Towards TabTransformer-based Operating System Fingerprinting: A Preliminary Approach using the Nmap Database

Rubén Pérez-Jove[1,2*], **Alejandro Pazos**[1,2,3], **Jose Vázquez-Naya**[1,2]

[1]Grupo RNASA-IMEDIR, Departamento de Ciencias de la Computación y Tecnologías de la Información, Facultade de Informática, Universidade da Coruña, Elviña, 15071 A Coruña, Spain
{alejandro.pazos, jose}@udc.es
[2]Centro de Investigación CITIC, Universidade da Coruña, Elviña, 15071 A Coruña, Spain
[3]IKERDATA S.L., ZITEK, University of Basque Country UPVEHU, Rectorate Building, 48940 Leioa, Spain
* *Corresponding author:* ruben.perez.jove@udc.es

***Abstract*—This work explores TabTransformers for operating system fingerprinting, leveraging their ability to learn relationships within data to make classification based on network traffic features. We utilise the Nmap database as the training dataset and evaluate a TabTransformer model on a hold-out test set. The optimised model achieves an outstanding AUC-ROC score of 0.9919, demonstrating its effectiveness in operating system classification. Expanding the number of different classes is one of the intriguing avenues for future work.**

***Index Terms*—Operating System; Fingerprinting; Deep Learning; TabTransformer; Nmap**

Contribution type: *Research work in progress*

I. Introduction

In the field of computer security it is very important to recognise, as accurately as possible, specific characteristics of a machine through the network. Knowing details such as the Operating System (OS) family and version of a machine is crucial for different tasks. Some examples of the defensive applications are network inventory or detection of unauthorised and dangerous devices. There are also offensive applications, such as the determination of the vulnerabilities of a target host, exploit tailoring, or information gathering for subsequent social engineering attacks.

OS fingerprinting is the procedure of identifying the family and version of the OS of a machine connected to the network, analysing its traffic. There are different techniques to achieve this task, but the fundamental one is to analyse the small differences existing in the network packets as a result of the different implementations that each OS makes of the communication protocol stack.

This task can be performed following two different approaches: the active or the passive scan. The active scan is based on the idea of interacting with the target, sending some predefined probes and analysing the responses to determine its nature. The advantages of this approach are the speed of the scan and the reliability of the process, while some negative aspects are the need of interacting with the victim and the possibility of being detected by a defensive system or the target itself. Some of the most known and spread tools in this context are Nmap [1], RING [2] and Xprobe2 [3]. Conversely, the passive scan leverages the task without generating any trace, just listening and analysing the normal traffic produced by the target [4]. This kind of scan is much noiseless than the active one as it does not produce any additional packets in the network, reducing the possibilities of being detected. However, it has the disadvantage of needing more time to get successful results, as well as having poorer effectiveness. Passive OS detection can be performed with p0f [5], Ettercap [6] or NetworkMiner [7] tools.

Traditional rule-based OS fingerprinting approaches struggle to adapt to the evolving landscape of OSs. Newly released systems, frequent updates, and customised configurations can all render signature-based methods ineffective. This necessitates a more robust solution capable of inferring the OS even under these challenging circumstances. Artificial Intelligence (AI) [8], specifically Machine Learning (ML) [9] and Deep Learning (DL) [10], have demonstrably addressed similar challenges in various domains. In the context of OS fingerprinting, research has explored the potential of AI to overcome the limitations of traditional methods [4].

In recent years, there has been a revolution in the field of AI and ML. The Transformer architecture [11] has revolutionised Natural Language Processing (NLP) by introducing multi-head self-attention mechanism for efficient sequence modelling. This enables Transformers to analyse long sequences and generalise across tasks. Large Language Models (LLMs) like GPT4 or Gemini leverage pre-trained Transformers, achieving State-of-the-Art (SoA) performance in text generation, translation, and more. The Transformer's impact extends beyond NLP, with the Visual Transformer (ViT) [12] as an effective example in image processing. In this context, TabTransformer [13] is another version of the Transformer architecture, adapted in this case to tabular data. This adaptability positions the Transformer as a potentially transformative force in network traffic analysis as well.

In this work, we initially explore the application of TabTransformers to the network domain, specifically to the OS fingerprinting task. We leverage the power of the official TabTransformer implementation to analyse the Nmap OS fingerprinting database, a widely recognised tool for network security and monitoring tasks. By applying this model to the OS fingerprinting task in a high-level approach, we aim to

assess its effectiveness in accurately identifying the OS family of a device based solely on its network traffic characteristics. This exploration contributes to the ongoing research on leveraging advancements in ML and DL for network security tasks and, to the best of the authors' knowledge, it represents a pioneering effort in utilising a Transformer-based approach for OS fingerprinting.

The rest of this paper is structured as follows: Section II cites a group of previous works where ML and DL, specifically Transformers, are applied to OS fingerprinting and computer networks in general; Section III describes the Transformer and the specific version for tabular data, the TabTransformer; Section IV explains the DL pipeline followed to solve the OS fingerprinting task using TabTransformers; and finally, results and discussion are presented in Section V.

II. Related Work

The field of OS fingerprinting has traditionally relied on rule-based approaches, but these methods struggle to adapt to the ever-evolving landscape of operating systems. Recent research has explored the potential of ML and DL to address these limitations [4].

Several studies have compared the performance of different ML methods for OS fingerprinting. Song et al. [14] evaluate K-Nearest Neighbors (K-NN), Decision Trees, and Artificial Neural Networks (ANNs), demonstrating that ANNs achieve the highest accuracy (94%) compared to a conventional rule-based method. Similarly, Laštovička et al. [15] compare Naive Bayes, Decision Tree, K-NN, and Support Vector Machine (SVM) on real-world data, analyzing processing time, memory requirements, and classification performance metrics (accuracy, precision, recall).

Traditional fingerprinting techniques are well-suited for static networks and managed environments, however, real-world networks present additional challenges. In [16] authors evaluate three fingerprinting methods applied to university wireless network traffic, demonstrating its viability in the context of evolving network traffic protocols (e.g., IPv6 or HTTP/2.0) This task can be applied to specific environments as well, as in [17], where authors classify diverse IoT devices using a logistic regression model enhanced by supervised learning (LogitBoost). Furthermore, fingerprinting can also be applied at the hardware level to identify physical devices [18] [19].

In this context, our team also has previous experience applying ML to the OS fingerprinting task. In [20] we explore the application of classical ML algorithms for OS fingerprinting using the Nmap signature database, achieving high accuracy (over 96%) with Random Forest. In a subsequent work [21], we apply the previous set of classical ML algorithms to other datasets, and implemented a tool that allows their utilisation in active and passive way.

Although there is no previous work that applies the Transformer to OS fingerprinting field, this architecture is being actively adapted and applied to the computer networks field in recent works. There are studies that leverage textual information for training in this field, with NetBERT [22] outperforming BERT on network tasks. Other works employ BERT for numerical representations of DNS data [23], or combine this model with graph neural networks for feature extraction from packet sequences [24]. Direct training on network traffic data is another approach. In [25], a residual 1-D Image Transformer achieves SoA performance in malware classification and generalises to DDoS attack detection. On the other hand, Flow Transformer [26] outperforms existing methods for anonymity network classification by considering temporal-spatial correlations in traffic data.

While Transformers are being applied in various network security tasks, large or foundational network traffic models, as in the case of NLP with LLMs, are scarce. In this sense, the best effort in this line is netFound [27], a foundational model for network security that leverages unlabelled network traffic for pre-training and achieves superior performance in various downstream tasks compared to existing methods. Addressing this gap, [28] analyses the potential benefits of network traffic foundational models from a theoretical perspective and [29] proposes a Probe-of-Concept (PoC) of this kind of models. However, challenges in generalizability remain, as exemplified by ET-BERT [30], which achieves SoA performance on encrypted traffic classification but lacks generalizability to other tasks.

III. Background

A. Transformer

The Transformer architecture, introduced by Vaswani et al. in their seminal work "Attention is All You Need" [11], has revolutionised various fields within AI. This encoder-decoder architecture processes input sequences and generates corresponding output sequences. Unlike traditional architectures reliant on recurrent or convolutional layers, Transformers leverage fully connected layers and a core mechanism known as multi-head self-attention. This self-attention mechanism empowers the model to analyse the importance of different parts of the input sequence based on their contextual relevance, effectively capturing various relationship aspects simultaneously through "multi-head" processing. Key strengths of the Transformer architecture include:

- Efficient parallelisation: The parallel nature of the architecture allows for faster training compared to sequential models.
- Long sequence processing: Self-attention mechanisms enable the model to efficiently handle long sequences, overcoming limitations observed in traditional Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks.
- Scalability: Transformers demonstrate impressive scalability, allowing them to handle large datasets and complex models, a crucial factor in training SoA models such as Generative Pre-trained Transformers (GPTs).
- Generalizability: Transformers exhibit remarkable generalizability across diverse tasks. Pre-trained models can be fine-tuned for specific applications without extensive modifications, promoting efficient adaptation.

The training process for Transformers typically involves two distinct phases: pre-training and fine-tuning. During pre-training, the model undergoes self-supervised learning on a large corpus of unlabelled data (e.g., textual information in NLP) to learn inherent relationships and semantic nuances within the data domain. The resulting models, often referred

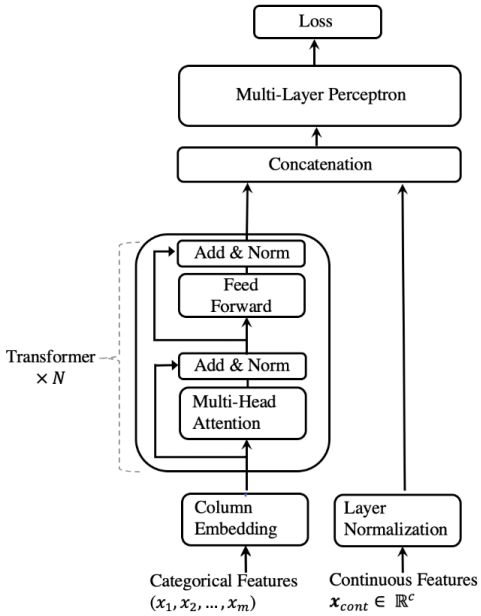

Figure 1. The architecture of TabTransformer. Source: [13]

to as foundational or base models, serve as a powerful starting point for addressing various downstream tasks. In the subsequent fine-tuning phase, these base models are specialised for specific tasks by exposing them to a smaller, labelled dataset relevant to the target application. Research has consistently shown that pre-trained Transformer models outperform those trained through traditional AI pipelines [31].

The success of the Transformer architecture has led to the development of numerous variants tailored for specific modalities and tasks. Prominent examples include the GPT series (e.g., GPT-3, GPT-4), which represent SoA Large Language Models (LLMs) based on a decoder-only Transformer architecture. Another notable variant is BERT [32], which has achieved new benchmarks in various language understanding tasks. Notably, the Transformer has been successfully adapted for diverse modalities beyond text, including images, video, and audio. The Visual Transformer (ViT) [12] exemplifies this versatility, offering a competitive alternative to conventional Convolutional Neural Networks (CNNs) for image classification. This remarkable adaptability has solidified the Transformer architecture as a standard tool in AI research, impacting not only language modelling but also diverse fields like image processing. Considering its success in various domains, the application of the Transformer to network traffic data presents a promising avenue for exploration, potentially leading to significant advancements in network security tasks.

B. TabTransformer

TabTransformer [13] builds upon the foundation of the powerful Transformer model. This architecture allows the processing of tabular data by introducing specific mechanisms to handle the inherent characteristics of tables, such as relationships between rows and columns. This adaptation makes TabTransformer a perfect fit for network traffic analysis, where features like source IP address, destination port, and protocol reside in a tabular format [4].

The process begins by converting each category within the data (e.g., TCP options) into a numerical representation called an embedding. This embedding essentially acts as a unique identifier for that category within the network traffic data. These embeddings are then fed through multiple Transformer layers. As the data progresses through these layers, TabTransformer performs its magic. It starts to learn the "context" of each feature, essentially uncovering how it interacts and relates to other features within the network traffic data. For instance, TabTransformer might learn that a specific order in TCP options might distinguish different network stack implementations.

Finally, these contextual embeddings, rich with the learned relationships between features, are combined with any continuous features present in the data (e.g., TTL value). This combined information is then fed into a final layer, often a Multi-Layer Perceptron (MLP), which utilises this comprehensive representation to make predictions. In the context of network security, this prediction might involve identifying OSs, malicious traffic patterns or flagging suspicious network activity.

A diagram of the architecture of TabTransformer can be found in Figure 1. In essence, TabTransformer leverages the strengths of Transformers, particularly their ability to learn complex relationships within sequential data. By applying this capability to tabular network traffic data, TabTransformer can automatically discover hidden patterns and interactions between various features. This newfound understanding of network traffic dynamics has the potential to significantly improve performance in network security tasks, leading to more robust and adaptable security solutions.

In this work we use the official implementation of TabTransformer available in [33].

IV. Methodology

A. Dataset

Our research utilised the signature database from the latest version (7.94) of Nmap [1] as the foundation for our dataset construction. This text file encompasses the entire collection of community-gathered fingerprints, categorised by Nmap since 1998. Each fingerprint represents the combined results of predefined tests that Nmap executes against response data received from a known operating system. To streamline our approach, we filtered the fingerprints, selecting only those belonging to a predefined group of the most prevalent contemporary OS families: Android, BSD, Linux, Solaris, Windows, iOS, and macOS. We have grouped operating systems into three groups (Windows, Linux, and Other) to simplify the initial tests performed in this work.

These features encompassed various information types, including numerical and boolean values, alongside specific categories. For all data types, we implemented a coding scheme that converted the data into numerical representations. Additionally, conditions involving multiple values, expressed using combinations of boolean OR value ranges, "greater than" or "less than" operators, were accommodated. We represented the OR operator as a linear combination between all possible feature values within a single fingerprint, generating a new row for each viable alternative. Conversely, the range operator was addressed by assigning a random value within the specified range for each row within the same fingerprint. Due to their minimal presence in the database, the "greater than" and "less than" operators were disregarded.

To enhance the effectiveness of subsequent stages, the dataset underwent further preprocessing steps, including the removal of duplicate entries. This process resulted in a final dataset comprised of 4.397 instances and 260 features. These features are classified in numerical (53), categorical (206) and the target label of the OS (1).

B. Deep Learning Pipeline

OS fingerprinting task is layed out as a multiclass classification problem. Our methodology begins with data preprocessing. Here, we study how the data is encoded in the Nmap database, the types of features present (categorical and numerical), and any applicable feature engineering techniques. Normalisation was applied to numerical features, and one-hot encoding technique was used for labels. As categorical features will be encoded as embeddings in the Transformer, they are not one-hot encoded but ordinal encoded, transforming their values from string to integer type. This preprocessing step is crucial for preparing the data for effective utilisation within the transformer model.

The final step of data pre-processing involves splitting the prepared data into training and testing sets, in a 80-20 portion. A stratified splitting strategy is employed to guarantee that the class distribution is mirrored across both sets. This ensures the model encounters a representative sample of all operating systems during training and that the evaluation on the testing set reflects real-world class imbalances. To ensure the reproducibility of the results, the random state seed is set during the splitting process.

Once the data is pre-processed, a hyperparameter optimisation framework comes into play to identify the ideal configuration for the TabTransformer model. The optimisation process explores various hyperparameters:

- The learning rate controls the magnitude of updates the model makes to its internal parameters during training. A well-tuned learning rate steers the model towards the optimal solution without getting stuck in local minima or overshooting the minimum. In this case, we try with the values 0.001, 0.01 and 0.1.
- The embedding dimension refers to the dimensionality of the vector representation learned for each feature by the transformer architecture. This essentially dictates the level of detail captured for each feature within the model. The optimisation process searches through a predefined list of discrete values for the embedding dimension, allowing the exploration of different levels of feature granularity to determine what works best for the specific fingerprinting data. The values used for this hyperparameter are 8, 16 and 32.
- The depth of the transformer architecture, specified by the number of encoder-decoder layers, determines the model's capacity to learn intricate relationships between the features. A deeper architecture allows for more complex feature interactions to be captured, but also increases the risk of overfitting. The optimisation process considers a predefined list of discrete depths, enabling the selection of an architecture that strikes a balance between capturing complex relationships and generalizability. The number of layers used during this process are 1, 2 and 3.
- The number of attention heads in the self-attention layers of the transformer architecture signifies the level of parallelism employed in identifying relationships between features. More attention heads allow for capturing a wider range of feature interactions simultaneously, but can also increase computational complexity. The optimisation process explores a predefined list of discrete values for the number of attention heads to determine the setting that yields the most effective balance between capturing feature relationships and computational efficiency. The values used in this case are 2, 3 and 4.
- Dropout rates are hyperparameters that introduce a random element during training. During each training step, a certain percentage of neurons are randomly dropped out, preventing the model from overfitting to the training data. The optimisation process considers a predefined list of dropout rates for both the attention layers and feed-forward layers, searching for the dropout rates that promote effective learning while mitigating overfitting. The values of both dropouts are 0.01, 0.1 and 0.5.

The optimiser used for updating the model's parameters in response to the computed gradients is AdamW, which is a variant of the Adam optimiser [34]. Adam is a popular choice for training deep learning models because it combines the advantages of two other extensions of stochastic gradient descent: AdaGrad and RMSProp. The "W" in AdamW refers to weight decay, a form of L2 regularisation that can help prevent overfitting.

The loss function measures the difference between the model's predictions and the actual values. The loss function

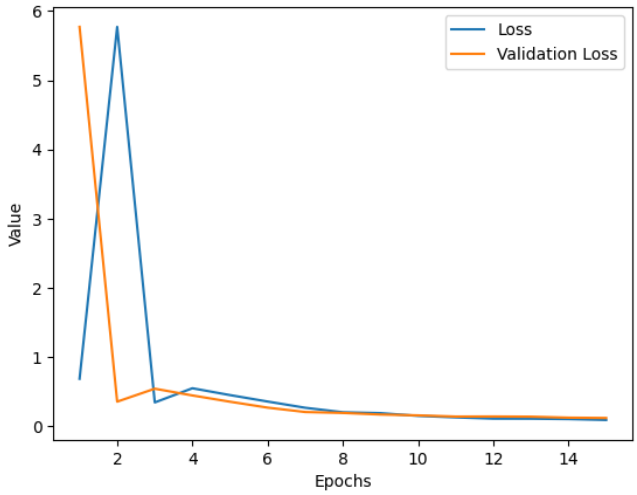

Figure 2. Evolution of the loss during training

used here is Binary Cross Entropy with Logits Loss. It combines a Sigmoid layer and the BCELoss in one single class. This version is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.

A Tree-structured Parzen Estimator (TPE) sampling method guides the exploration of these hyperparameters. TPE is an efficient algorithm that iteratively allocates resources towards trying promising hyperparameter configurations while avoiding redundant evaluations. A maximum of 50 trials are allocated for this exploration, and multiple threads are leveraged to concurrently evaluate different hyperparameter settings, accelerating the optimization process.

Within each trial, the AUC-ROC (Area Under the Receiver Operating Characteristic Curve) metric is used to assess the model's performance on a validation set (10%) derived from the training data. AUC-ROC is a single metric for assessing binary classification models. It considers the trade-off between correctly identifying positive cases (true positive rate) and mistakenly classifying negative cases as positive (false positive rate) across all possible classification thresholds. For multiclass the metric is calculated by iteratively treating each class as the positive class and all other classes as the negative, which is referred to as the one-vs-rest approach, calculating the average over all classes.

After the search is finished, we take the best combinations of hyperparameters and retrain the model with the training set. After it, we evaluate its performance against the initially hold-out test set.

V. Results & Discussion

The hyperparameter optimization process provided valuable insights into the model's behaviour under various configurations. We visualised the training and validation loss curves across epochs in Figure 2. The training loss steadily decreases with epochs, indicating the models successfully learned the underlying patterns within the data. The validation loss curve closely follows the training loss curve, suggesting the models achieved a good balance between learning and generalizability.

Table I
Best hyperparameter values

Hyperparameter	Value
Learning Rate	0.001
Embedding Dimension	8
Depth	1
Heads	3
Attention Dropout	0.5
Feed Forward Dropout	0.01

The hyperparameter optimization process identified a specific configuration that yielded the best performance for the TabTransformer model. This optimal configuration included the values shown in Table I.

The final stage of the evaluation involved assessing the model's performance on unseen data. We employed the models with the identified optimal hyperparameters on the testing set. The achieved AUC-ROC score on the testing set was 0.9919, signifying exceptional performance in correctly classifying operating systems between the predefined set: Windows, Linux or Other.

The results are encouraging and demonstrate the effectiveness of TabTransformer model for operating system fingerprinting when equipped with carefully chosen hyperparameters. The outstanding AUC-ROC score of 0.9919 on the testing set highlights the model's remarkable ability to distinguish between different operating systems.

However, It's crucial to acknowledge limitations of the performed experiments. The evaluation was conducted on a single dataset, and the generalizability of these findings to other datasets might require further investigation. Additionally, exploring the impact of different feature engineering techniques or preprocessing steps, as well as expanding the classes of the classification task to different OSs families and versions, are interesting avenues for future work.

ACKNOWLEDGEMENTS

This work was supported by the grant **ED431C 2022/46 – Competitive Reference Groups GRC** – funded by: EU and Xunta de Galicia (Spain). This work was also supported by **CITIC**, funded by Xunta de Galicia through the collaboration agreement between the Consellería de Cultura, Educación, Formación Profesional e Universidades and the Galician universities to strengthen the research centres of the Sistema Universitario de Galicia (CIGUS); and by the **"Formación de Profesorado Universitario" (FPU)** grant from the Spanish Ministry of Universities to Rubén Pérez Jove (Grant FPU22/04418)

REFERENCES

[1] nmap.org, "Nmap: the Network Mapper - Free Security Scanner." [Online]. Available: https://nmap.org/

[2] F. Veysset, O. Courtay, and O. Heen, "New Tool And Technique For Remote Operating System Fingerprinting," p. 13, 2002.

[3] O. Arkin and F. Yarochkin, "A "Fuzzy" Approach to Remote Active Operating System Fingerprinting," p. 20, 2002.

[4] M. Laštovička, M. Husák, P. Velan, T. Jirsík, and P. Čeleda, "Passive operating system fingerprinting revisited: Evaluation and current challenges," p. 109782. [Online]. Available: https://doi.org/10.1016/j.comnet.2023.109782

[5] M. Zalewski, "p0f v3." [Online]. Available: https://lcamtuf.coredump.cx/p0f3/#/p0f.shtml

[6] "Ettercap Home Page." [Online]. Available: https://www.ettercap-project.org/

[7] "NetworkMiner - The NSM and Network Forensics Analysis Tool ." [Online]. Available: https://www.netresec.com/?page=NetworkMiner

[8] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*, third edition, global edition ed., ser. Prentice Hall series in artificial intelligence. Pearson.

[9] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. New York: Springer, 2006.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. The MIT Press.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need." [Online]. Available: https://doi.org/10.48550/arXiv.1706.03762

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale." [Online]. Available: https://doi.org/10.48550/arXiv.2010.11929

[13] X. Huang, A. Khetan, M. Cvitkovic, and Z. Karnin, "TabTransformer: Tabular data modeling using contextual embeddings." [Online]. Available: https://doi.org/10.48550/arXiv.2012.06678

[14] J. Song, C. Cho, and Y. Won, "Analysis of operating system identification via fingerprinting and machine learning," *Computers & Electrical Engineering*, vol. 78, pp. 1–10, Sep. 2019. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2019.06.012

[15] M. Laštovička, A. Dufka, and J. Komárková, "Machine Learning Fingerprinting Methods in Cyber Security Domain: Which one to Use?" in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Jun. 2018, pp. 542–547. [Online]. Available: https://doi.org/10.1109/IWCMC.2018.8450406

[16] M. Laštovička, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive os fingerprintingmethods in the jungle of wireless networks," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2018, pp. 1–9. [Online]. Available: https://doi.org/10.1109/NOMS.2018.8406262

[17] I. Cvitić, D. Peraković, M. Periša, and B. Gupta, "Ensemble machine learning approach for classification of IoT devices in smart home," vol. 12, no. 11, pp. 3179–3202. [Online]. Available: https://doi.org/10.1007/s13042-020-01241-0

[18] P. M. Sánchez Sánchez, A. Huertas Celdrán, G. Bovet, and G. Martínez Pérez, "Single-board device individual authentication based on hardware performance and autoencoder transformer models," vol. 137, p. 103596. [Online]. Available: https://doi.org/10.1016/j.cose.2023.103596

[19] P. M. S. Sánchez, A. Huertas Celdrán, G. Bovet, G. M. Pérez, and B. Stiller, "A trustworthy federated learning framework for individual device identification," in *2023 JNIC Cybersecurity Conference (JNIC)*, pp. 1–8. [Online]. Available: https://doi.org/10.23919/JNIC58574.2023.10205950

[20] R. Pérez-Jove, C. R. Munteanu, A. P. Sierra, and J. M. Vázquez-Naya, "Applying Artificial Intelligence for Operating System Fingerprinting," *Engineering Proceedings*, vol. 7, no. 1, p. 51, 2021. [Online]. Available: https://doi.org/10.3390/engproc2021007051

[21] R. Pérez-Jove, C. R. Munteanu, J. Dorado, A. Pazos, and J. Vázquez-Naya, "Operating system fingerprinting tool based on classical machine learning algorithms," in *2023 JNIC Cybersecurity Conference (JNIC)*. IEEE, pp. 1–8. [Online]. Available: https://doi.org/10.23919/JNIC58574.2023.10205734

[22] A. Louis and U. d. L. \. M. i. c. s. d. Fin, "NetBERT: A pre-trained language representation model for computer networking." [Online]. Available: https://matheo.uliege.be/handle/2268.2/9060

[23] F. Le, D. Wertheimer, S. Calo, and E. Nahum, "NorBERT: NetwOrk representations through BERT for network analysis & management," in *2022 30th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, pp. 25–32. [Online]. Available: https://doi.org/10.1109/MASCOTS56607.2022.00012

[24] B. Pang, Y. Fu, S. Ren, S. Shen, Y. Wang, Q. Liao, and Y. Jia, "A multi-modal approach for context-aware network traffic classification," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 1–5. [Online]. Available: https://doi.org/10.1109/ICASSP49357.2023.10095124

[25] O. Barut, Y. Luo, P. Li, and T. Zhang, "R1dit: Privacy-preserving malware traffic classification with attention-based neural networks," vol. 20, no. 2, pp. 2071–2085. [Online]. Available: https://doi.org/10.1109/TNSM.2022.3211254

[26] R. Zhao, Y. Huang, X. Deng, Z. Xue, J. Li, Z. Huang, and Y. Wang, "Flow transformer: A novel anonymity network traffic classifier with attention mechanism," in *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE, pp. 223–230. [Online]. Available: https://doi.org/10.1109/MSN53354.2021.00045

[27] S. Guthula, N. Battula, R. Beltiukov, W. Guo, and A. Gupta, "netFound: Foundation model for network security." [Online]. Available: https://doi.org/10.48550/arXiv.2310.17025

[28] F. Le, M. Srivatsa, R. Ganti, and V. Sekar, "Rethinking data-driven networking with foundation models: challenges and opportunities," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, ser. HotNets '22. Association for Computing Machinery, pp. 188–197. [Online]. Available: https://doi.org/10.1145/3563766.3564109

[29] A. Dietmüller, S. Ray, R. Jacob, and L. Vanbever, "A new hope for network model generalization," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, ser. HotNets '22. Association for Computing Machinery, pp. 152–159. [Online]. Available: https://doi.org/10.1145/3563766.3564104

[30] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proceedings of the ACM Web Conference 2022*. ACM, pp. 633–642. [Online]. Available: https://doi.org/10.1145/3485447.3512217

[31] N. Tran, H. Chen, J. Bhuyan, and J. Ding, "Data curation and quality evaluation for machine learning-based cyber intrusion detection," vol. 10, pp. 121 900–121 923. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3211313

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding." [Online]. Available: https://doi.org/10.48550/arXiv.1810.04805

[33] P. Wang, "lucidrains/tab-transformer-pytorch." [Online]. Available: https://github.com/lucidrains/tab-transformer-pytorch

[34] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," no. arXiv:1711.05101. arXiv. [Online]. Available: https://doi.org/10.48550/arXiv.1711.05101