A Robustness and Computational Efficiency Evaluation Pipeline for Machine Learning-based Traffic Classification in 5G Network Slicing

João V. da Silva Campos[1], Abdel F. Chabi[1], and Andson M. Balieiro[1]

[1] Centro de Informática - Universidade Federal de Pernambuco, Recife, Brasil
[2] {jvsc2, afc9, amb4}@cin.ufpe.br

Abstract. Network slicing represents a core technology to improve heterogeneous Quality of Service (QoS) in 5G and beyond. However the effective network slicing implementation is challenging due to the diverse QoS requirements and dynamic conditions in 5G networks. Machine learning techniques have emerged as a viable solution to mitigate network slicing complexity in 5G. Moreover, the use of machine learning techniques introduces potential vulnerabilities that can lead to severe security risks. In this paper, we propose a structured five-stage pipeline to analyze the trade-offs between predictive performance, computational efficiency, and adversarial robustness in MLP-based 5G traffic classification. Feature selection using Mutual Information and PCA reduced the original feature set by more than 50% while preserving accuracy. Hyperparameter optimization further decreased CPU usage and training time with minimal performance loss. However, adversarial evaluation under FGSM and PGD attacks revealed a significant degradation in performance as the intensity of the perturbation increased. The adversarial training strategies improved robustness at different levels of perturbation, albeit with increased computational cost.

Keywords: Network Slice · MLP · Adversarial Attacks.

1 Introduction

The Fifth Generation of Mobile Networks (5G) has been deployed, promoting high bandwidth, low latency, and support for a high density of connected devices. These capabilities foster a wide range of applications, including autonomous vehicles, augmented reality, and the Internet of Things (IoT), which are categorized into three service classes, Ultra-Reliable and Low-Latency Communications (URLLC), enhanced Mobile Broadband (eMBB), and massive Machine-Type Communications (mMTC). To address this growing service diversity, network slicing (NS) has emerged as an key enabler, allowing the creation of specialized end-to-end logical networks (slices) over a shared physical infrastructure [1]. This approach extends network capabilities to a broad range of verticals with diverse requirements in terms of latency, throughput, density connection, and reliability [2].

However, managing multiple slices and efficiently assign users and resources across these slices pose significant challenges, where traditional techniques often fail to handle the dynamic and diverse needs of 5G users/traffics, In this respect, Artificial Intelligence (AI) and Machine Learning (ML) techniques have been widely explored [3] to enhance the network performance and intelligently classify network slices in real-time, where neural network-based solutions have shown notable capability in modeling complex patterns and optimizing decisions regarding network slice management.

For instance, [4] uses traditional and incremental learning combined with Convolutional Neural Networks (CNN) to select the most suitable slice for different users and device types. The authors in [5] adopt ML models, including Random Forest, XGboost, Gradient Boosting Decision Tree (GBDT), and K Nearest Neighbor (KNN), for traffic classification, enabling dynamically configuration and optimization of slice resources based on user traffic demands. Similarly, [6] proposes a Federated Learning (FL) framework integrated with CNN for service assignment in slicing-based 5G networks. By predicting the load on each network slice, the solution allocates the incoming traffic to the most suitable slice, demonstrating performance gains over centralized CNN-based approaches. Although these proposals present valuable insights and promising results, they evaluate the solutions and network performance without exploring potential vulnerabilities and anomalies that may potentially affect their behaviors, thereby limiting their robustness. In contrast, the authors in [7] explore security issues in 5G network slicing. Their discussion is centered on 5G network slicing, investigating attack vectors and mitigation approaches across 5G layers such as orchestration, virtualization, and inter-slice communication. Their findings highlight the need for layered defenses, AI-driven monitoring, and architectural isolation as critical components to enhance the resilience of 5G slicing deployments. However, the authors do not employ AI/ML solutions, and, consequently, overlook adversarial attacks against their structures.

Adversarial attacks pose significant challenges to AI/ML-based approaches, as they introduce carefully crafted perturbations into input data (e.g., adversarial traffic samples) with the objective of manipulating model predictions. These perturbations can lead the system to produce highly confident yet incorrect classifications, thereby compromising reliability. Consequently, designing or evaluating traffic classifiers solely in adversarial attack-free environments may conceal robustness limitations that would emerge under real-world conditions. Therefore, a comprehensive investigation of AI/ML-based traffic classification solutions for slicing-based 5G networks is required, considering not only classification accuracy and resource efficiency but also robustness against adversarial threats.

In this context, this work proposes a systematic pipeline that encompasses multiple stages of the machine learning development process, including data preprocessing, feature selection strategies, hyperparameter optimization, adversarial attack and training. The proposed methodology evaluates robustness, training and inference time, and computational resource consumption (CPU and mem-

ory) under various adversarial attack types and intensities, as well as different defense mechanisms. We apply the methodology to a Multi-Layer Perceptron (MLP)-based traffic classifier, as this model type is widely adopted for classification tasks and captures of nonlinear relationships between traffic metrics and quality-of-service requirements, contributing to more efficient and adaptive network slice management [8]. Results revealed that the model is able to maintain strong accuracy and F1-score metrics with a reduction of more than 50% of the original attributes, furthermore, the hyperparameter optimization stage reduced the CPU mean usage by 42% during training time, with a loss of less than 1% in accuracy. Under Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) attacks, performance degraded significantly, but the adversarial defenses were able to maintain accuracy above 0.8 on realistic perturbation values, at the cost of a significant increase in training time.

This paper is structured as follows. Section 2 presents the proposed methodology and explains the pipeline structure. Section 3 analyzes the results, and finally, Section 4 concludes the paper and outlines directions for future research.

cial Intelligence (AI) techniques have been widely explored [3]. Among these techniques, neural network-based models have demonstrated significant capability in modeling complex patterns and optimizing decisions related to network slice management and the use of MLP enables the capture of nonlinear relationships between traffic metrics and quality-of-service requirements, contributing to more efficient and adaptive management. [8]

2 Proposed Methodology

To ensure a structured and reproducible evaluation, the proposed framework is organized into five sequential stages, covering data preprocessing, feature selection, model training, and robustness assessment. Different feature selection strategies, model configurations, and perturbation intensities are investigated to analyze their impact on predictive performance and computational cost.

The optimized model is subsequently evaluated under adversarial attacks targeting traffic classification into network slices. Furthermore, adversarial training strategies are incorporated to increase robustness by integrating dynamically generated perturbed samples into the model training. Fig. 1 presents the structural diagram of this pipeline.

2.1 Stage 1 - Data Cleaning

The dataset consists of ARFF and CSV files representing global 5G traffic and specific subsets of slices (eMBB, mMTC, and URLLC), containing traffic statistics and information related to attacks [9]. Data preprocessing was performed using Pandas [10] and included the removal of non-informative attributes, imputation of missing values, and categorical coding. Columns with an excess of missing values or index-like behavior were discarded; some of these columns had

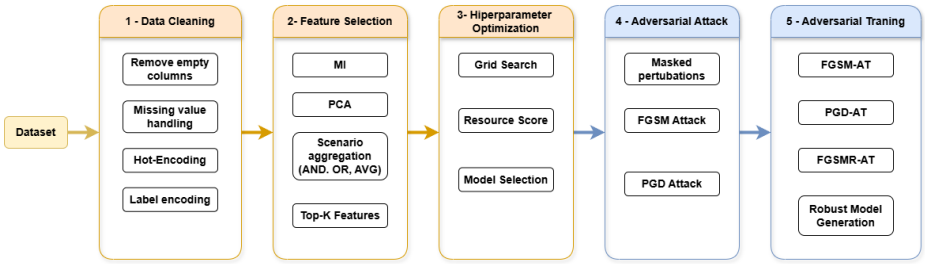

Fig. 1. Proposed methodological pipeline illustrating the five main stages.

approximately 95% missing values. Missing values were treated according to feature semantics: specific TCP attributes were filled with -1 to represent non-TCP flows, and the remaining numerical features were imputed using the median. Categorical variables were converted to numeric format via one-hot coding, and the target's binary label was coded as 0 (Benign) and 1 (Malicious). After preprocessing, no missing values remained in the dataset.

2.2 Stage 2 - Feature Selection

The second stage aimed to reduce dimensionality while preserving discriminative power and minimizing computational cost. Two feature-ranking strategies were independently computed, Mutual Information (MI) and Principal Component Analysis (PCA). The former is a measure of dependence between two variables. It computes the mutual information between each feature and the target class (label). The main idea is to estimate an MI value where, the higher the value, the more related the feature is to the label. This estimation is performed using non-parametric methods, which are based on nearest-neighbor distances, without assuming linear models or data normalization. The function used to perform this calculation was *mutual_info_classif* from the sklearn library [11], which follows Eq. 1, where $H(y)$ represents the entropy of target variable and $H(Y \mid X)$ denotes the conditional entropy given feature X.

$$I(X;Y) = H(Y) - H(Y \mid X) \tag{1}$$

The PCA is a technique for linear dimensionality reduction that projects data into a new lower-dimensional space while capturing the maximum possible variance. This technique first centers the data by subtracting the mean of each feature. It then computes the Singular Value Decomposition (SVD), or alternatively the eigenvalues and eigenvectors of the data covariance matrix. From this point, the components are ordered according to the amount of explained variance, and the original data are projected into a new space defined by the principal components. [12]. Eqs. 2, 3, and 4 describe the PCA computation, which is also obtained by using the sklearn library *decomposition.PCA*. Eq. 2

defines the covariance matrix of a centered data matrix $\tilde{X}$. The principal components are obtained by solving the eigenvalue problem defined in Eq. 3, and the projection onto the first k principal components is given by Eq. 4.

$$S = \frac{1}{n-1}\tilde{X}^T\tilde{X} \tag{2}$$

$$Sv_i = \lambda_i v_i \tag{3}$$

$$Z = \tilde{X}V_k \tag{4}$$

After computing the MI and PCA values for each feature, threshold values were defined to eliminate features with low scores. The thresholds were set to MI = 0.1 and PCA = 0.002. Features with scores below these thresholds were discarded. These threshold values were determined after observing a significant drop in the scores beyond these points. Only features simultaneously selected by both MI and PCA were retained. This strategy prioritizes robustness and consensus between statistical relevance and contribution to variance. A feature selected exclusively by MI may have high relevance to the target, but may present redundancy with other features or represent high-variance noise. A feature selected exclusively PCA may explain the variability of the data well, but have little relation to what is desired to be predicted. A feature that passes both filters simultaneously demonstrates that it is relevant to the target MI and structurally representative of the data PCA — reducing the probability of including redundant variables.

The models were trained incrementally using increasing values of K (number of most relevant features), starting with the 5 most relevant obtained in the previous step and incrementing by another 5, and so on. The goal was to determine the point at which increasing dimensionality no longer produces significant performance gains.

2.3 Stage 3 - Hyperparameter Optimization

Once the optimal feature scenario and corresponding value of K were determined, hyperparameter optimization was performed the primary objective of this stage was not necessarily to maximize predictive performance metrics such as accuracy, F1-score, or AUC (Area Under the Curve), but rather to identify hyperparameter configurations that optimize the model in terms of computational resource consumption.

A manual grid search was conducted, exploring combinations of the following hyperparameters: number of hidden units, dropout rate, learning rate, and batch size. To evaluate the configurations at this stage, a resource score was defined as the mean of the normalized values of three computational metrics: average CPU usage, average RAM consumption, and training time. The final model selection followed a two-step criterion: the configurations were filtered based on predictive performance, and among the best-performing candidates, the configuration with

the lowest feature score was selected, ensuring a balance between predictive effectiveness and computational efficiency.

2.4 Stage 4 - Adversarial Attacks

After model optimization, adversarial samples were generated using the K main features selected in the feature selection step. Two adversarial attacks were considered, the FGSM and PGD. The FGSM is a single step attack designed for computational efficiency. It generates an adversarial example by computing the gradient of the loss function with respect to the input and adding a perturbation proportional to the sign of that gradient, moving the sample in the direction that maximizes the model's error [13]. The formula used to generate the perturbed image x' from the original input x is, where ϵ is a parameter that controls the magnitude of the perturbation, ensuring that the modification remains subtle. Although computationally efficient, FGSM can be mitigated relatively easily through basic adversarial training [13].

$$x' = x + \epsilon \cdot \text{sign}\left(\nabla_x L(\theta, x, y)\right) \tag{5}$$

To overcome the limitations of FGSM, PGD was proposed as a more powerful iterative variant. PGD applies the FGSM principle multiple times using smaller step sizes (determined by a factor α), projecting the result back into a permissible neighborhood around the original sample after each iteration. This iterative process enables a more thorough exploration of the input space, making PGD one of the most effective first-order attacks and a standard benchmark for evaluating model robustness. [14]

All perturbations were applied to the scaled feature space to ensure numerical stability during gradient calculation. The bounds of each feature were calculated from the training set and stored as minimum and maximum per feature to ensure semantic consistency. A feature-level attack domain was loaded from the pipeline cache file. This domain contains: a binary mask indicating which features can be perturbed, minimum and maximum bounds of the raw space derived from the training data, and index lists identifying integer and binary features. During attack generation, the gradient-based perturbation was first calculated on the scaled space and then multiplied by the attack mask, ensuring that only valid dimensions were modified. Features marked as non-perturbable remained unchanged.

After perturbation, adversarial samples were clipped to the scaled training boundaries and subsequently projected back onto the raw feature space using the inverse transformation of the tuned *StandardScaler*. The samples were then rescaled before being used for inference. This projection procedure ensures that the generated adversarial traffic remains realistic and consistent with the domain constraints, rather than producing mathematically valid but semantically impossible samples.

Adversarial perturbations were generated directly through TensorFlow's automatic differentiation mechanism *tf.GradientTape*, enabling explicit computa-

tion of gradients with respect to the input features. No external adversarial machine learning libraries were used; instead, all attack routines were implemented manually. Furthermore, attacks were applied to the model at different intensity levels (ϵ values = [0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3]) to verify the robustness of a model without adversarial training.

2.5 Stage 5 - Adversarial Training

Adversarial Training was implemented directly inside the mini-batch optimization loop rather than as a pre-generated dataset augmentation step. During each epoch, training samples were shuffled and processed in batches. For every clean batch, adversarial counterparts were generated on-the-fly using the same attack configuration and feature-level constraints described previously.

Importantly, the adversarial samples were not permanently stored in memory nor added to the dataset. Instead, they were dynamically computed for the current batch, designed for the realistic domain, and used immediately in the gradient update step, where a 50% ratio of adversarial samples to the total number of samples in training was applied. All adversarial training variants (FGSM-AT, FGSMR-AT, PGD-AT, and META-AT) share this same structural implementation; they differ only in how adversarial perturbations are generated within the batch.

3 Discussion and Results

The proposed pipeline was developed in a Jupyter Notebook environment using Mini Anaconda, running on a local machine. The machine has a Ryzen 5500X3D processor with 6 cores and 12 threads, and two 8GB RAM modules running at 3200Hz, totaling 16GB (DDR 4). All computational steps were executed in isolated processes to ensure a fair measurement of computational resources, which is monitored using the psutil library [15]. This library logs CPU utilization, peak memory usage (RSS), and actual execution time. Monitoring is performed via a dedicated background thread that samples resource usage at fixed intervals. The percentage values in CPU usage are related to the processor's cores. The processor used has 12 threads, thus having 12 logical cores from the OS point of view, therefore, a result of 210% CPU usage means 2 cores are completely used and an additional 10% of a third core.

The data preprocessing stage reduced the dimensionality of the dataset from X to Y features by removing empty columns and properly handling missing values. After this procedure, no missing entries remained in the dataset. Additionally, two features were discarded due to their limited usability, as each contained approximately 95% missing values. The final dataset comprised 14456 samples and 74 features. Following the feature selection procedure described in Step 2 of the methodology, the graphs in Fig. 2 were obtained, showing the accuracy achieved along with the average RAM consumption during the evaluation for different subsets of top-K features.

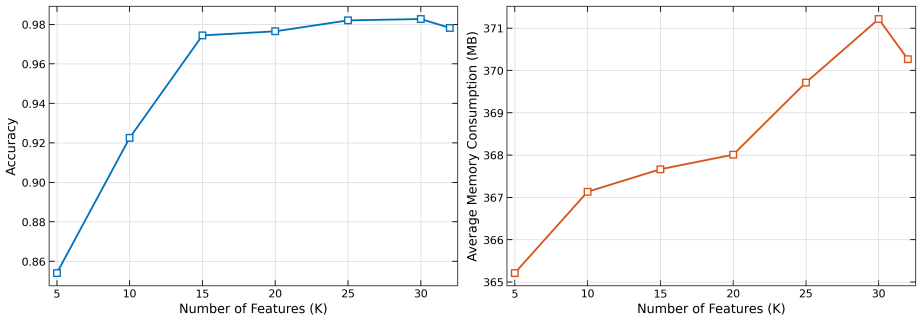

Fig. 2. Accuracy (left) and average RAM usage (right) as a function of the number of selected features (K).

This scenario produced a subset totaling 32 features, since it only includes variables that simultaneously satisfy the MI and PCA selection thresholds. It can be observed in Fig. 2 that, even with a relatively small number of features, the model is already able to effectively separate the classes. Furthermore, the average memory usage increases as more features are incorporated at each training step.

To proceed with the pipeline, the value of K = 30 was selected, corresponding to the 30 main characteristics of the dataset. This configuration achieved the best performance in the operations research scenario, with an accuracy of 0.9938, an F1 score of 0.9939, and an AUC of 0.9993. The average CPU usage during training was 244.55%, with an average RAM consumption of 370.54 MB. During inference, the model required 109.93% of the CPU and 375.28 MB of RAM.

In the grid search stage, the primary objective was not to improve classification metrics such as accuracy, F1-score, or ROC curve performance, but rather to optimize the model so that it preserves the previously achieved predictive performance while reducing computational resource consumption. To this end, configurations with fewer hidden units, smaller batch sizes, and alternative hyperparameter settings were evaluated. Table 1 summarizes the results obtained in this stage. During training, the optimized model achieved approximately a 40% reduction in average CPU usage and a 33% decrease in training time. In the validation phase, a reduction of about 22% in average CPU usage was observed, while accuracy decreased marginally by 0.31%. These findings highlight that CPU utilization is one of the most sensitive metrics in this context and can be effectively optimized to reduce overall computational cost.

During the feature implementation stage, an attack mask was employed to define which features could not be perturbed and, for the set of perturbable features, constrain the allowable minimum and maximum values within their respective domains. This mechanism ensured that adversarial manipulations remained consistent with the data distribution and respected feature-specific bounds. A sample of the values obtained during the mask construction, as its structural organization, is presented in Table 2. Table 3 presents the hyperparameter adopted in the baseline feature selection model and those identified through the grid search optimization process.

Table 1. Training and validation results for baseline and optimized models.

Training Phase			
Metric	Baseline	Optimized	Improvement (%)
CPU mean (%)	227.31	130.84	42.44
RAM mean (MB)	375.49	366.87	2.29
RAM peak (MB)	379.85	372.46	1.94
Training time (s)	5.1410	3.4333	33.22
Validation Phase			
Metric	Baseline	Optimized	Improvement (%)
Accuracy	0.9938	0.9907	-0.31
CPU mean (%)	116.01	90.41	22.07
RAM mean (MB)	380.58	373.24	1.93
RAM peak (MB)	381.16	373.89	1.91
Inference time (s)	0.1753	0.1721	1.85

Table 2. Sample of attack mask configuration for selected features.

feature	perturbable	Type	min_value	max_value
DstTCPBase	True	CONTINUOUS	-1.0	4.291e+09
sTtl	True	INTEGER_COUNT	36.0	2.550e+02
SrcBytes	True	CONTINUOUS	0.0	5.170e+05
State_REQ	False	BINARY	0.0	1.0
predicted_1:eMBB	False	BINARY	0.0	1.0

Table 3. Baseline and optimized model hyperparameter.

Hyperparameter	Baseline	Optimized
Hidden Units	256	32
Dropout	0.3	0.2
Learning Rate	0.001	0.0005
Batch Size	256	256
Epochs	50	50

After using the mask in the attack generation process, Fig. 3 illustrates the model's accuracy under different magnitudes of perturbation (ϵ). As expected, both FGSM and PGD progressively degrade model performance as episilon increases, indicating lower robustness under stronger adversarial perturbations. PGD produces a more pronounced degradation compared to FGSM at most perturbation levels. This behavior can be attributed to the iterative nature of PGD, while FGSM performs a single gradient-based update step to generate adversarial examples, PGD applies multiple smaller, successive perturbation steps, projecting the perturbed samples back into the admissible ϵ after each iteration.

The implementation of adversarial defenses also relied on the same attack mask defined in the previous stage. Adversarial training was performed by injecting attack-generated samples into the training process so that the model could learn patterns associated with malicious traffic. Fig. 4 presents the accuracy results of the adversarially trained models under each implemented attack. The (FGSM attack) and (PGD attack) curves represent the baseline accuracy under FGSM and PGD attacks, respectively, obtained in the previous stage. Although the defense techniques also exhibit performance degradation as the perturbation magnitude increases in both attack scenarios, they maintain substantial robustness up to $\epsilon = 0.15$. Even under more aggressive attack settings adversarial training strategies preserve accuracy levels close to 0.7 and above 0.6, whereas the model without defenses drops below 0.5 under the same conditions.

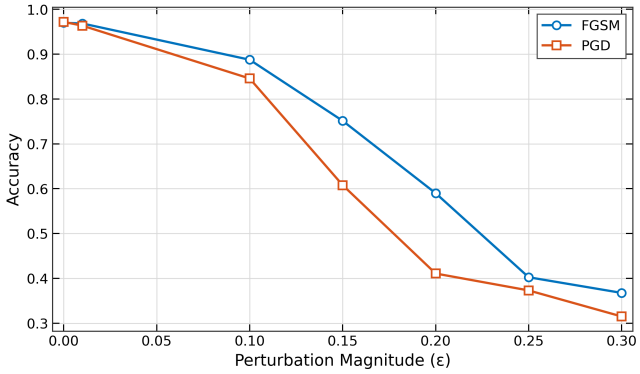

Fig. 3. Accuracy under FGSM and PGD attacks varying the perturbation magnitude (ϵ).

Table 4 provides a computational cost analysis of model training with and without adversarial defenses, highlighting an important trade-off associated with the robustness-enhancing strategies.The adversarial training introduces a computational overhead, primarily reflected in increased training time, especially for iterative methods such as PGD.

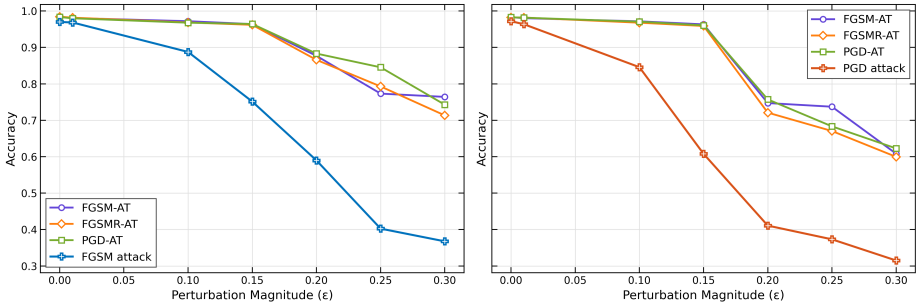

Fig. 4. Accuracy under FGSM (left) and PGD (right) attacks varying the perturbation magnitudes (ϵ) for baseline and adversarially trained models.

Although the training duration grows, memory consumption remains relatively stable across the evaluated configurations, and the average CPU utilization does not increase proportionally. This behavior suggests that the additional cost is mainly driven by repeated gradient computations performed during adversarial sample generation.

Table 4. Computational overhead of adversarial training methods.

Model	Train time (s)	CPU mean (%)	RAM peak (MB)	Time Val. (%)	CPU Val.(%)	RAM Val. (%)
MLP (No AT)	3.33	129.69	371.36	0.00	0.00	0.00
FGSM-AT	17.97	111.14	375.57	439.03	-14.30	1.13
FGSMR-AT	21.53	110.16	482.37	545.89	-15.05	29.89
PGD-AT	71.43	102.92	375.87	2043.27	-20.64	1.22

4 Conclusion

This paper presented a pipeline to analyze the robustness of an MLP-based classification model under adversarial attack scenarios. The feature selection step, combining Mutual Information and PCA through a top-K analysis, reduced more than 50% of the features of the original dataset while preserving predictive performance. Hyperparameter optimization via grid search significantly reduced CPU usage during training and validation, with almost no loss of accuracy. Adversarial evaluation showed a degradation in performance as perturbation levels increased, however adversarial training strategies maintained relatively stable accuracy up to $\epsilon = 1.5$ for both FGSM and PGD attacks. This improvement in robustness comes at the cost of increased computational overhead, particularly, in training time for PGD-AT attack. As future work, other deep leaning archi-

tectures, such as autoencoder, could be evaluated under the same adversarial criteria or under different methods of attacks

References

1. R. F. Olimid and G. Nencioni, "5G Network Slicing: A Security Overview," *IEEE Access*, vol. 8, pp. 99999–100009, 2020. doi:10.1109/ACCESS.2020.2997702
2. C.-C. Tseng, A. Mihovska and S.-Y. Lien (eds.), "The Road to B5G/6G Mobile Communication Networks: Technologies and Applications," River Publishers, 2025.
3. Z.-X. Wu, Y.-Z. You, C.-C. Liu and L.-D. Chou, "Machine Learning Based 5G Network Slicing Management and Classification," in *Proc. 2024 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Osaka, Japan, Feb. 2024. doi:10.1109/ICAIIC60209.2024.10463325
4. M. Ahmadinejad, T. Azmin and N. Shahriar: 5G Network Slice Type Classification using Traditional and Incremental Learning. NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, 2023,1-5, https://doi.org/10.1109/NOMS56928.2023.10154385
5. Z. -X. Wu, Y. -Z. You, C. -C. Liu and L. -D. Chou: Machine Learning Based 5G Network Slicing Management and Classification. 2024 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Osaka, Japan, 2024, pp. 371-375, https://doi.org/10.1109/ICAIIC60209.2024.10463325.
6. Nitul Dutta, Shashikant P. Patole, Rajesh Mahadeva, and Gheorghita Ghinea: Federated learning framework for prediction based load distribution in 5G network slicing. 2024 Sixteenth International Conference on Contemporary Computing (IC3-2024). Association for Computing Machinery, New York, NY, USA, 421–426. https://doi.org/10.1145/3675888.3676085]
7. Dias, J.; Pinto, P.; Santos, R.; Malta, S: 5G Network Slicing: Security Challenges, Attack Vectors, and Mitigation Approaches. Sensors 2025, 25, 3940.https://doi.org/10.3390/s25133940
8. N. A. Mohammedali, T. Kanakis, A. Al-Sherbaz and M. O. Agyeman, "Traffic Classification using Deep Learning Approach for End-to-End Slice Management in 5G/B5G," in *Proc. 2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju Island, Korea, Oct. 2022. doi:10.1109/ICTC55196.2022.9952446
9. Ammar, M. et al.: 5G-SliciNdd dataset. Figshare (2023). https://doi.org/10.6084/m9.figshare.24446515
10. The Pandas Development Team, *Pandas: Python Data Analysis Library*, https://pandas.pydata.org/docs/, accessed 2026/01/06.
11. Scikit-learn Developers: mutual_info_classif — scikit-learn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.html, last accessed 2026/01/27
12. Scikit-learn Developers: PCA — scikit-learn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html, last accessed 2026/01/27
13. Ma, Y., An, D., Gu, Z., Lin, J., Liu, W.: Fast-M adversarial training algorithm for deep neural networks. Applied Sciences **14**(11), 4607 (2024). https://doi.org/10.3390/app14114607
14. Villegas-Ch, W., Jaramillo-Alcázar, A., Luján-Mora, S.: Evaluating the robustness of deep learning models against adversarial attacks: An analysis with

FGSM, PGD and CW. Big Data and Cognitive Computing **8**(1), 8 (2024). https://doi.org/10.3390/bdcc8010008
15. G. Rodola, *psutil: Cross-platform process and system monitoring library*, Python Package Index (PyPI), https://pypi.org/project/psutil/, ast accessed 2026/01/16