INL/CON-22-68438-Revision-2

Machine Learning 5G Attack Detection in Programmable Logic

January 2023

Matthew William Anderson, Cooper Wayne Coldwell, Matthew R Sgambati, Brendan Glen Jacobson, Bryton John Petersen, Damon Renel Spencer, Denver Shane Conger, Edward Goodell

Changing the World's Energy Future

INL is a U.S. Department of Energy National Laboratory operated by Battelle Energy Alliance, LLC

INL/CON-22-68438-Revision-2

Machine Learning 5G Attack Detection in Programmable Logic

Matthew William Anderson, Cooper Wayne Coldwell, Matthew R Sgambati, Brendan Glen Jacobson, Bryton John Petersen, Damon Renel Spencer, Denver Shane Conger, Edward Goodell

January 2023

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Machine Learning 5G Attack Detection in Programmable Logic

Cooper Coldwell*†, Denver Conger*‡, Edward Goodell*§,
Brendan Jacobson*‡, Bryton Petersen*‡, Damon Spencer*¶,
Matthew Anderson*, Matthew Sgambati*
* Idaho National Laboratory, Idaho Falls, ID
† The University of Alabama, Tuscaloosa, AL
‡ Brigham Young University Idaho, Rexburg, ID
§ The University of Utah, Salt Lake City, UT
¶ University of Houston, Houston, TX

***Abstract*—Machine learning-assisted network security may significantly contribute to securing 5G components. However, machine learning network security inference generally requires tens to hundreds of milliseconds, thereby introducing significant latency in 5G operations. The inference latency can be reduced by deploying the machine learning model to programmable logic in a field programmable gate array at the cost of a small loss in accuracy. In order to quantify this loss, as well as to establish baseline performance inference latency for programmable logic implementations, this work explores an autoencoder and a β-variational autoencoder deployed on two different field programmable gate array evaluation boards and compares accuracy and performance against an NVIDIA A100 graphics processing unit implementation. A publicly available 5G dataset containing 10 types of attacks along with normal traffic is introduced as part of the evaluation.**

I. Introduction

Machine learning-assisted network security has gained significant research attention in the context of 5G security, and multiple studies have explored how machine learning may become an important component for securing 5G communications [3, 6, 8]. Ultra-low latency is a fundamental characteristic of 5G; however, the same is not generally true for machine learning inference. Deep machine learning models can take tens to hundreds of milliseconds per inference—thereby limiting their application to 5G security regardless of their ability to identify anomalous or malicious behavior.

The inference speed of a machine learning model can be significantly accelerated by leveraging the deep pipeline parallelism available in programmable logic. Furthermore, this acceleration is possible without increasing power consumption typically seen when using graphics processing units (GPUs). Using the programmable logic in field programmable gate arrays (FPGAs), high performance inference pipelines can be built featuring ultra-low latency without the need for batching as would typically be required for GPUs [4]. One issue with FPGA implementations of machine learning models is the tendency to lose some inference accuracy compared to GPU implementations. This is a consequence of quantizing the model for use in programmable logic. To investigate the inference latency and accuracy for a machine learning-based 5G attack detection system, this work develops and deploys an autoencoder (AE) and a β-variational autoencoder (β-VAE) [12] for 5G attack detection on the Xilinx ZCU104 and Xilinx VCK190 evaluation boards. The inference metrics from the AE and β-VAE implementations on FPGA are then compared against the latency and accuracy achieved on an NVIDIA A100 GPU.

To evaluate the latency and accuracy of these models, a new 5G network traffic dataset is also introduced as part of this work consisting of normal traffic and 10 types of 5G attacks. This dataset, called the 5G Attack Detection (5GAD-2022) dataset, is made publicly available [13] to facilitate performance and accuracy comparisons of the models presented in this work. In this study, the AE and β-VAE are trained solely using normal traffic. These models are then tested by passing them normal traffic mixed with 5G attacks in order to measure how well anomalous packets can be identified and how quickly the inference pipeline operates.

This work is structured as follows: related work for machine learning applied to 5G network security is in Section II; a description of the 5GAD-2022 dataset and an overview of explored machine learning models are in Section III; details regarding the FPGA implementation are in Section IV; latency and accuracy results for the models are in Section V; conclusions and future work are in Section VI.

II. Related Work

There are multiple efforts underway to deploy machine learning models to help secure 5G networks. In [10], a convolutional neural network (CNN) model was explored in an effort to classify 5G packets via supervised learning but not in the context of network security. Similarly, in [14], a CNN was deployed for classifying network traffic with anomaly detection. In [7], a support vector machine model and multilayer perceptron model were used to detect attacks on mobile networks, but the study was limited to radio signals. In [2], distributed denial-of-service attacks were identified with several artificial intelligence algorithms, including a random forest classifier and logistic regression. Similarly, [5] explored

anomaly detection and classification in network traffic using a random forest classifier and k-means clustering. None of the related works have explored anomaly detection via an AE or β-VAE nor have they explored implementations in programmable logic. Therefore, this work fills an important gap for 5G security applications utilizing machine learning.

III. Preliminaries

A. The 5G Attack Detection Dataset

The 5GAD-2022 dataset consists of intercepted 5G network data by a 3rd party attacker, including both normal and malicious traffic. The data was generated in a simulated environment and collected via an internet-connected Linux machine running a 5G Core (5GC) network implemented with the open source free5GC [1] software. The 5GC was connected through Ethernet to another device which simulated user equipment (UE) and the radio-access network using another open source software called UERANSIM [11]. Wireshark was utilized to capture all network traffic on the 5GC machines interfaces. The structure of the 5G network with IP addresses in 5GAD-2022 is shown in Figure 1.

The "normal" data fall into two categories: data collected while running one UE simulation and data collected while running two UE simulations. In both scenarios, a facsimile of typical user traffic was created by streaming YouTube videos, making HTTP/ HTTPS requests to commonly visited websites, joining video conference meetings with chats, making FTP requests and downloads, and accessing Samba servers for image, video, and document uploads and downloads.

The malicious data are made up of ten attacks that fall under three main categories: reconnaissance, denial-of-service (DOS), and network reconfiguration. These attacks work under the assumption that a 3rd party attacker has illegitimate communication access to certain components of 5GC due to inadequate security configurations. The high-level strategies of these attacks are based on vulnerabilities documented by Positive Technologies [15, 16] or based on flaws found in free5GC. Additional details of the attacks are outlined below.

1) Reconnaissance Attacks: The "Access and Mobility Management Function Looking for Unified Data Management" attack is performed by requesting information about the unified data management (UDM) network function while impersonating an access and mobility management function (AMF). Internally, this attack appears to be a benign system request and exploits the fact that the network repository function (NRF) does not check if the source of the request is actually an AMF. This attack is executed as follows:

```
curl "http://127.0.0.10:8000/nnrf-disc/v1/nf-inst
ances?requester-nf-type=AMF&target-nf-type=UDM"
```

where 127.0.0.10 is the IP address of the NRF.

The "Get All Network Functions" attack is performed identically to "AMF Looking for UDM" except the target-nf-type is not specified. This results in the NRF returning all network functions (NF) to the requester.

The "Get User Data" attack requests information from the UDM regarding a user with `subscriberID=0000000003`. This attack is executed as follows:

```
curl "http://127.0.0.3:8000/nudm-dm/v1/imsi-20893
${subscriberID}/am-data?plmn-id=%7B%22mcc%2
2%3A%22208%22%2C%22mnc%22%3A%2293%22%7D"
```

The "Random Data Dump" attack exploits a lack of input validation in free5GC and sets the requester-nf-type to a random string when making a `nf-instances` request to the NRF. The NRF will still respond with all of the NFs. This attack is executed as follows:

```
curl "http://127.0.0.10:8000/nnrf-disc/v1/nf-inst
ances?requester-nf-type=$randomString&target-nf
-type="
```

The "Automatic Redirect with Timer" attack is based on Positive Technologies' report: *5G Standalone core security research* Section 4.3 [15], which listens to network traffic while the UE is connecting. The attack code listens for a packet forwarding control protocol session establishment

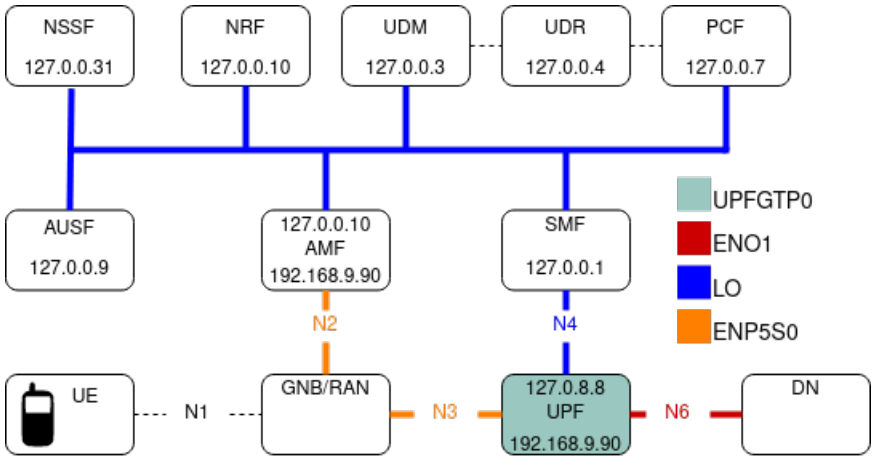

Fig. 1: The 5G network with IP addresses as used in the 5GAD-2022 dataset.

request, then checks if the UE address is in a list of victim addresses. If the UE address is listed, the attack records session information that it uses to redirect traffic from the UE. This is achieved by sending a spoofed packet forwarding control protocol session modification request to the user plane function with the discovered session ID and forwarding action rule ID. The attack will send two such modification requests, wait 5 seconds, send two more modification requests to return the UE to its normal path, wait 5 more seconds, and repeat.

2) Network Reconfiguration Attacks: The "Fake AMF Insert" attack registers a fake AMF with the NRF. This is achieved by running the `curl` command to `PUT` a JSON object to the NRF. In the environment where this attack was run, there is no check on authority to prevent the attacker from registering a fake AMF.

The "Fake AMF Delete" attack (described in Section III-A3) is subsequently run to remove the fake AMF. The instance-ID must be a Version 4 universally unique identifier (UUID); however, free5GC does not check the instance ID or other details about the AMF before adding it to the core. This includes whether or not the instance ID is a properly formatted UUID, which consists of hexadecimal values in a string. A few 1's in the attack's UUID string were replaced with *l*'s while writing the attack code, so the string was not correctly formatted as hexadecimal. The instance ID was therefore an invalid UUID; regardless, the instance ID was accepted by free5GC.

The "Random AMF Insert" attack is the same as a "Fake AMF Insert," except the instance ID is a randomly generated string.

3) Denial-of-Service Attacks: The "Crash Network Repository Function Attack" attack on the 5GC relies on an exploit in free5GC wherein a malformed request to the NRF will cause it to crash. This attack is executed as follows:

```
curl "http://127.0.0.10:8000/nnrf-disc/v1/nf-inst
ances?requester-nf-type=&target-nf-type="
```

where 127.0.0.10 is the IP address of the NRF. As of free5GC v3.1.1, this exploit appears to have been patched, as this HTTP `GET` request will no longer result in failure of the core.

The "Fake AMF Delete" attack is run in conjunction with the "Fake AMF Insert" and is executed as follows:

```
curl -s -o /dev/null -w "\n\nHTTP Status Code:
%{http_code}\n\n" -X DELETE http://127.0.0.10
:8000/nnrf-nfm/v1/nf-instances/$fakeAMF
```

where `fakeAMF` is the hexadecimal session ID of the false AMF inserted into the system. This attack coupled with "Get All NFs" to find other AMFs could be exploited to remove legitimate AMFs from the network, disrupting network functionality.

The "Automated Drop with Timer" attack is similar to the "Automatic Redirect with Timer" attack but alternates between redirecting user traffic and dropping user traffic, effectively disconnecting the user from the data network. This attack is based on Positive Technologies' report, Section 4.2 [15].

B. Data Preparation

To ensure that information, such as the source and destination IP addresses, did not influence the model training, all packets were stripped down until only the application layer remained. This could be done without fear of losing important distinctions between normal and attack packets because all of the attacks were contained in the application layer. Packets that did not contain an application layer were dropped from the training set. Every packet was then truncated to 1,024 bytes or padded with 0s until it reached a length of 1,024 bytes. Truncating packets increased the training speed while still fully including the majority of the packet payloads, since most attack packets were shorter than 1,024 bytes.

C. Model Design

The AE and β-VAE were each trained on normal data to create latent spaces. The latent spaces were then passed into a one-class neural network (OCNN), which made predictions based on the clustering seen in the latent spaces.

1) Traditional Autoencoder Model Design: The autoencoder is a vanilla autoencoder and the encoder takes in packets of length 1024 bytes and reshapes them to 32x32x1. The square tensor is then compressed down into an 8x8x1 tensor using four Conv2D layers each followed by LeakyReLU activation layers with an α value of 0.1. The data is then flattened and reduced down to a dimension of 3 using a Dense layer. This tensor of dimension 3 defines the latent space which can be passed into the decoding model.

The decoder takes the 3D latent space from the encoder and up-scales it back to 1024x1x1. This is done by passing the data through four Conv2DTranspose layers each followed by LeakyReLU activation functions with α values of 0.1. The encoder and decoder are then combined and compiled as a single model. Upon training, reconstruction loss is used to

Hyperparameters	Test Cases	Optimal β-VAE Params.	Optimal AE Params.
Learning Rate	5e-5, 1e-4, 1e-3, 5e-3, 1e-2	1e-4	1e-3
Latent Space Dimension	2, 3, 4, 5	3	3
β Value	1e-4, 1e-3, 1e-2, 1e-1, 3e-1, 1, 1.5	0.22	N/A
Epochs	2, 3, 5, 7, 10, 20, 50, 100	100	3
β-VAE Loss Function	KLDivergence+BCE, KLDivergence+MSE	KLDivergence+MSE	N/A
AE Loss Function	BCE, MSE, KLDivergence	N/A	MSE

TABLE I: The hyperparameter search space and optimal parameters for the AE and β-VAE models are shown here.

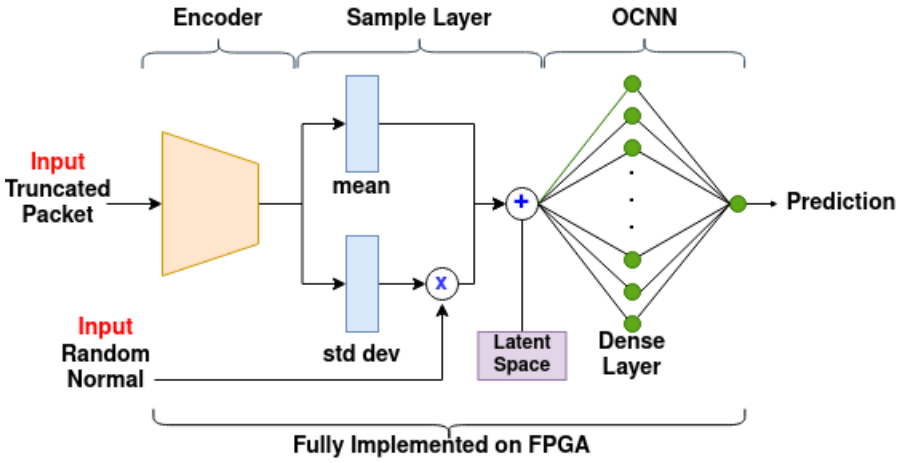

Fig. 2: Implementation of the encoder from the trained β-VAE with the OCNN classifier. The sample layer has been modified from a lambda sample layer to multiplication and addition layers to allow full implementation on FPGA

update weights and biases in both the encoder and decoder. For optimal hyper-parameters for the AE, please see Table I.

2) β-Variational Autoencoder Model Design: The β-VAE architecture is nearly identical to the AE with the exception of the final dense layers in the encoder and a modified sample layer, as seen in Figure 2. Instead of a single dense layer, two three dimensional dense layers are created. The first is multiplied by a random number taken from a normal distribution and is used to calculate KL Divergence loss. The second is used to calculate reconstruction loss. The latent space is then created by multiplying KL Divergence loss by the β value and adding it to the reconstruction loss. A representation of the latent space created by the β-VAE is shown in Figure 3 where a clear distinction between normal and anomalous packets is visible. The random numbers used to calculate KL Divergence loss are passed in as inputs to the model and are not computed on the FPGA. Optimal hyperparameters for the β-VAE are shown in Table I.

3) One-Class Neural Network: The unsupervised model used to classify encoded packets is the OCNN as described by Chalapathy et al. [9]. Encoded normal packets were used to train the OCNN, allowing the OCNN to later be used in conjunction with an encoder to detect anomalous packets.

IV. FPGA Deployment

The models were deployed using Vitis AI 2.5. The β-VAE model required the creation of a custom sample layer to be fully implementable on the deep processing unit of the FPGA, see Figure 2. This was done by calculating standard deviation (σ) in place of the log variance for simpler model computations that could be implemented on the FPGA. The standard deviation is computed by creating a dense layer for the standard deviation with a RELU activation function and adding epsilon via another dense layer with no activation function. The second dense layer has identity weights and a bias term of epsilon. While this technically means the standard deviation cannot be smaller than epsilon, the error introduced by this is extremely negligible and is counterbalanced by increased speed. Additionally, the β-VAE includes the random normal sample as a secondary input instead of a layer. This was done for ease of implementation. The normally distributed random numbers are produced on the CPU in large batches and passed into the FPGA as the secondary input. An LFSR and Box-Muller approach for normally distributed random number generation on the FPGA would be an improvement for the future.

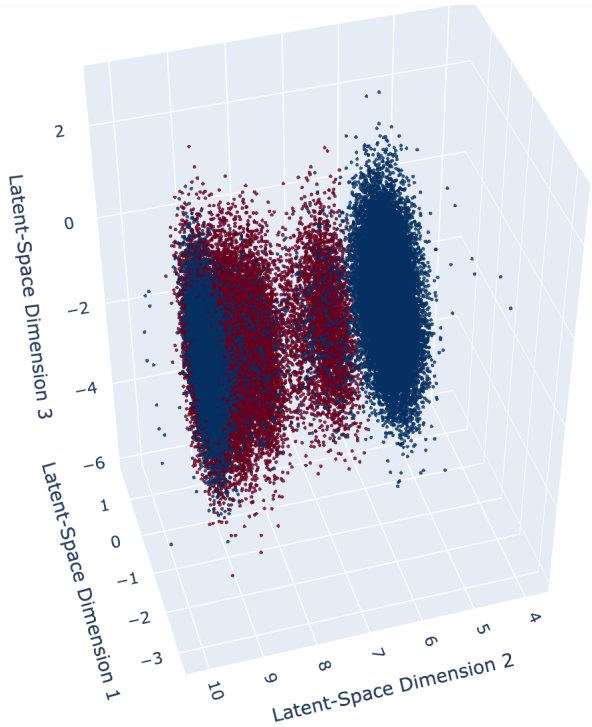

Fig. 3: Latent space generated from β-VAE. The red points represent anomalous packets and the blue points represent normal packets. Distinct clustering in the latent space between normal packets and anomalous packets is evident.

	FPGA								GPU			
	ZCU104				VCK190				A100			
Model	Latency	Packets/Sec	F1	Accuracy	Latency	Packets/Sec	F1	Accuracy	Latency	Packets/Sec	F1	Accuracy
β-VAE	0.307 ms	6.63k	0.848	84.8%	0.352 ms	23.6k	0.831	83.5%	1.31 ms	26.0k	0.889	88.9%
AE	0.236 ms	9.74k	0.926	92.6%	0.278 ms	33.0k	0.917	91.6%	1.15 ms	33.6k	0.919	92.0%

TABLE II: A comparison of the latency, throughput, F1 score, and accuracy for the unsupervised learning models explored in this work on an NVIDIA A100 GPU and two different FPGA evaluation boards, the ZCU104 and the VCK190.

V. Results

The latency, throughput, F1 score, and accuracy for the AE and β-VAE models on both FPGA and GPU using the 5GAD-2022 dataset are given in Table II. The FPGA implementations of the β-VAE model lost 4–6% in accuracy compared with the GPU implementation. In contrast, there was no significant accuracy difference between the FGPA and GPU implementations of the AE model. This is because the β-VAE model is more susceptible to quantization error in the modified sample layer (see Figure 2), as described in Section IV.

The inference latency for both the AE and β-VAE models was significantly lower on the FPGAs than on the A100 GPU by a factor ranging from 3.7 to 4.8. Ultra-low latency is the critical component for 5G. The throughput is much higher for the VCK190 than the ZCU104 and is comparable to the throughput measured on the A100 GPU. In the throughput experiments, the ZCU104 used three threads and a batchsize of one, the VCK190 used three threads and a batchsize of six, and the A100 used a batchsize of 32.

VI. Conclusions

Machine learning models are well positioned to identify anomalous network traffic in 5G communications. This work has explored two different machine learning models trained only on normal 5G network traffic and then used to detect anomalous behavior. A public 5G dataset has been created and introduced here containing 10 types of 5G attacks in order to test the ability of an AE and β-VAE model to detect anomalous network traffic. The models were fully implemented in programmable logic on two different FPGAs in order to leverage the deep pipeline parallelism available and reduce latency while also taking advantage of the FPGA's low-power, high-duty cycle, and cost-effective characteristics. As expected, there was a loss of accuracy of 4–5% in the β-VAE FPGA implementation when compared with GPU due to quantization error. However, the inference latency crucial for use in 5G was nearly 4$\times$ lower or more on the FPGAs than on the GPU.

The FPGA implementations can be further refined by implementing the deep processing unit directly on the bare-metal FPGA and avoiding the Petalinux Operating system. This and other fine tuning of the AE and β-VAE models for improved accuracy on FPGA are left as future work.

VII. Acknowledgements

This research made use of the resources of the High Performance Computing Center at Idaho National Laboratory, which is supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517. We acknowledge Christopher Becker, Jessie Cooper, and the Wireless Security Institute from Idaho National Laboratory for their technical assistance and review. This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

References

[1] free5GC. URL: https://github.com/free5gc/free5gc.

[2] B. S. Rawal et al. Identifying ddos attack using split-machine learning system in 5g and beyond networks. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2022.

[3] C. Zhang et al. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys Tutorials*, 21(3):2224–2287, 2019.

[4] J. Duarte et al. FPGA-Accelerated Machine Learning Inference as a Service for Particle Physics Computing. *Computing and Software for Big Science*, 3(1), 10 2019.

[5] J. Li et al. Machine learning-based ids for software-defined 5g network. *IET Networks*, 7(2):53–60, 2018.

[6] M. E. Morocho-Cayamcela et al. Machine Learning for 5G/B5G Mobile and Wireless Communications: Potential, Limitations, and Future Directions. *IEEE Access*, 7:137184–137206, 2019.

[7] M. Hachimi et al. Multi-stage jamming attacks detection using deep learning combined with kernelized support vector machine in 5g cloud radio access networks. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–5, 2020.

[8] N. Haider et al. Artificial Intelligence and Machine Learning in 5G Network Security: Opportunities, advantages, and future research trends. 2020. URL: https://arxiv.org/abs/2007.04490.

[9] R. Chalapathy et al. Anomaly detection using one-class neural networks. *arXiv preprint arXiv:1802.06360*, 2018.

[10] W. Chen et al. A cnn-based packet classification of embb, mmtc and urllc applications for 5g. In *2019 International*

Conference on Intelligent Computing and its Emerging Applications (ICEA), pages 140–145, 2019.

[11] Ali Güngör. UERANSIM. URL: https://github.com/aligungr/UERANSIM.

[12] D. P. Kingma and M. Welling. Auto-encoding variational bayes. 2013. URL: https://arxiv.org/abs/1312.6114.

[13] Idaho National Laboratory. 5GAD-2022. https://github.com/IdahoLabResearch/5GAD, 2022.

[14] J. Lam and R. Abbas. Machine learning based anomaly detection for 5g networks. *CoRR*, 2020.

[15] Positive Technologies. 5G Standalone Core Security Research, 2020. URL: https://img.lightreading.com/5gexchange/downloads/5G-Standalone-core-security-research.pdf.

[16] Positive Technologies. Threat vector: GTP, 2020. URL: https://img.lightreading.com/5gexchange/downloads/5G-Standalone-core-security-research.pdf.