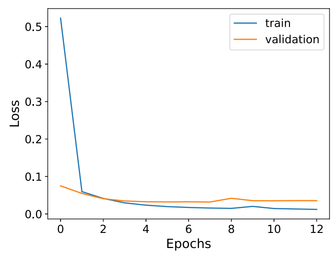
(g) IoT-DNL Learning Curve.



(h) B_BaIoT Learning Curve.



(i) IoTID20 Learning Curve.



(j) RT-IOT Learning Curve.

Fig. 8. Learning curves for the trained DNN of each dataset. The number of epochs varies as a result of early stopping.

Table 3
Training time results of the various models for slice attribution datasets. The results show the time in seconds and, in parentheses, the acceleration relative to the DNN model.

Model	Slicing5G	NetSlice5G	QoS-QoE	KPI-KQI	UNAC
DNN	76.6544(–)	7.2701(–)	75.8395(–)	3.0568(–)	3.2406(–)
k-NN	0.9293(99%)	0.0078(100%)	0.0397(100%)	0.0028(100%)	0.0045(100%)
GaussianNB	0.1981(100%)	0.0222(100%)	0.3296(100%)	0.0061(100%)	0.0083(100%)
DT	0.2766(100%)	0.0129(100%)	0.4927(99%)	0.0053(100%)	0.0073(100%)
RF	0.4777(99%)	0.1397(98%)	2.3669(97%)	0.0450(99%)	0.4223(87%)
AdaBoost	1.9766(97%)	0.1067(99%)	5.0403(93%)	0.0357(99%)	0.2642(92%)
Gradient boosting	3.7604(95%)	0.2141(97%)	122.5219(−62%)	0.9617(69%)	0.9395(71%)

Discarding the k-NN as it is a lazy learner (explained in detail when analyzing inference time), the cost-per-sample difference between DNNs and shallow models is orders of magnitude higher in most cases. It is also clear that, in some situations, for example, AdaBoost in B_BaIoT, although training was more expensive, the cost per sample of AdaBoost is considerably lower.

4.4. Explainability

Another aspect that has become crucial in recent years and is further emphasized by the EU AI Act is model explainability. As explained in Section 2, the shallow models used during the experiments are all inherently explainable, as their decision process is simple and easy to

Table 4
Inference time results of the various models for slice attribution datasets. The results show the time in seconds and, in parentheses, the acceleration relative to the DNN model.

Model	Slicing5G	NetSlice5G	QoS-QoE	KPI-KQI	UNAC
DNN	5.6943(–)	0.4629(–)	1.0136(–)	0.0760(–)	0.0767(–)
k-NN	8.1156(−43%)	0.5436(−17%)	1.3284(−31%)	0.1355(−78%)	0.4222(−450%)
GaussianNB	0.0424(99%)	0.0061(99%)	0.1176(88%)	0.0021(97%)	0.0026(97%)
DT	0.0099(100%)	0.0023(100%)	0.0091(99%)	0.0016(98%)	0.0027(96%)
RF	0.0361(99%)	0.0164(96%)	0.6905(32%)	0.1234(−62%)	0.6054(−689%)
AdaBoost	0.0852(99%)	0.0086(98%)	0.0483(95%)	0.0038(95%)	0.0159(79%)
Gradient boosting	0.0713(99%)	0.0078(98%)	0.2275(78%)	0.0052(93%)	0.0060(92%)

Table 5
Training time results of the various models for IDS datasets. The results show the time in seconds and, in parentheses, the acceleration relative to the DNN model.

Model	UNSW	IoT-DNL	B_BaIoT	IoTID20	RT-IOT
DNN	359.0939(–)	474.6060(–)	3107.5984(–)	523.1083(–)	192.4992(–)
k-NN	0.3678(100%)	0.0457(100%)	1.9659(100%)	5.4472(99%)	0.1117(100%)
GaussianNB	2.3827(99%)	0.4522(100%)	9.6579(100%)	0.8151(100%)	0.6041(100%)
DT	2.2790(99%)	0.4937(100%)	28.7028(99%)	1.5103(100%)	0.4505(100%)
RF	9.6083(97%)	2.3168(100%)	70.8945(98%)	6.5106(99%)	6.7084(97%)
AdaBoost	28.3139(92%)	9.8509(98%)	3669.4095(−18%)	217.6092(58%)	4.2111(98%)
Gradient boosting	674.7595(−88%)	44.8977(91%)	2348.3039(24%)	2683.3082(−413%)	1160.9591(−503%)

Table 6
Inference time results of the various models for IDS datasets. The results show the time in seconds and, in parentheses, the acceleration relative to the DNN model.

Model	UNSW	IoT-DNL	B_BaIoT	IoTID20	RT-IOT
DNN	14.3578(–)	5.9676(–)	88.6096(–)	7.9947(–)	1.8122(–)
k-NN	88.5907(−517%)	13.9572(−1385%)	2851.0803(−3118%)	32.1239(−302%)	3.4492(−90%)
GaussianNB	3.0264(79%)	0.2102(96%)	2.8237(97%)	1.0124(87%)	0.7414(59%)
DT	0.1159(99%)	0.0235(100%)	0.4194(100%)	0.0722(99%)	0.0256(99%)
RF	3.5526(75%)	0.5392(91%)	2.1981(98%)	2.1625(73%)	1.4896(18%)
AdaBoost	1.3075(91%)	1.0831(82%)	54.2710(39%)	10.9578(−37%)	0.2009(89%)
Gradient boosting	3.6002(75%)	0.3558(94%)	8.2365(91%)	15.6977(−96%)	6.5542(−262%)

Table 7
Energy consumption results for the model training of the various models for Network Slicing datasets. The results are expressed in kJ; missing values, presented as (–), are a consequence of fast execution times that do not allow for measuring the energy consumed.

Model	Slicing5G	NetSlice5G	KPI-KQI	QoS-QoE	UNAC
DNN	5.9072	222.6787	2.8743	157.8605	3.1627
k-NN	0.0554	–	–	–	–
GaussianNB	0.0117	–	0.0006	0.7400	0.0006
DT	0.0146	–	0.0006	1.3581	–
RF	0.0368	0.5439	0.0098	7.4547	0.0125
AdaBoost	0.1107	0.4867	0.0161	15.8077	0.0247
Grad. Boost.	4.6941	0.9522	0.0097	367.5373	0.0392

Table 8
Energy consumption results for the model inference of the various models for Network Slicing datasets. The results are expressed in kJ; missing values, presented as (–), are a consequence of fast execution times that do not allow for measuring the energy consumed.

Model	Slicing5G	NetSlice5G	KPI-KQI	QoS-QoE	UNAC
DNN	0.3590	2.1228	0.0090	4.4197	0.0110
k-NN	1.3076	3.4215	0.0061	15.5313	0.0126
GaussianNB	0.0026	–	–	–	–
DT	0.0004	–	–	–	–
RF	0.0030	–	0.0023	0.4717	0.0018
AdaBoost	0.0042	–	0.0016	–	0.0023
Grad. Boost.	0.0037	–	–	0.5491	0.0005

understand. Take, for example, the decision process presented in Fig. 9, for the DT of the Slicing5G dataset.

Anyone, even without AI knowledge, can understand the decision-making process of this model, as it can be easily presented as a set of sequential decisions. However, when considering DNN, the same does not apply, as it is not possible to explain how each feature and model weight impacted the final decision by simply examining the weights of the trained model. With this in mind, several approaches were developed to provide feature importance values to explain DNN behavior.

Nonetheless, since there is no way to compare the values predicted by the explainability metrics with the real importance that the DNN gives to each feature, there is no way to evaluate the correctness of the explanation. This, in turn, can lead to what is shown in Fig. 10, where two explainability metrics provide two considerably different sets of importances and there is no way to validate which set is the correct one.

To further highlight this problem, Fig. 11 presents the correlation between the feature importance results provided by the PI and PDV explainability metric for every dataset. In this scenario, the set of feature importances outputed by PI and PDV are considered two vectors, and the correlation metric evaluates the similarity in the direction of these two vectors. This means that the two metrics can provide different values of importance to each feature as long as the relative importance between features is maintained.

For example, in Fig. 10, the PI metric provided approximately 0.25 positive importance to *eth.dst*, while PDV provided 0.6. This would not impact the correlation as long as the values for the remaining variables

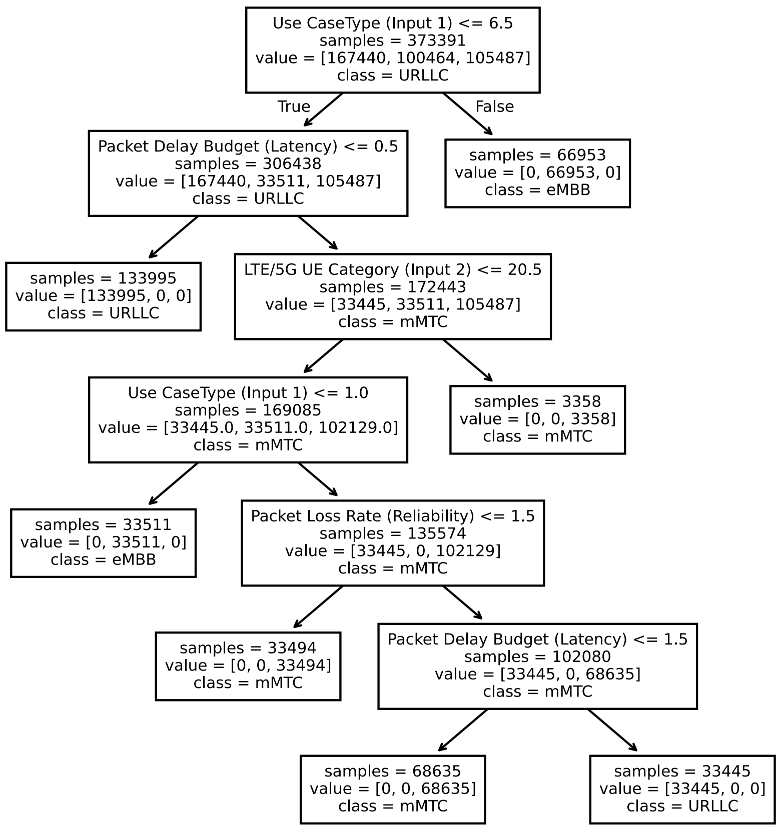
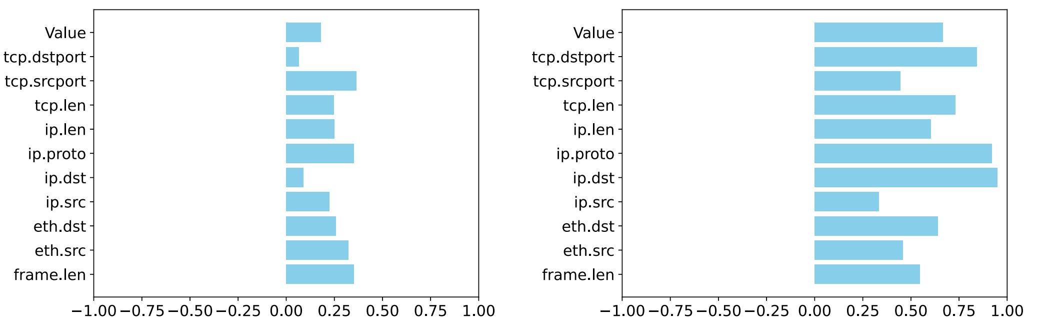


Fig. 9. DT decision diagram for inferring slice attribution for the Slicing5G dataset.



(a) Feature importance results for the PI metric. (b) Feature importance results for the PDV metric.

Fig. 10. Feature importance provided by PDV and PI for the DNN trained for the IOT-DNL dataset.

followed the same trend. However, this does not happen. For example, in PI, the *eth.src* has higher importance than *eth.dst*, while in PDV it has a lower importance. These differences are what change the direction of the vector considered in the correlation analysis.

Consequently, high correlation indicates that the two metrics assign reasonably similar importance to each feature, whereas near-zero values suggest that the metrics strongly disagree. Negative values mean that the metrics provide opposite importance to the features.

As we can see, NetSlice, Slicing5G, and UNAC appear to have significantly high correlations, and the remaining have near-zero or negative correlations, highlighting how even when using explainability metrics, the inner workings of DNNs might be very hard to uncover.

Given the exceptional performance of the DNN, we anticipated a clear and consistent feature importance ranking across different explainability methods. However, our findings revealed a different story. While the simplest datasets yielded similar results, more complex datasets showed a lower correlation between the metrics.

While it is natural for different explainability models to produce slightly varying results, we expected a higher consensus, especially for high-performing models (with MCC exceeding 0.8). Unfortunately, this was not the case for the most complex datasets.

This is highlighted by the poor performance of most IDS datasets, which have more samples and features, and the models typically do not solve them perfectly. Meaning that, unless there is a clear pattern to solve a dataset, the explainability metrics can disagree on which pattern the model found.

It is also notable that the DNNs for these more complex datasets also tend to be larger, meaning that the explainability models might

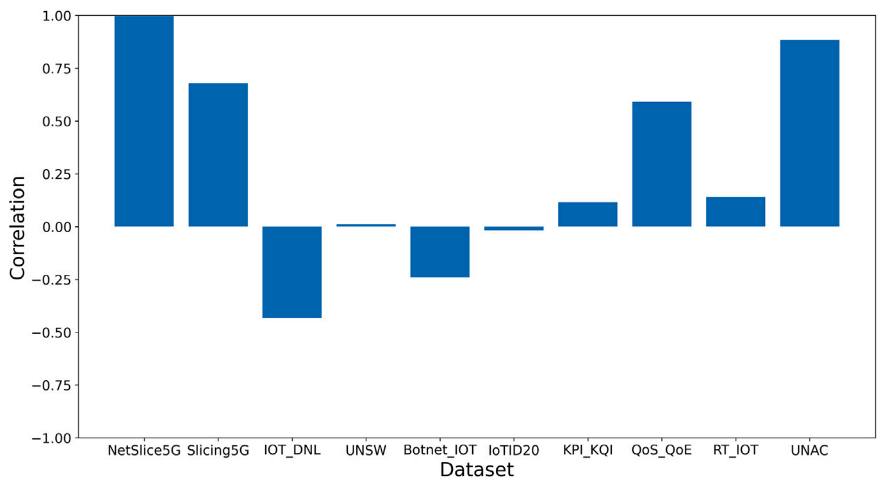


Fig. 11. Correlation between the explanation provided by the PI and PDV metrics.

Table 9
Energy consumption results for model training of the various models for IDS datasets. The results are expressed in kJ; missing values, presented as (–), are a consequence of fast execution times that do not allow for measuring the energy consumed.

Model	UNSW	IoT-DNL	B_BaIoT	IoTID20	RT-IOT
DNN	60.377	69.824	391.878	98.948	4.630
k-NN	0.022	0.002	0.103	0.305	0.004
GaussianNB	0.134	0.029	0.677	0.043	0.030
DT	0.168	0.026	1.609	0.079	0.015
RF	1.683	1.404	132.610	15.143	1.147
AdaBoost	1.587	0.552	205.655	13.721	2.614
Grad. Boost.	37.818	2.516	131.613	150.389	34.985

Table 10
Energy consumption results for model inference of the various models for IDS datasets. The results are expressed in kJ; missing values, presented as (–), are a consequence of fast execution times that do not allow for measuring the energy consumed.

Model	UNSW	IoT-DNL	B_BaIoT	IoTID20	RT-IOT
DNN	0.9053	0.3554	5.2766	0.4761	0.1091
k-NN	258.1884	42.0455	8349.0970	100.8233	4.0349
GaussianNB	0.1696	0.0118	0.1879	0.0568	0.0372
DT	0.0057	0.0016	0.0234	0.0000	0.0005
RF	0.2611	0.0680	0.3696	0.3409	0.0288
AdaBoost	0.0733	0.0569	3.0416	0.6143	0.0651
Grad. Boost.	0.2017	0.0198	0.4616	0.8248	0.1173

also struggle to explain larger models, as they are capable of generating more complex non-linearities that the explainability metrics can misinterpret.

5. Discussion

Having analyzed the results, it appears that DNNs has significant disadvantages compared to shallow ML models in the explored scenarios. However, some critical factors merit further discussion. This section addresses these factors to enable the reader to fully understand the extent to which DNNs should be applied.

5.1. Results representativeness of 5G tasks

Although the results regarding the proposed tasks seem to present shallow ML models as the best approach, and that the use of DNNs is unnecessary, there are some limitations regarding the generability of these results to every possible task in B5G or 6G.

The main limitation is that, although the selected tasks are representative of B5G/6G tasks, they are based on tabular and labeled datasets. This means that when considering tasks based on time series, with unlabeled datasets, or even without datasets (as is the case in reinforcement learning), the results might differ, and DNNs might be the only approach to achieve the necessary performance.

Nonetheless, the results obtained from this work are valid for the tasks considered and generalizable to some degree to most tabular labeled datasets in B5G/6G. Furthermore, the primary intent of this work is to highlight the overlooked capacity of shallow models to effectively solve many tasks in B5G/6G and the additional cost of using DNNs unnecessarily. Meaning that, even with a limited generalization, the results successfully highlight the overlooked issue in current research, where DNNs are seen as a go-to solution and their cheaper alternatives are disregarded. Leading to increased system costs and complexity, and a reduced level of understanding regarding its internal operation.

5.2. XAI metrics (dis)agreement

Another topic that merits further discussion is the correlation results between PI and PDV, as well as their implications in a real-world deployment. As stated before, the correlation results, in practice, serve as a metric indicating how similar the two XAI metrics' outputs are to one another. Meaning that, if the two XAI metrics provide a similar proportional feature importance value for every feature, then the correlation is high.

Since the results show that even for models with high performance (>0.8 MCC), there can be low correlations, it means that the explanations provided by the two metrics differ considerably. In addition, since there is no way to validate which of the two metrics provided the correct explanation, it means that the end-user, who would rely on these explanations to justify the model's behavior, does not know how the model behaves.

In summary, the disagreement between explanation metrics highlights the black-box nature of DNNs, the current lack of a solution to this issue, and its continued status as a significant disadvantage of such models. Making it even harder to justify the use of DNNs when a simple and inherently explainable model like a DT achieves the same performance.

5.3. Which model to use

Although ideally, there would be a clear winner among all the tested models, the reality is that there is no "one-size-fits-all" in ML. Some models consistently stand out, like the RF. However, even when using RF, there were datasets where other models performed better in terms of MCC or training and inference times.

These results highlight that before applying any model, the options should be carefully analyzed. Furthermore, it highlights the importance of hyperparameter tuning for shallow ML models, as most did not retain the default values used by Scikit-learn in the tuned hyperparameters.

Nonetheless, upon examining the results, it appears that there is no need to use DNNs. And that assessment is correct for the datasets considered. Furthermore, there is evidence that for most tabular datasets, there is no need to apply DNNs [6].

However, let us consider NLP, computer vision, or time series data. There is clear evidence that DNNs are state-of-the-art models and that traditional ML approaches cannot reach the desired performance. In these situations, incurring the extra training and inference costs, as well as reduced explainability, is the only way to achieve the desired accuracy.

That said, DNNs are aimed towards problems that simpler models cannot solve, as the extra costs incurred and the lack of explainability by applying them cannot be ignored.

Another essential factor that attentive readers might have noticed is that the testbed did not have a GPU. Although a GPU would not change the accuracy results, it could help to improve the training and inference times of the DNNs. However, this is not what is expected to happen. For those interested, we refer to our previous work [7], where we used and tested some of the same datasets and DNNs.

In the previous work, the DNNs models tested were trained on an RTX 2080, and in three out of the four datasets, the training was slower than when using the Kunpeng server alone. Since the Kunpeng CPU differs from the one used with the RTX 2080, the results cannot be compared directly. However, there is strong evidence that because of the DNNs' reduced size and the small amount of dataset samples, the overhead of transferring it to the GPU made the training procedure considerably slower in three out of four datasets.

Furthermore, with increasing concerns about energy efficiency and reduced latency, enabling ML models to run directly on devices, such as IoT devices or network devices, is crucial. So, experiments should ideally not require a GPU to provide training and inference times within the expected network requirements.

6. Conclusion

In conclusion, this paper compares the performance, training time, inference time, energy costs, and explainability of DNNs with shallow ML models on various datasets of essential tasks for 5G/B5G networks. The DNNs used were proposed in the state-of-the-art, and the shallow models were built based on the best-performing hyperparameters of a GridSearch, ensuring a fair comparison between them.

The results show that the most shallow ML model performed similarly to the DNNs at a fraction of the training and inference time, and the overall best-performing shallow model was the RF as it maintained consistent results in every dataset tested.

Furthermore, when assessing explainability, the two metrics considered for feature importance of DNNs did not yield consistent results, highlighting the difficulty in explaining such complex models. This is another advantage of shallow models, as they can easily comply with the EU AI Act regulations. At the same time, the DNNs would require further study to be applicable, as ML models need to be explained to be applicable to decisions that affect individuals.

Nonetheless, DNNs should not be disregarded entirely, as the datasets considered are tabular and do not include some data types where DNNs are crucial, such as NLP, computer vision, or time series. In those scenarios, the value of applying DNNs is evident as the accuracy achieved outweighs the remaining costs. That said, this paper demonstrates that applying DNNs first is a poor practice, as their costs cannot be disregarded if a simple, shallow ML model can solve the same task.

In future work, we would like to expand this analysis by considering an even broader set of 5G/B5G tasks, an increased number of datasets, and expanding the GridSearch performed on the shallow ML models.

CRediT authorship contribution statement

Rafael Teixeira: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Data curation, Conceptualization. **Leonardo Almeida:** Writing – original draft, Visualization, Software, Data curation. **Pedro Rodrigues:** Writing – original draft, Visualization, Validation, Software, Conceptualization. **Mário Antunes:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Diogo Gomes:** Writing – review & editing, Visualization, Validation, Supervision, Formal analysis, Conceptualization. **Rui L. Aguiar:** Writing – review & editing, Validation, Supervision, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was funded by the PRR – Plano de Recuperação e Resiliência and by the NextGenerationEU funds at University of Aveiro, Portugal, through the scope of the Agenda for Business Innovation "NEXUS: Pacto de Inovação – Transição Verde e Digital para Transportes, Logística e Mobilidade" (Project n° 53 with the application C645112083-00000059).

Appendix A. Hyperparameters used in the models' optimization

See Table A.11.

Table A.11
Hyperparameter values considered in hyperparameter optimization.

Hyperparameter	Values
k-NN - 192 combinations	
Weights	{ *uniform*, *distance* }
N° Neighbors	{ 3, 5, 7, 9 }
Algorithm	{ *auto*, *ball_tree*, *kd_tree*, *brute* }
Leaf Size	{ 1, 10, 30, 60, 90, 120 }
GaussianNB - 100 combinations	
Var. Smoothing	$logspace(0, -9, num = 100)$
DT - 24 combinations	
Criterion	{ *gini*, *entropy* }
Max depth	{ 3, 5, 7, 9 }
Max features	{ *auto*, *sqrt*, log_2 }
RF & Gradient Boosting - 48 combinations	
N° Estimators	{ 5, 10, 50, 100 }
Max depth	{ 3, 5, 7, 9 }
Max features	{ *auto*, *sqrt*, log_2 }
AdaBoost - 4 combinations	
N° Estimators	{ 5, 10, 50, 100 }

Table B.12
Model selection results for QoS_QoE. The dataset name in bold highlights the best model. Values for accuracy metrics are obtained with the validation set, and the training time is presented in seconds.

Performance metric	Models						
	IoT_DNL	KPI-KQI	NetSlice5G	RT_IoT	Slicing5G	UNAC	UNSW
MCC	99.35	99.70	99.75	99.38	99.63	99.68	99.49
F1-Score	99.75	99.88	99.90	99.77	99.86	99.88	99.81
ROC-AUC	99.28	99.52	99.28	99.79	99.38	99.64	99.58
Training time	40.62	59.33	69.49	20.26	58.65	61.74	28.18

Table B.13
Model selection results for IoTID20. The dataset name in bold highlights the best model. Values for accuracy metrics are obtained with the validation set, and the training time is presented in seconds.

Performance metric	Models						
	IoT_DNL	KPI-KQI	NetSlice5G	RT_IoT	Slicing5G	UNAC	UNSW
MCC	70.50	65.07	63.90	68.81	61.50	63.57	70.64
F1-Score	70.52	68.57	66.16	72.34	62.12	65.91	73.88
ROC-AUC	80.98	78.79	77.29	81.42	75.65	77.34	82.59
Training time	164.54	116.56	135.32	84.91	106.51	104.25	103.54

Table B.14
Model selection results for B_BaIoT. The dataset name in bold highlights the best model. Values for accuracy metrics are obtained with the validation set, and the training time is presented in seconds.

Performance metric	Models						
	IoT_DNL	KPI-KQI	NetSlice5G	RT_IoT	Slicing5G	UNAC	UNSW
MCC	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F1-Score	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ROC-AUC	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Training time	39.10	32.63	72.30	21.55	66.70	35.63	24.46

Table C.15
Accuracy results of the various models for slice attribution datasets using F1-Score as metric.

Model	Slicing5G	NetSlice5G	KPI-KQI	QoS-QoE	UNAC
DNN	1.00	1.00	0.94	0.97	0.95
k-NN	1.00	1.00	0.86	0.91	0.93
GaussianNB	0.92	1.00	1.00	0.69	0.91
DT	1.00	1.00	0.97	0.85	0.88
RF	1.00	1.00	1.00	1.00	0.95
AdaBoost	0.92	1.00	0.82	1.00	0.95
Grad. Boost.	1.00	1.00	1.00	1.00	0.95

Table C.16
Accuracy results of the various models for slice attribution datasets using ROC-AUC as metric.

Model	Slicing5G	NetSlice5G	KPI-KQI	QoS-QoE	UNAC
DNN	1.00	1.00	0.97	0.98	0.94
k-NN	1.00	1.00	0.92	0.91	0.93
GaussianNB	0.95	1.00	1.00	0.93	0.96
DT	1.00	1.00	0.98	0.90	0.91
RF	1.00	1.00	1.00	1.00	0.96
AdaBoost	0.95	1.00	0.92	1.00	0.96
Grad. Boost.	1.00	1.00	1.00	1.00	0.96

Table C.17
Accuracy results of the various models for IDS datasets using F1-Score as metric.

Model	UNSW	IoT-DNL	B_BaIoT	IoTID20	RT-IOT
DNN	0.45	1.00	1.00	0.66	0.94
k-NN	0.53	1.00	1.00	0.77	0.97
GaussianNB	0.19	1.00	0.98	0.41	0.62
DT	0.41	1.00	1.00	0.60	0.96
RF	0.43	1.00	1.00	0.72	0.97
AdaBoost	0.37	1.00	1.00	0.50	0.49
Grad. Boost.	0.47	1.00	1.00	0.77	0.97

Table C.18
Accuracy results of the various models for IDS datasets using ROC-AUC as metric.

Model	UNSW	IoT-DNL	B_BaIoT	IoTID20	RT-IOT
DNN	0.72	1.00	1.00	0.83	0.96
k-NN	0.77	1.00	1.00	0.87	0.99
GaussianNB	0.62	1.00	0.97	0.70	0.89
DT	0.70	1.00	1.00	0.79	0.97
RF	0.71	1.00	1.00	0.84	0.98
AdaBoost	0.70	1.00	1.00	0.73	0.75
Grad. Boost.	0.76	1.00	1.00	0.87	0.98

Appendix B. Model selection results

See Tables B.12–B.14.

Appendix C. Model performance using additional metrics

See Tables C.15–C.18.

Data availability

All experimental data is available at the paper's github: https://github.com/rgtzths/shallow_vs_deep_learning_extended.

References

[1] A. Thantharate, R. Paropkari, V. Walunj, C. Beard, Deepslice: A deep learning approach towards an efficient and reliable network slicing in 5 g networks, in: 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference, UEMCON, 2019, pp. 0762–0767.

[2] M. Agiwal, A. Roy, N. Saxena, Next generation 5 g wireless networks: A comprehensive survey, IEEE Commun. Surv. Tutor. 18 (3) (2016) 1617–1655.
[3] C. Zhang, P. Patras, H. Haddadi, Deep learning in mobile and wireless networking: A survey, IEEE Commun. Surv. Tutor. 21 (3) (2019) 2224–2287.
[4] G. Flagship, 6 g white paper on edge inteligence, 2021, https://5g-ppp.eu/wp-content/uploads/2021/05/AI-MLforNetworks-v1-0.pdf, (Accessed 15 September 2023).
[5] X. Hexa, Deliverable d5.1 initial 6 g architectural components and enablers, 2022, https://hexa-x.eu/wp-content/uploads/2022/03/Hexa-X_D5.1_full_version_v1.1.pdf, (Accessed 15 September 2023).
[6] L. Grinsztajn, E. Oyallon, G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data? in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems, Vol. 35, Curran Associates, Inc., 2022, pp. 507–520.
[7] R. Teixeira, L. Almeida, P. Rodrigues, M. Antunes, D. Gomes, R. Aguiar, Shallow vs. deep learning: Prioritizing efficiency in next generation networks, 2024.
[8] N. Alliance, Ngmn 5 g white paper, 2015, https://ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf (Accessed 15 September 2023).
[9] X. Zhu, J. Wang, Q. Lai, X. Luo, R. Ren, H. Lu, Research on 5 g network slicing type prediction based on random forest and deep neural network, in: 2023 IEEE 8th International Conference on Big Data Analytics, ICBDA, 2023, pp. 154–158.
[10] S. Wijethilaka, M. Liyanage, Survey on network slicing for internet of things realization in 5 g networks, IEEE Commun. Surv. & Tutorials 23 (2) (2021) 957–994.
[11] H. Babbar, S. Rani, A.A. AlZubi, A. Singh, N. Nasser, A. Ali, Role of network slicing in software defined networking for 5 g: Use cases and future directions, IEEE Wirel. Commun. 29 (1) (2022) 112–118.
[12] V. Vasilev, J. Leguay, S. Paris, L. Maggi, M. Debbah, Predicting qoe factors with machine learning, in: 2018 IEEE International Conference on Communications, ICC, 2018, pp. 1–6.
[13] Y. Xu, Y. Zhou, P. Sekula, L. Ding, Machine learning in construction: From shallow to deep learning, Dev. Built Environ. 6 (2021) 100045.
[14] A. Singh, N. Thakur, A. Sharma, A review of supervised machine learning algorithms, in: 2016 3rd International Conference on Computing for Sustainable Global Development, INDIACom, 2016, pp. 1310–1315.
[15] S. Ray, A quick review of machine learning algorithms, in: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing, COMITCon, 2019, pp. 35–39.
[16] X. Dong, Z. Yu, W. Cao, Y. Shi, Q. Ma, A survey on ensemble learning, Front. Comput. Sci. 14 (2020) 241–258.
[17] A. Holzinger, A. Saranti, C. Molnar, P. Biecek, W. Samek, Explainable ai methods-a brief overview, in: International Workshop on Extending Explainable AI beyond Deep Models and Classifiers, Springer, 2020, pp. 13–38.
[18] I. Kök, F.Y. Okay, O. Muyanlı, S. Özdemir, Explainable artificial intelligence (xai) for internet of things: A survey, IEEE Internet Things J. 10 (16) (2023) 14764–14779.
[19] M.T. Ribeiro, S. Singh, C. Guestrin, Why should i trust you? explaining the predictions of any classifier, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135–1144.
[20] S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Advances in Neural Information Processing Systems, Vol. 30, 2017.
[21] J. Huang, Z. Wang, D. Li, Y. Liu, The analysis and development of an xai process on feature contribution explanation, in: 2022 IEEE International Conference on Big Data (Big Data), 2022, pp. 5039–5048.
[22] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.
[23] B.M. Greenwell, B.C. Boehmke, A.J. McCarthy, A simple and effective model-based variable importance measure, 2018.
[24] J.H. Friedman, Greedy function approximation: A gradient boosting machine, Ann. Statist. 29 (5) (2001) 1189–1232.
[25] D. Chowdhury, R. Das, R. Rana, A.D. Dwivedi, P. Chatterjee, R.R. Mukkamala, Autodeepslice: A data driven network slicing technique of 5 g network using automatic deep learning, in: 2022 IEEE Globecom Workshops, GC Wkshps, 2022, pp. 450–454.
[26] J.E. Preciado-Velasco, J.D. Gonzalez-Franco, C.E. Anias-Calderon, J.I. Nieto-Hipolito, R. Rivera-Rodriguez, 5G/b5g service classification using supervised learning, Appl. Sci. 11 (11) (2021).
[27] C. Sandeepa, T. Senevirathna, B. Siniarski, M.-D. Nguyen, V.-H. La, S. Wang, M. Liyanage, From opacity to clarity: Leveraging xai for robust network traffic classification, in: U. Jayasinghe V. Narayanan R. Ragel D. Herath, J. Wang (Eds.), Asia Pacific Advanced Network, Springer Nature Switzerland, Cham, 2024, pp. 125–138.
[28] I. Parsa, KDD Cup 1998 Data, UCI Machine Learning Repository, 1998, http://dx.doi.org/10.24432/C5401H.
[29] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, P. Chan, KDD Cup 1999 Data, UCI Machine Learning Repository, 1999, https://doi.org/10.24432/C51C7N.
[30] M. Tavallaee, E. Bagheri, W. Lu, A.A. Ghorbani, A detailed analysis of the kdd cup 99 data set, in: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 1–6.
[31] K. Kostas, M. Just, M.A. Lones, Iotgem: Generalizable models for behaviour-based iot attack detection, 2023, arXiv preprint arXiv:2401.01343.
[32] N. Moustafa, J. Slay, Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set), in: 2015 Military Communications and Information Systems Conference, MilCIS, 2015, pp. 1–6.
[33] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, Y. Elovici, N-baiot—network-based detection of iot botnet attacks using deep autoencoders, IEEE Pervasive Comput. 17 (3) (2018) 12–22.
[34] I. Ullah, Q.H. Mahmoud, A scheme for generating a dataset for anomalous activity detection in iot networks, in: C. Goutte, X. Zhu (Eds.), Advances in Artificial Intelligence, Springer International Publishing, Cham, 2020, pp. 508–520.
[35] B.S. Sharmila, R. Nagapadma, Quantized autoencoder (qae) intrusion detection system for anomaly detection in resource-constrained iot devices using rt-iot2022 dataset, Cybersecurity 6 (2023) 41.
[36] M. Sarhan, S. Layeghy, M. Portmann, Towards a standard feature set for network intrusion detection system datasets, Mob. Netw. Appl. 27 (2021) 357–370.
[37] R. Abou Khamis, A. Matrawy, Evaluation of adversarial training on different types of neural networks in deep learning-based idss, in: 2020 International Symposium on Networks, Computers and Communications, ISNCC, IEEE, 2020, pp. 1–6.
[38] D.K.K. Reddy, J. Nayak, B. Naik, G.S. Pratyusha, 11 deep neural network–based security model for iot device network, Deep. Learn. Internet Things Infrastruct. (2021) 223.
[39] G. Ranjith Kumar, N.S. Govekar, A. Karthik, G. Nijhawan, A.H. Alawadi, A. V, Real-time monitoring and anomaly detection in hospital iot networks using machine learning, in: 2023 International Conference on Artificial Intelligence for Innovations in Healthcare Industries, ICAIIHI, Vol. 1, 2023, pp. 1–8.
[40] J. Corona, M. Antunes, R.L. Aguiar, Eco-friendly intrusion detection: Evaluating energy costs of learning, in: 2023 IEEE 9th World Forum on Internet of Things, WF-IoT, 2023, pp. 1–6.
[41] I. Cohen, Y. Huang, J. Chen, J. Benesty, J. Benesty, J. Chen, Y. Huang, I. Cohen, Pearson correlation coefficient, in: Noise Reduction in Speech Processing, 2009, pp. 1–4.