Detection of Stealthy Bitstreams in Cloud FPGAs using Graph Convolutional Networks*

Jayeeta Chaudhuri and Krishnendu Chakrabarty
School of Electrical, Computer, and Energy Engineering
Arizona State University, Tempe, AZ, USA

2024 IEEE European Test Symposium (ETS) | 979-8-3503-4932-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ETS61313.2024.10567821

***Abstract*—FPGAs are frequently utilized in cloud computing environments for high performance computing and neural network accelerators. Furthermore, multi-tenancy allows multiple users to upload customized modules on the FPGA, while maintaining logical isolation. However, attackers can take advantage of the multi-tenant environment to launch voltage-based attacks and denial-of-service (DoS). An attacker might stealthily split power-wasting ring oscillators (ROs) across multiple windows within an FPGA configuration bitstream, making it challenging for traditional detection mechanisms to identify these dispersed components as part of a larger malicious circuit. We propose a methodology to detect these malicious bitstreams by transforming individual windows within an FPGA bitstream into a graph-based representation. Leveraging this graph structure, our method employs graph convolutional networks (GCNs) in the training phase to capture malicious patterns from the bitstreams. We use the classification accuracy, true-positive rate, and false-positive rate metrics to quantify the effectiveness of our method across diverse power-wasting circuits on multiple FPGA boards.**

I. Introduction

Multi-tenant FPGAs are reconfigurable hardware platforms that are often used in cloud computing centers, high-performance computing, and neural network accelerators. However, the shared environment in multi-tenancy introduces attack vectors that can be exploited by a third-party adversary. Malicious bitstreams implementing ring oscillator (RO)-based circuits can be configured on multi-tenant FPGAs. Composed of an odd number of inverters, ROs can be manipulated for generating high-frequency oscillations, leading to voltage-based attacks and denial-of-service (DoS) attacks.

Prior work on bitstream detection focuses on checking a bitstream before FPGA configuration via reverse-engineering (RE) and machine learning (ML)-based methods. However, RE is a time-intensive procedure and often requires significant modification of the reversal tools to adapt to larger bitstreams [1]. While ML-based methods used in [2] [3] [4] [5] successfully detect and diagnose a wide range of RO variants, loop-free ROs, and several power-wasting circuits, the detection of stealthy, power-wasting Trojans have not been explored. An adversary can craft obfuscated ROs to increase power consumption, while evading detection by ML-based methods that rely on contiguous windows of an FPGA bitstream for malicious circuit detection; these methods do not consider the spatial relationship among the windows [3] [5]. For example, an attacker can split the inverters of an RO design across multiple look-up tables (LUTs) of an FPGA, making the malicious RO patterns in the bitstream appear to be distributed.

Hence, it is crucial to employ detection techniques that consider the spatial context and are capable of identifying dispersed and obfuscated malicious patterns. In this paper, we analyze the impact of stealthy, malicious circuits on multi-tenant FPGAs and present an efficient technique using Graph Convolutional Networks (GCNs) to learn spatial relationships in bitstream data and capture malicious patterns in FPGA bitstreams. Based on a supervised learning approach, GCN leverages both the structural information and the dependencies within bitstream data to detect malicious patterns corresponding to power-wasting circuits. Moreover, we show that the proposed method is generalizable for larger bitstreams used in realistic applications, thus making it a robust and scalable mechanism for enhancing FPGA security.

The key contributions of our paper are as follows.

- We assess the impact of stealthy RO-based circuits on a Pynq FPGA board.
- We present a similarity-based framework to generate a graph adjacency matrix that captures spatial dependencies among windows of an FPGA bitstream.
- We apply the knowledge of the graph embedding generated by a GCN model for a given bitstream's adjacency matrix in order to capture malicious patterns that are indicative of stealthy behavior.
- We present an ML-based classification method that leverages the GCN-generated graph embeddings to identify bitstreams as malicious or benign.
- We generate a variety of bitstreams implementing stealthy power-wasting circuits and benign circuits for training and inferencing of the GCN model.

The remainder of the paper is organized as follows. Section II provides an overview of attacks and countermeasures on multi-tenant FPGAs. The threat model is described in Section III. Section IV presents the security threats from stealthy FPGA bitstreams. Details of our GCN-based malicious bitstream detection framework are provided in Section V. In Section VI, we compare the effectiveness of the proposed method with baseline detection techniques. Finally, Section VII concludes the paper.

II. Background and Motivation

A. Attacks

A power distribution network (PDN) plays a critical role in supplying power to all the modules within multi-tenant FP-

*This work was supported in part by the National Science Foundation under grant no. CNS-2011561.

GAs. The PDN consists of resistive, capacitive, and inductive elements. The voltage drop across the PDN is dependent on the summation of voltage drops across all the reconfigurable modules of the FPGA. An adversary might deploy malicious power-wasting circuits on the FPGA that impacts the PDN, leading to voltage fluctuations and subsequently, DoS. For example, ROs can be maliciously crafted to consume excessive power, thereby disrupting the normal functioning of multi-tenant FPGAs. Even when occupying less than 12% on a Kintex 7 FPGA board, the ROs can successfully crash the FPGA [6]. In [7], glitch generator circuits using XOR gates and delay lines have been implemented; these circuits draw excessive power from the PDN, which can result in undesirable voltage fluctuations affecting other modules on the same FPGA. In extreme cases, such excessive power draw might even lead to DoS of the FPGA. [8] demonstrates power-based attacks on cloud FPGAs using circuits implementing loop-free oscillators. These circuits evade the Design Rule Check (DRC) on Amazon Web Services (AWS) instances, leading to their deployment on the cloud infrastructure and resulting in DoS scenarios. Additional power-wasting circuits have been shown in [9]; by carefully inserting XOR gates between AES rounds or by generating chains of shift registers, the authors demonstrate voltage- and power-based attacks on the FPGA.

B. Prior Art in Detecting FPGA-Based Attacks

Several methods have focused on mitigating the deployment of malicious circuits on multi-tenant FPGAs. AWS offers multiple fences to detect malicious signatures in design checkpoint files that are being uploaded by third-party users [10]. These fences include (a) *netlist checking* that identifies combinational loops and floating nets, (b) *secure bitstream generation* when a netlist is identified as benign, and (c) *run-time power monitoring* of the FPGA to detect unusual voltage fluctuations. However, malicious designs, utilizing glitch generation and non-combinational loops, consume high power but remain deployable on AWS due to the absence of explicit loops.

In [11], a RE-based method is used to detect combinational loops that are indicative of RO circuits in technology-mapped netlists. In this scenario, the netlist needs to be manually analyzed and compared with the original (golden) netlist to identify malicious behavior. Although this technique successfully detects combinational ROs, attackers might circumvent this countermeasure by deploying loop-free oscillators, such as self-clocked ROs and latched ROs [8] [12]. Extending [11], the authors updated their bitstream checking mechanism to detect combinational cycles and timing violations [12]. However, extending the applicability of the RE tool to FPGAs in cloud environments may demand substantial modifications, potentially impacting the cost-effectiveness and scalability. Furthermore, stealthy ROs can be intentionally scattered across various LUTs and logic elements of an FPGA, making it challenging to distinguish them solely via manual inspection. In [13], a malicious bitstream detection technique is proposed that is suitable for large-scale cloud deployments. This methodology detects glitches, short circuits, and large fanouts.

TABLE I: A qualitative comparison of the proposed method with prior work on malicious circuit detection.

Parameter	[3]	[5]	[11]	[14]	[15]	[18]	**Proposed method**
Dataset	B	B	B	B	B	N	B
Threat model	CT	CT	CT	CT	FT	CT	CT, FT
Self-clocked RO detected?	✓	✗	✗	✗	✓	✗	✓
RE used	✗	✗	✓	✓	✓	✗	✗
Spatial analysis?	✗	✗	✗	✗	✗	✗	✓
Generalizability	✓	✓	✗	✓	✓	✗	✓
Detection latency	Low	Low	Med	Med	Med	Low	Low

B: FPGA bitstream; N: RTL Netlist; CT: Co-tenants; FT: FPGA tools.

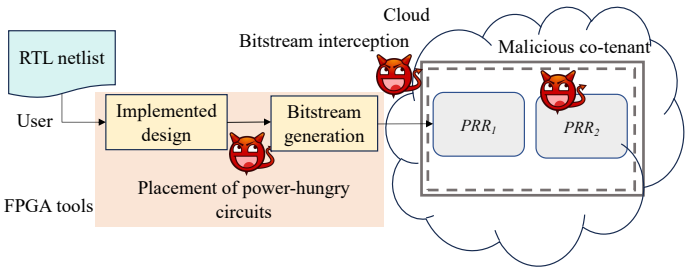


Fig. 1: Threat models considered in this paper.

In [14], the authors propose a run-time countermeasure against voltage-based attacks on FPGAs by disabling the interconnects of an attacker module. Another approach uses RE to identify fan-outs in netlists, but with limited scalability in handling larger and complex FPGA bitstreams [15].

Alternative ML-based approaches were explored in [2] [3], and [5]. These methods have been shown to successfully detect non-combinational ROs and glitch-based circuits. [16], [17], and [18] explore ML-based techniques to detect hardware Trojans from register-transfer level netlists, but they do not specifically address the detection of malicious bitstreams.

Table I presents a qualitative comparison of the proposed framework with prior malicious circuit detection techniques.

C. Motivation

Some of the prior work to analyze FPGA bitstreams via RE-based methods do not evaluate a variety of non-combinational ROs. Recent work [12] targets detection of loop-free ROs but they might be ineffective against emerging stealthy circuits. ML-based detection methods often rely on patterns extracted from contiguous windows in FPGA bitstreams [2] [3] [5]. However, adversaries might intentionally disperse ROs across different windows of the bitstream, such that each window is classified as benign by the ML model. In other words, detecting these dispersed elements becomes a challenging task for ML models that do not consider spatial dependencies present among bitstream data, thus limiting their ability to effectively identify these circuits as being malicious.

III. Threat Model

The threat model is illustrated in Fig. 1. In a multi-tenant scenario, several users can upload their customized modules on the partial reconfigurable regions (PRRs) of the FPGA (marked as PRR_x in Fig. 1). All the PRRs share a common power distribution network (PDN). Thus, an attacker co-tenant can attempt to induce a huge power consumption on the FPGA and disrupt the performance of victim PRR modules. As shown in

[19], activating a large number of ROs at a particular frequency is sufficient to cause significant power consumption, causing the FPGA to shut down automatically. An attacker may deploy malicious stealthy, power-wasting circuits for inducing voltage fluctuations in the PDN of the FPGA, causing DoS. Moreover, an attacker gaining illegitimate access to the placed and routed netlist might embed malicious circuits before bitstream generation. We also consider the scenario where an attacker attempts to alter the bitstream during its transmission, before FPGA deployment. Usually, a bitstream is decrypted before it is configured on the FPGA [20]. Therefore, we evaluate our method specifically on decrypted bitstreams.

We assume that the FPGA is configured within a trusted cloud environment. The proposed detection framework operates within this trusted cloud environment, external to the FPGA chip, and is therefore secure against unauthorized access. Therefore, an attacker would be unable to corrupt an FPGA bitstream after its verification using our method.

IV. Analysis of Stealthy FPGA Bitstreams

A. Dataset Generation

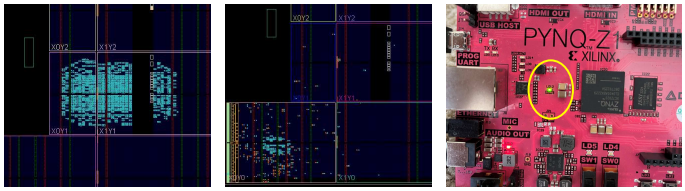
For our experimental analysis, we generate a dataset of benign and malicious bitstreams. We obtain the benign circuits from diverse sources, including the ISCAS'85, ITC'99, and EPFL benchmarks, alongside the OpenCore repository including AES cores, microcontrollers, and arithmetic cores. To generate the dataset of malicious bitstreams, we implement a variety of power-wasting circuits:

1) N_{RO} single-stage ROs, where N_{RO} is sufficient to cause voltage-based attacks on the Pynq and Virtex FPGAs [6].
2) N_{RO} conditional ROs are selectively enabled using multiplexers to induce voltage fluctuations on the FPGA.
3) Non-combinational ROs, including latched and self-clocked oscillators, implemented in [8] [12].
4) Glitch generator circuits that can result in DoS of the FPGA are implemented using delay stages, which feed into N-input XOR gates, $2 \leq N \leq 10$, as detailed in [7].
5) Power-wasting AES circuits and chains of shift register circuits are evaluated, per [9].
6) For this work, we specifically implement *stealthy circuits* by dispersing the RO inverters of circuits detailed in (1)-(3). Stealthy RO-based circuit generation involves strategically scattering the inverters of ROs across different LUT blocks and slices within the FPGA architecture, aiming to obscure their presence within the hardware.

Evaluations were conducted across three distinct FPGA boards: Virtex Ultrascale (VU440), Kintex Ultrascale (KU085), and Pynq xc7z020 (Pynq). We generate 152 benign bitstreams and 160 malicious bitstreams for each of the three FPGA boards. To automate the bitstream generation process, we execute a TCL script within the Vivado Design Suite.

B. Security Threats from Stealthy Bitstreams

An FPGA configuration bitstream consists of a sequence of contiguous frames [21]. Each frame encapsulates a set of LUTs and other functional blocks within the FPGA and corresponds to a specific portion of the FPGA fabric. In other words, the bitstream configuration data is directly correlated to the frames it configures on the FPGA. Therefore, if RO circuits are intentionally dispersed across various LUTs, their patterns in the resulting bitstream may not be contiguous. Sequentially placed frames typically correspond to a consistent mapping of configuration data on the bitstream [21]. By distributing ROs across various frames, the attacker disrupts this sequential alignment. ML-based detection methods that learn malicious patterns from contiguous bitstream data may not be capable of detecting these ROs as they are no longer in a recognizable sequence within the bitstream [3] [5].



(a) (b) (c)

Fig. 2: Pynq FPGA floorplan for: (a) Normal RO circuits and (b) Stealthy RO circuits, (c) Experimental setup (yellow circle indicates that the the stealthy RO circuit is successfully programmed on the Pynq FPGA).

TABLE II: Power consumption of implemented RO circuits.

Type of RO circuit	% LUTs used	Power (W)
Normal	5.6	11.087
	15	49.277
Stealthy	5.6	13.523
	15	49.765

In order to evaluate the security threat posed by stealthy bitstreams, we assess RO power consumption in two scenarios:

- Normal RO circuits: We implement multiple RO circuits, with the inverters placed and routed in a single slice.
- Stealthy RO circuits: In this scenario, we consider RO circuits with inverters dispersed across multiple LUT slices, thus creating non-sequential configurations.

We computed the power consumption of the above two types of circuits on a Pynq FPGA board, which consists of 53200 LUT elements. We use the *'set_property'* metric within the Vivado design suite to define the LUT coordinates for manually placing the inverters across different LUT regions. These constraints about the RO placements are automatically generated and stored in a .xdc (Xilinx Design Constraints) file, which we subsequently use to perform synthesis and implementation of the RO circuits. Fig. 2 illustrates the placed and routed netlists corresponding to normal and stealthy RO circuits, each occupying 5.6% LUTs on the Pynq FPGA. The power consumption is listed in Table II. The number of ROs is chosen carefully so that they are sufficient to cause overheating and DoS on the FPGA [6]. We observe that stealthy RO circuits consume similar or significantly larger power compared to normal RO circuits, even when they are dispersed across multiple LUTs. When ROs occupy around 15% of LUTs, we obtain a warning that the FPGA junction temperature has exceeded the safe threshold. For the Pynq FPGA, the threshold thermal margin is 43.3 W; a value beyond that leads to overheating, risking FPGA functionality.

V. Proposed 3-Tier Detection Framework

A. GCN-Based Spatial Analysis of FPGA Bitstreams

A GCN is a semi-supervised ML model that operates on graph-structured data. GCNs are widely used in applications such as node classification, graph classification, and latent space clustering [22] [23]. A graph consists of a set of nodes and edges. A GCN aggregates feature information from adjacent nodes and subsequently generates node embeddings. These embeddings represent information about the nodes and their spatial relations. An FPGA bitstream can be represented as an undirected graph G, where the nodes correspond to bitstream windows and edges signify spatial relationships between these windows. Given the ability of GCNs to learn spatial dependencies inherent in graph-structured data, they offer a promising solution to address the problem of detecting distributed malicious signatures in a bitstream [23].

The mapping of bits to LUTs, flip-flops and logic blocks is stored in a sequential manner within the bitstream [20] [21]. Therefore, we partition a bitstream into windows for capturing spatial relationships as well as structural features that are indicative of malicious circuits. Splitting a bitstream into non-overlapping windows has the following advantages:

- **Feature extraction from graphs:** Each window contains a distinct portion of the bitstream, allowing for a focused extraction of the signatures within that window.
- **Avoiding redundancy:** Overlapping windows might introduce ambiguity at the boundary between two consecutive segments, making it difficult to capture the graph relationships. In contrast, non-overlapping windows maintain independent segment information, reducing redundant features.

A GCN consists of several convolutional layers; at each layer, features from adjacent nodes are aggregated and used to update the representation of each node. Therefore, the GCN learns to incorporate both local and global patterns from bitstream data, enriching the representation of each node in the graph. The aggregated node feature matrix H^l at a particular convolutional layer l is denoted as: $H^l = \sigma(D^{\frac{-1}{2}} \cdot A \cdot D^{\frac{-1}{2}} \cdot H^{l-1} \cdot W^{l-1})$, where H^{l-1} is the node feature matrix at $(l-1)^{th}$ layer, A is the adjacency matrix, D is the diagonal matrix of node degrees in the graph, W^{l-1} is the weight matrix at layer $l-1$, and σ is the non-linear activation function that is applied to the node feature aggregation operation. From the above equation, we observe that choosing the number of convolutional layers in a GCN model is pivotal in leveraging spatial relationships among bitstream data to identify malicious circuits.

B. Graph and Feature Extraction from Bitstream Data

Adjacency matrix generation For a bitstream split into ψ windows, the resultant adjacency matrix A will have dimensions $\psi \times \psi$. We use the structural similarity index (SSIM) metric for generating A. SSIM is chosen as it compares images based on multiple features of similarity such as luminance, contrast, and structure. This facilitates the construction of a meaningful graph structure that captures spatial similarities within the bitstream windows and subsequently aids the GCN model in identifying malicious signatures. We convert each bitstream window W_i (stored as Numpy file) into an image representation Im_i. For an image pair $(Im_i, Im_j), 1 \leq i, j \leq \psi, i \neq j$, the SSIM value [24] is calculated as:

$$\text{SSIM}(Im_i, Im_j) = \frac{(2\mu_{Im_i}\mu_{Im_j} + (k_1 l)^2)(2\sigma_{Im_i,Im_j} + (k_2 l)^2)}{(\mu^2_{Im_i} + \mu^2_{Im_j} + k_1 l)(\sigma^2_{Im_i} + \sigma^2_{Im_j} + k_2 l)},$$

where $\mu_{Im_i}(\mu_{Im_j})$ indicates the mean pixel value of $Im_i(Im_j)$, $\sigma^2_{Im_i}(\sigma^2_{Im_j})$ is the variance of $Im_i(Im_j)$, σ_{Im_i,Im_j} is the covariance of the image pair, and l is the range of the pixel values i.e., 255. The default values for k_1 and k_2 are 0.01 and 0.03, respectively. The range of SSIM is [0, 1], where 1 indicates a high similarity and 0 indicates no similarity. If $\text{SSIM}(Im_i, Im_j)$ is greater than a pre-defined threshold α_{thres}, $0 \leq \alpha_{thres} \leq 1$, we set $A_{ij} = 1$ (indicating an edge), else we set $A_{ij} = 0$ (indicating no edge), where A_{ij} indicates the presence or absence of edges between windows W_i and W_j. Fig. 3 illustrates the graph generation procedure.

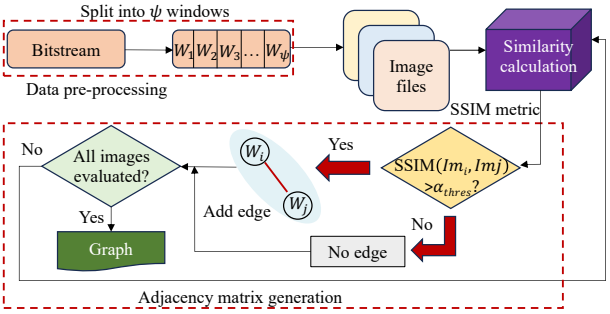


Fig. 3: Bitstream-to-graph conversion before GCN evaluation.

Node feature matrix generation We split the FPGA bitstream into ψ non-overlapping windows. For a VU440 FPGA bitstream, the size of each window is $\frac{128966372}{\psi}$. The choice of ψ depends on the specific FPGA bitstream and is obtained by hyperparameter tuning as described below.

FPGA bitstreams comprises numerous features (in the order of 10^8 for a VU440 bitstream), which is a challenge for traditional ML-based classification algorithms. However, Support Vector Machine (SVM) models are particularly useful in handling high-dimensional datasets, especially FPGA bitstreams [25]. We partition each of the benign and malicious bitstreams into ψ non-overlapping windows. Each set of ψ benign and malicious windows is trained on ψ identical SVM classifiers, following the procedure outlined in [2]. We use the average training accuracy obtained from the ψ SVM classifiers in determining the optimum value of ψ. Fig. 4 shows the best choice of ψ for the evaluated FPGA boards.

The high-dimensional windows can be challenging for direct usage as feature matrices to the GCN model; they can lead to increased training time and reduced computational efficiency [26]. Hence, we utilize convolutional neural network (CNN) layers to reduce the dimensionality of the windows while capturing essential bitstream patterns required for subsequent evaluation by the GCN model. For the i^{th} bitstream window $W_i, 1 \leq i \leq \psi$, the reduced feature vector for the i^{th} window is $F_1^i = f(W_i)$, where f denotes the convolution and pooling transformations applied by the CNN model, namely CNN-1. Subsequently, the output of CNN-1 is the reduced feature

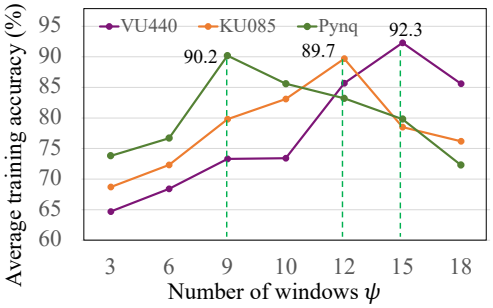


Fig. 4: Evaluating the best choice of ψ for different FPGA boards (marked by green dotted lines).

matrix $F_1 : \{F_1^i\}$, where the dimensionality of F_1 is $\psi \times k$ (k is the number of features obtained after reduction). We introduce another feature vector by performing Fast Fourier Transform (FFT) on each bitstream window. FFT captures frequency-domain characteristics, aiding in the identification of malicious patterns [4]. We obtain the FFT entropy value F_2^i for each window W_i using the *scipy.fft* function from Scikit library [27], and generate the feature matrix $F_2 : \{F_2^i\}$. The dimensionality of F_2 is $\psi \times 1$. Finally, we merge F_1 and F_2 to obtain the feature matrix F for the FPGA bitstream.

C. Stealthy Bitstream Identification using Graph Embedding

Fig. 5 shows the 3-tier pipeline for detection of stealthy FPGA bitstreams. The adjacency matrix A and the node feature matrix F are fed as inputs to the GCN model, namely GCN-2. For a graph having ψ nodes, we obtain ψ node embeddings after training GCN-2. The GCN-2 model generates node embeddings H_i^l for window W_i, $1 \leq i \leq \psi$, which capture the low-dimensional representations of each node in the graph based on its neighboring nodes. Next, we take an average of the ψ node embeddings to generate a single graph embedding for the bitstream, denoted by: $M^l = \frac{1}{\psi}\sum_{i=1}^{\psi} H_i^l$. We pass M^l through a multilayer perceptron (MLP) model, namely MLP-3. An MLP consists of an input layer and an output layer with one or multiple hidden layers; the neurons in each layer are fully connected. The output of MLP-3 passes through a series of activation functions, allowing the model to distinguish between benign and malicious graph embeddings.

The training loss for binary classification ('benign' or 'malicious') by MLP-3 is determined by comparing its predictions to the ground truth labels. This loss guides weight updates across CNN-1, GCN-2, and MLP-3 models during training. We run N_{epoch} iterations of training of the pipeline. The α_{thres} metric influences edge creation in the graph representation. We apply grid search to determine the α_{thres} value that yields the highest training accuracy of MLP-3 [28]; the results are shown in Fig. 6(a). This metric contributes to the generation of the adjacency matrix for a particular FPGA bitstream. It is subsequently used for generating graph embeddings via the GCN model. The optimum α_{thres} value remains fixed for a given family of FPGA bitstreams during inferencing.

Fig. 6(b) depicts the SSIM values for bitstream image pairs. The hyperparameters for the ML classifiers are shown in Table III, where I (O) indicates the input size (output feature size),

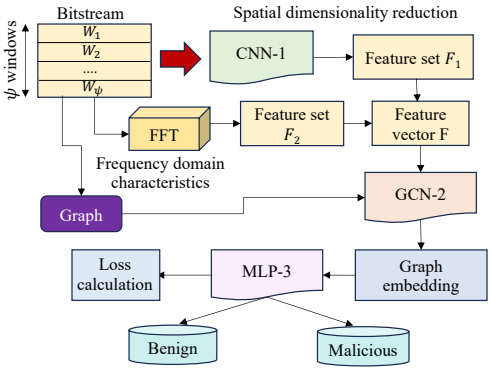


Fig. 5: Training and evaluation of the GCN-based framework. Note that, during training, the loss value of MLP-3 is back-propagated to MLP-3, GCN-2, and CNN-1 for end-to-end training of the classification framework.

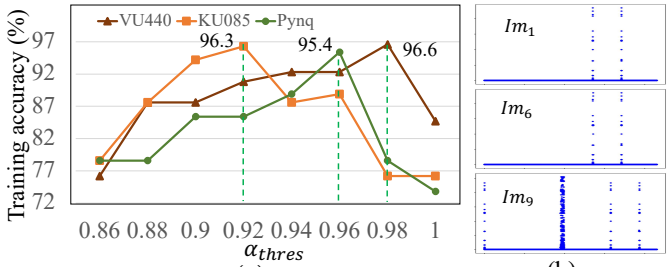


Fig. 6: (a) Evaluating best choice of α_{thres} for different FPGA boards based on MLP-3 training accuracy; for VU440 FPGA bitstream (α_{thres}: 0.98) implementing a single-stage self-clocked RO: (b) SSIM(Im_1, Im_6) = 0.99, which indicates a high similarity between Im_1 and Im_6; SSIM(Im_1, Im_9) = 0.93 indicates a lower similarity between Im_1 and Im_9.

L is the number of layers in forward propagation of the GCN model, H is the number of hidden layers, and lr is the learning rate. The loss function for MLP-3 is chosen as CrossEntropy.

VI. Results and Comparison with Prior Work

A. Experimental Setup

We generate the bitstreams using Xilinx Vivado 2018.2. We utilize the PyTorch framework for seamless integration of CNN, GCN, and MLP into a 3-tier malicious bitstream detection pipeline. We train the framework for 200 iterations using the best choice of hyperparameters (obtained by hyperparameter tuning) mentioned in Table III. Training and inferencing of the ML models are run on a 2.1 GHz Intel Xeon Gold 6230 CPU with 192 GB of RAM.

We split the training and test datasets in the ratio 70:30; this ensures a substantial number of samples for model training,

TABLE III: Best hyperparameters for the selected classifiers for different FPGA bitstreams.

Classifier	Selected hyper-parameters		
	VU440	KU085	Pynq
CNN-1	I: 64 × 2149438, O: 128	I: 64 × 753930, O: 128	I: 32 × 126424, O: 64
GCN-2	L: 6, I: 128, H: 32	L: 5, I: 128, H: 32	L: 6, I: 64, H: 16
MLP-3	H: 32, lr: 0.001	H: 32, lr: 0.01	H: 16, lr: 0.001

TABLE IV: Performance comparison of proposed framework with previous methods (baselines) for bitstream detection.

Type of FPGA board	Method	$T_m - T_s$		T_m		FPR (%)	t (s)
		TPR (%)	A_c (%)	TPR (%)	A_c (%)		
VU440	[2]	100	95.7	75	84	6.5	88.3
	[3]	88	85.9	68.75	76.5	15.2	242
	[5]	80	83	60.4	72.3	15.2	233
	Proposed	100	98.5	93.75	95.7	2.1	55.9
KU085	[2]	92	92.9	79.1	86.1	6.5	62.5
	[3]	80	85.9	60.4	74.4	10.8	191
	[5]	80	87.3	60.4	74.4	8.69	188
	Proposed	96	94.3	93.75	93.6	6.5	34.9
Pynq	[2]	96	92.9	66.6	78.7	8.69	21.8
	[3]	76	84.5	62.5	75.5	10.86	112
	[5]	80	83	64.5	74.4	15.2	113.2
	Proposed	96	95.7	95.8	95.7	4.34	11.62

TABLE V: Breakdown of time overhead for inferencing on FPGA bitstreams.

Operation	Time (s)		
	VU440	KU085	Pynq
Split bitstream into windows	46.2	27.4	9.5
FFT-based feature extraction	1.3	0.8	0.15
Feature reduction using CNN-1	5.4	4.9	1.5
Graph generation	0.3	0.27	0.07
Obtain node embeddings using GCN-2	2.7	1.6	0.4
Classification using MLP-3	$7e^{-4}$	$7e^{-4}$	$5.8e^{-4}$

without the risk of overfitting [29]. The training (test) dataset includes 106 (46) benign bitstreams and 112 (48) malicious bitstreams. Within the test dataset comprising malicious bitstreams, 23 bitstreams incorporate stealthy RO-based circuits.

B. Evaluation Metrics

We use the following metrics to evaluate the efficiency of the proposed 3-tier classification framework.

- True Positive Rate (TPR) is the percentage of malicious bitstreams that are correctly detected by the proposed method.
- False Positive Rate (FPR) is the percentage of benign bitstreams that are incorrectly classified as malicious.
- Classification accuracy (A_c) is computed as: $A_c = \frac{C_p}{T_p} \times 100\%$, where C_p is the number of correct predictions and T_p is the total number of predictions.
- Detection latency t (in seconds) is the time taken during model inferencing for the test bitstream.

C. Performance Evaluation of GCN-based Framework

Table IV presents a comparison of the proposed GCN-based framework with three baseline methods [2] [3] [5]. We perform two independent experiments on the malicious bitstreams included in the test dataset: (1) $T_m - T_s$, which excludes the set of stealthy bitstreams T_s from evaluation, and (2) T_m, which includes the full set of malicious bitstreams. We observe that baseline frameworks detect ROs when these ROs are placed in contiguous LUTs of the FPGA but are less effective in identifying stealthy ROs because they do not consider spatial relationships within the bitstream data during model training. In contrast, our method performs efficiently in both sets of experiments and detects stealthy bitstreams within a short duration (less than 1 minute of run time).

D. Time Overhead

We also computed the time overheads (in seconds) associated with evaluating a test bitstream across different FPGA boards using the GCN-based method; see Table V. Note that the FFT-based feature extraction, CNN-based feature reduction, and graph generation procedures are independent of each other and are therefore performed concurrently. Consequently, for these operations, the total time overhead is influenced by the operation with the highest time overhead. From Table IV and Table V, we see that the evaluation time for a bitstream is significantly lower than previous bitstream detection techniques [2] [3] [5] across all FPGA boards.

VII. Conclusion

We have presented a GCN-based framework to analyze spatial relationships among FPGA bitstream data and detect stealthy bitstreams. The proposed framework outperforms baseline detection methods in terms of classification accuracy across multiple FPGA boards.

References

[1] S. Choi and H. Yoo, "Fast logic function extraction of LUT from bitstream in Xilinx FPGA," *Electronics*, vol. 9, no. 7, p. 1132, 2020.

[2] J. Chaudhuri and K. Chakrabarty, "Diagnosis of malicious bitstreams in cloud computing FPGAs," *IEEE TCAD*, vol. 42, no. 11, 2023.

[3] ——, "Detection of malicious FPGA bitstreams using CNN-based learning," in *2022 IEEE European Test Symposium (ETS)*, 2022.

[4] ——, "Criticality analysis of ring oscillators in FPGA bitstreams*," in *2023 IEEE European Test Symposium (ETS)*, 2023.

[5] R. Elnaggar *et al.*, "Learning malicious circuits in FPGA bitstreams," *IEEE Trans. CAD*, 2022.

[6] D. Gnad *et al.*, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *FPL*, 2017, pp. 1–7.

[7] K. Matas *et al.*, "Power-hammering through glitch amplification – attacks and mitigation," in *FCCM*, 2020, pp. 65–69.

[8] T. Sugawara *et al.*, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, 2019.

[9] G. Provelengios *et al.*, "Power wasting circuits for cloud FPGA attacks," in *FPL*, 2020.

[10] T. La *et al.*, "Denial-of-service on FPGA-based cloud infrastructures—attack and defense," *IACR TCHES*, 2021.

[11] D. Gnad *et al.*, "Checking for electrical level security threats in bitstreams for multi-tenant FPGAs," in *FPT*, 2018, pp. 286–289.

[12] J. Krautter *et al.*, "Mitigating Electrical-level Attacks towards Secure Multi-Tenant FPGAs in the Cloud," *ACM TRETS*, vol. 12, no. 3, 2019.

[13] T. M. La *et al.*, "FPGADefender: Malicious self-oscillator scanning for Xilinx UltraScale + FPGAs," *ACM TRETS*, 2020.

[14] H. Nassar *et al.*, "LoopBreaker: Disabling interconnects to mitigate voltage-based attacks in multi-tenant FPGAs," in *ICCAD*, 2021.

[15] J. Yoon *et al.*, "A bitstream reverse engineering tool for FPGA hardware trojan detection," in *ACM SIGSAC*, 2018.

[16] H. S. Choo *et al.*, "Machine-learning-based multiple abstraction-level detection of hardware trojan inserted at register-transfer level," in *ATS*, 2019, pp. 98–980.

[17] J. Yang *et al.*, "Hardware trojans detection through RTL features extraction and machine learning," in *AsianHOST*, 2021, pp. 1–4.

[18] K. Hasegawa *et al.*, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in *ISCAS*, 2017, pp. 1–4.

[19] J. Krautter, D. Gnad, and M. Tahoori, "FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," 2018.

[20] Xilinx, "Ultrascale architecture configuration," https://bit.ly/3yyxvQ9 .

[21] J.-B. Note and É. Rannaud, "From the bitstream to the netlist," in *Proc. of ACM/SIGDA FPGA*, 2008, pp. 264–264.

[22] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[23] ——, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[24] Z. Wang *et al.*, "Multiscale structural similarity for image quality assessment," in *ACSSC*, 2003.

[25] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *ECML*. Springer, 1998.

[26] H. Gao *et al.*, "Large-scale learnable graph convolutional networks," in *ACM SIGKDD*, 2018.

[27] SKlearn, "Fourier transforms," bit.ly/3hco841.

[28] ——, "Grid search with cross validation," https://bit.ly/3hEHNnQ .

[29] Scikit-learn, "Machine learning in Python," https://bit.ly/3OzdLBZ .