cryptography

MDPI

Article

Natural Language Processing for Hardware Security: Case of Hardware Trojan Detection in FPGAs

Jaya Dofe [1,*], Wafi Danesh [2], Vaishnavi More [1] and Aaditya Chaudhari [1]

[1] Department of Electrical and Computer Engineering, California State University, Fullerton, CA 92831, USA; vaishnavimore@csu.fullerton.edu (V.M.); achaudhari@fullerton.edu (A.C.)
[2] Division of Engineering Program, The State University of New York, New Paltz, NY 12561, USA; daneshw@newpaltz.edu
* Correspondence: jdofe@fullerton.edu

Abstract: Field-programmable gate arrays (FPGAs) offer the inherent ability to reconfigure at runtime, making them ideal for applications such as data centers, cloud computing, and edge computing. This reconfiguration, often achieved through remote access, enables efficient resource utilization but also introduces critical security vulnerabilities. An adversary could exploit this access to insert a dormant hardware trojan (HT) into the configuration bitstream, bypassing conventional security and verification measures. To address this security threat, we propose a supervised learning approach using deep recurrent neural networks (RNNs) for HT detection within FPGA configuration bitstreams. We explore two RNN architectures: basic RNN and long short-term memory (LSTM) networks. Our proposed method analyzes bitstream patterns, to identify anomalies indicative of malicious modifications. We evaluated the effectiveness on ISCAS 85 benchmark circuits of varying sizes and topologies, implemented on a Xilinx Artix-7 FPGA. The experimental results revealed that the basic RNN model showed lower accuracy in identifying HT-compromised bitstreams for most circuits. In contrast, the LSTM model achieved a significantly higher average accuracy of 93.5%. These results demonstrate that the LSTM model is more successful for HT detection in FPGA bitstreams. This research paves the way for using RNN architectures for HT detection in FPGAs, eliminating the need for time-consuming and resource-intensive reverse engineering or performance-degrading bitstream conversions.

Keywords: hardware trojan; bitstream analysis; deep learning; recurrent neural network; long short-term memory; supervised learning

check for updates

Citation: Dofe, J.; Danesh, W.; More, V.; Chaudhari, A. Natural Language Processing for Hardware Security: Case of Hardware Trojan Detection in FPGAs. *Cryptography* **2024**, *8*, 36. https://doi.org/10.3390/cryptography8030036

Academic Editor: Jim Plusquellic

Received: 24 May 2024
Revised: 25 July 2024
Accepted: 6 August 2024
Published: 8 August 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

A field-programmable gate array (FPGA) is an integrated circuit (IC) with a post-fabrication hardware programming ability to implement custom functionality on a dedicated hardware platform. Due to the post-fabrication nature of the programming, an FPGA can provide economically feasible runtime processing, which has led to its increased utilization in various sectors, including automotive electronics, medical devices, the military, and consumer electronics [1,2]. Recently, FPGAs have found a place in large-scale cloud data centers [3,4] and flexible system-on-chips (SoCs) [5,6]. FPGAs, considered semi-custom circuits within the application-specific integrated circuit (ASIC) domain, offer distinct advantages over other programmable chips. FPGAs present benefits, such as shorter time-to-market and reduced non-recurring engineering costs [7]. They are dynamically configured at runtime using a configuration bitstream, which contains information on how to configure the logic resources on the FPGA layout for specific circuit implementations [8]. An increasing number of the top industry names in the tech sector have been using FPGAs in their data center and cloud operations, such as Amazon, Alibaba, and Huawei [9,10]. Furthermore, FPGAs are also being used extensively in emerging internet of things (IoT) applications [2,11].

In comparison to ASICs, designs implemented using FPGA can be modified in real time after deployment in the field. Such capability for post-deployment reconfiguration makes FPGAs more robust against tampering attacks, such as hardware trojan (HT) insertion in silicon, by untrusted foundries, since the attacker has no a priori knowledge of the final design to be implemented. However, in current application scenarios of FPGAs, configuration bitstream security has become a critical concern, particularly in applications where the confidentiality and integrity of the configuration bitstream is paramount. Remote access to FPGAs has created a scenario where there is unauthorized access to or tampering with the bitstream, leading to severe consequences, such as intellectual property theft [12,13], reverse engineering [14], and even malicious alterations of the FPGA's functionality [15–17]. Malicious bitstreams can enable remote monitoring, launch voltage and timing attacks, overheat the FPGA, leak sensitive information via side channels, and initiate denial of service attacks [18]. Henceforth, the security vulnerability of FPGA bitstreams coupled with the increasing market share of FPGAs is a growing incentive for many individuals with malicious intentions.

Many countermeasures for HT detection in FPGAs have been proposed, and significant research focus has been relegated to incorporating machine learning (ML) and deep learning (DL) approaches. In the current work, we propose an HT detection approach that uses a deep recurrent neural network (RNN)-based supervised learning method to detect the presence of HT in configuration bitstreams. Our method analyzes the configuration bitstreams for malicious patterns that indicate the presence of HT. In addition, the proposed method (1) foregoes time-consuming reverse engineering and reconstruction of the bitstream; (2) does not require HT activation, and (3) prevents data loss by bypassing the need for conversion of the bitstream to any intermediate data format. The main contributions of our work are as follows:

- We analyze bitstream architecture and formats, to determine the most effective data preprocessing techniques for RNN analysis.
- We utilize dynamic partial reconfiguration with various optimization subroutines, to create a comprehensive data set containing different circuit profiles of benchmark circuits.
- We develop a special preprocessing algorithm that transforms configuration bitstreams into a format suitable for processing by an RNN.
- We demonstrate the effectiveness of using RNN-based supervised learning methods on preprocessed configuration bitstreams, to identify malicious patterns indicative of HTs.

To the best of our knowledge, this is the first study to utilize deep learning, specifically basic RNN and LSTM networks, for detecting HT from configuration bitstreams.

The paper is structured as follows: Section 2 establishes the foundation by defining hardware trojans, bitstream architecture, and the specifics of the RNN networks employed. We then delve into related research on machine learning for HT detection in general (Section 3), with a particular focus on methods targeting FPGA bitstreams. Section 4 details the methodology of our proposed approach, outlining the data generation process and the preprocessing steps we implement to prepare the data for RNN analysis. The effectiveness of our method is then demonstrated through our experimental results, presented in Section 5. Finally, Section 6 concludes our work.

2. Preliminaries

2.1. Hardware Trojan

Hardware trojan attacks involve injecting malicious circuitry into integrated circuits (ICs) during any stage of the manufacturing process or supply chain [19]. These trojans remain dormant until they are triggered, at which point they can carry out various malicious activities, such as altering functionalities, leaking sensitive data, or completely disrupting device operation [1]. The challenge of detecting these trojans is compounded by their infrequent activation, and the potential consequences they pose, especially to critical systems, are a significant threat to hardware security and integrity.

The growing use of FPGAs in applications that are critical for security has intensified concerns about their susceptibility to hardware trojans. Similar to application-specific integrated circuits (ASICs), FPGAs are vulnerable to malicious modifications throughout their design and fabrication lifecycle [13]. Attackers have various avenues to introduce trojans into FPGAs, including manipulating the hardware description language (HDL) code, altering the FPGA fabric, modifying physical parameters, tampering with bitstreams, or exploiting vulnerabilities in FPGA computer-aided design (CAD) tools. For example, a trojan circuit could be stealthily embedded to monitor the FPGA's internal nodes, logic modules, and look-up tables. Upon activation, this trojan could cause malfunctions by altering LUT values, disrupting configuration cells to induce routing errors, or writing erroneous data into block random access memory [20].

2.2. Bitstream Architecture

The bitstream is a binary file that contains the configuration information to program the internal logic of the FPGA. In essence, a bitstream can be considered analogous to a stream of bytes arranged in layers that resemble a protocol stack for computer networking [21]. In this work, we focus on Xilinx 7-series devices, and the corresponding layers in the bitstream protocol stack are *physical interface*, *configuration packets*, and *configuration memory frames*, as shown in Figure 1.

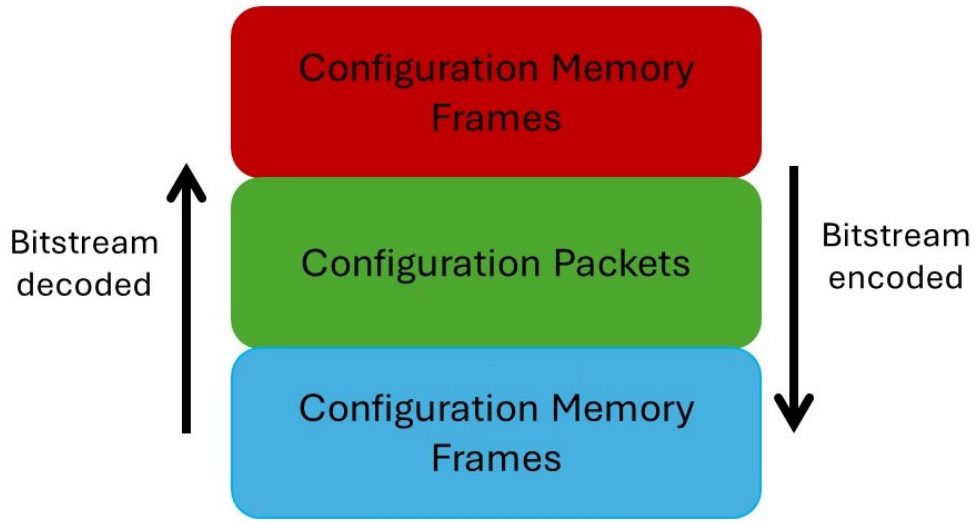

Figure 1. Concept of a bitstream protocol stack for Xilinx 7 series FPGAs.

Moving up the analogous bitstream protocol stack gives successively more information about the configuration of the physical resources on the FPGA. Starting at the bottom layer of the bitstream protocol stack, the physical interface indicates which configuration interface—such as JTAG, SPI (serial peripheral interface), BPI (byte peripheral interface), and SelectMAP—is used to program the FPGA. As a result of multiple configuration interfaces being available for programming the FPGA, the generated bitstream can have multiple file formats, as shown in Figure 2. In the current work, the bitstream was generated in the *bit* file extension.

The bitstreams generated for Xilinx 7-series FPGAs consist of instructions for modifying the configuration logic along with configuration data. In particular, for Xilinx 7-series FPGAs, a bitstream consists of three sections [22]:

- bus width auto-detection;
- sync word;
- FPGA configuration.

File Extension	Bit Swapping[(1)]	Xilinx Tool[(2)]	Description
BIT	Not Bit Swapped	ISE BitGen or Vivado write_bitstream (generated by default)	Binary configuration data file containing header information that does not need to be downloaded to the FPGA. Used to program devices from the iMPACT tool or Vivado device programmer with a programming cable.
RBT	Not Bit Swapped	ISE BitGen generated if `-b` option is set) or Vivado write_bitstream (generated with -raw_bitfile argument)	ASCII equivalent of the BIT file containing a text header and ASCII `1`s and `0`s. (Eight bits per configuration bit.)
BIN	Not Bit Swapped	ISE BitGen (generated if `-g binary:yes` option is set) or PROMGen, or Vivado write_bitstream (generated with -bin_file argument)	Binary configuration data file with no header information. Can be used for custom configuration solutions (for example, microprocessors), or in some cases to program third-party PROMs.
MCS	Bit Swapped[(3)]	ISE PROMGen or iMPACT, or Vivado write_cfgmem -format MCS	ASCII PROM file format containing address and checksum information in addition to configuration data. Used mainly for device programmers and the iMPACT tool.
HEX	Determined by User	ISE PROMGen or iMPACT, or Vivado write_cfgmem -format HEX	ASCII PROM file format containing only configuration data. Used mainly in custom configuration solutions.

Figure 2. Xilinx configuration file formats. (Note: For (1), (2), (3) please refer to [22]).

As a result of the multiple physical interfaces present for FPGA programming, it gives rise to both parallel and serial modes of configuration. The *bus width auto-detection* pattern, incorporated at the beginning of every bitstream, is used to set the configuration bus width. Detection of the *bus width auto-detection* pattern is ignored for serial configuration modes. After this pattern detection is complete, a *sync word* is used to align the configuration logic at a 32-bit word boundary, and it signals the start of the packet processing. For parallel configuration modes, the bus width detection must occur before the *sync word* is detected, which for Xilinx 7-series FPGAs is *0xAA995566* [22].

The *physical interface* layer, henceforth, can be considered to pass a sequence of packets containing a series of read/write operations for the configuration logic of the FPGA. In Xilinx 7-series FPGA devices, the sequence of packets go through the FPGA configuration process shown in Figure 3. From Figure 3, it can be observed that the steps in the configuration process can be grouped into two stages: *setup* and *bitstream loading* [22].

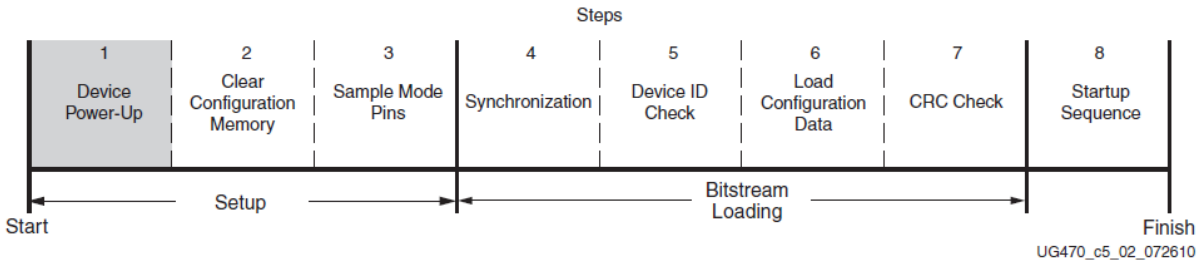

Figure 3. Steps in 7-series FPGA configuration.

In the *setup* stage, the device first powers up and provides power to different pins corresponding to various physical interfaces, it sequentially clears the configuration memory, and it samples the mode pins on the rising edge of the configuration clock, to determine if the serial mode or the parallel mode of configuration has been activated [22].

After the *setup* stage, the *bitstream loading* stage commences, and it is similar for both the serial and the parallel configuration modes. In the first step of *bitstream loading*, synchronization occurs by means of a special 32-bit synchronization word (*0xAA995566*). The special synchronization word signals the device to align the configuration data with the internal configuration logic. For parallel configuration modes, such as BPI, SlaveSelectMAP, and Master SelectMAP, bus width auto-detection must occur before synchronization, whereas the bus width auto-detection pattern is ignored for serial configuration modes , as Slave Serial, Master Serial, SPI, and JTAG. The data on the input pins prior to the synchronization word are ignored except for bus width auto-detection. After synchronization is complete, a *device ID check* is carried out and must pass, before the configuration data frames can be loaded. The *device ID check* ensures that the bitstream loaded is for the correct device. In the following step, the configuration data frames are loaded. At the final step of the *bitstream loading* stage, a *cyclic redundancy check (CRC)* is performed on the configuration data packets. Xilinx 7-series devices perform a 32-bit CRC check, in order to detect errors in the transmission of the bitstream. If the CRC check fails, configuration is aborted. When the loading of the configuration frames is complete, the device enters the *startup sequence* as per the command contained in the bitstream [22].

It is to be noted that only after detection of the *sync* word does the configuration logic process each 4-byte data word on the interface as a configuration packet or component of a multiple-word configuration packet [22]. In the final layer of the bitstream protocol stack, the *configuration memory frames* are determined. For Xilinx 7-series FPGA devices, the configuration memory is organized into frames that are distributed as tiles across the physical layout of the device. A frame is the smallest addressable segment of the Xilinx 7-series FPGA, and all operations, therefore, modify whole configuration frames [22]. The bitstream is in a packetized form, from which the configuration memory frame data are to be extracted. As such, the bitstream consists of two packet types: *Type 1* and *Type 2*. Through these packets, all 7-series FPGA bitstream commands are executed by read/write operations on the configuration registers. Such operations focus on the overall programming of the FPGA. Regarding the packet types, *Type 1* packets, shown in Figure 4, are used for configuration register reads and writes [22]. The opcode related to *Type 1* packets, illustrated in Figure 5, specifies the intended operation, such as read, write, or other actions. *Type 2* packets, shown in Figure 6, follow a *Type 1* packet and are used to write to longer blocks. Among the configuration registers, the *frame address register (FAR)* and the *frame data register, input (FDRI)* register are used to configure frame data [22]. All 7-series Xilinx FPGAs are divided into two halves, top and bottom, with all the frames having a length of 3232 bits (which is 101 32-bit words) [22]. Using this arrangement, the FAR divides the FPGA device into five fields, as given in Figure 7: block type, top/bottom bit, row address, column address, and minor address. Using this division by the FAR, frame data can be written at a frame address specified in the FAR.

Header Type	Opcode	Register Address	Reserved	Word Count
[31:29]	[28:27]	[26:13]	[12:11]	[10:0]
001	xx	RRRRRRRRRxxxxx	RR	xxxxxxxxxxx

Notes:

1. "R" means the bit is not used and reserved for future use. The reserved bits should be written as 0s.

Figure 4. Type 1 packet header format for Xilinx 7-series FPGA.

OPCODE	Function
00	NOP
01	Read
10	Write
11	Reserved

Figure 5. Opcode for Type 1 packet header.

Header Type	Opcode	Word Count
[31:29]	[28:27]	[26:0]
010	XX	xxxxxxxxxxxxxxxxxxxxxxxxxx

Figure 6. Type 2 packet header format for Xilinx 7-series FPGA.

Address Type	Bit Index	Description
Block Type	[25:23]	Valid block types are CLB, I/O, CLK (000), block RAM content (001), and CFG_CLB (010). A normal bitstream does not include type 011.
Top/Bottom Bit	22	Select between top-half rows (0) and bottom-half rows (1).
Row Address	[21:17]	Selects the current row. The row addresses increment from center to top and then reset and increment from center to bottom.
Column Address	[16:7]	Selects a major column, such as a column of CLBs. Column addresses start at 0 on the left and increase to the right.
Minor Address	[6:0]	Selects a frame within a major column.

Figure 7. Frame address register description.

2.3. Recurrent Neural Networks (RNN)

Recurrent neural networks (RNNs) are a category of neural networks that can process sequential data, where the sequences can be of varying length. The term recurrent refers to the fact that previous outputs can be fed back into the inputs while maintaining hidden states. The architecture of a conventional RNN is shown in Figure 8 [23], where $x^{<t>}$ is the input vector, $a^{<t>}$ is the vector of the activation functions, and $y^{<t>}$ is the output vector.

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad (1)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \quad (2)$$

In each time step, the activation $a^{<t>}$ and output $y^{<t>}$ are computed by Equations (1) and (2), respectively. In the equations, W_{ax}, W_{aa}, W_{ya}, b_a, b_y are temporally shared coefficients and g_1, g_2 are the activation functions. The construction of the hidden layer for a conventional RNN is given in Figure 9, which grants it the ability to store historical information. However, for a conventional RNN it becomes increasingly challenging to retain information very far into the past, as the gradient can exponentially increase or decrease in relation to the number of layers. As a result, the conventional RNN encounters a problem known as the vanishing gradient. A modification to the RNN architecture, where the hidden layer is replaced by a cell, as shown in Figure 10, mitigates the issue of vanishing gradient. The resulting RNN, which uses the cell in Figure 10, is called long short-term memory (LSTM), and it manages to retain only the most important historical information for generating the output and discards the rest. Therefore, it avoids the possible explosion of the gradient encountered in a conventional RNN.

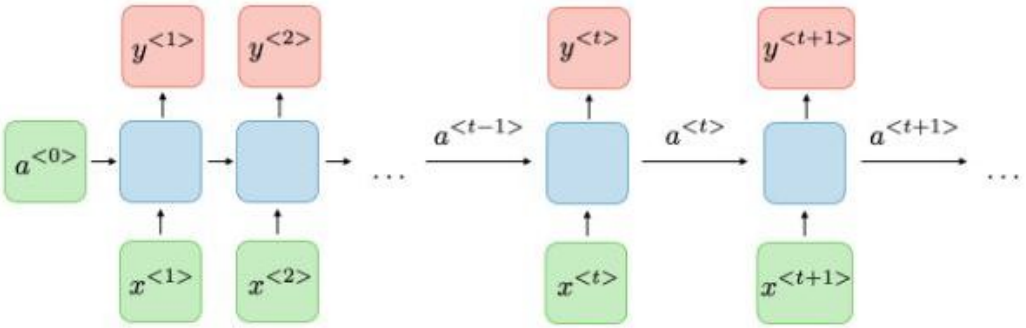

Figure 8. Conventional RNN architecture.

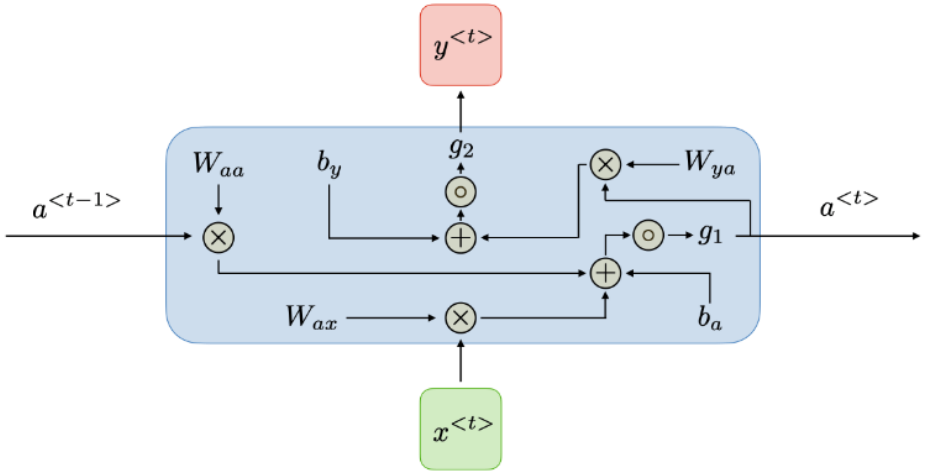

Figure 9. Hidden layer for conventional RNN.

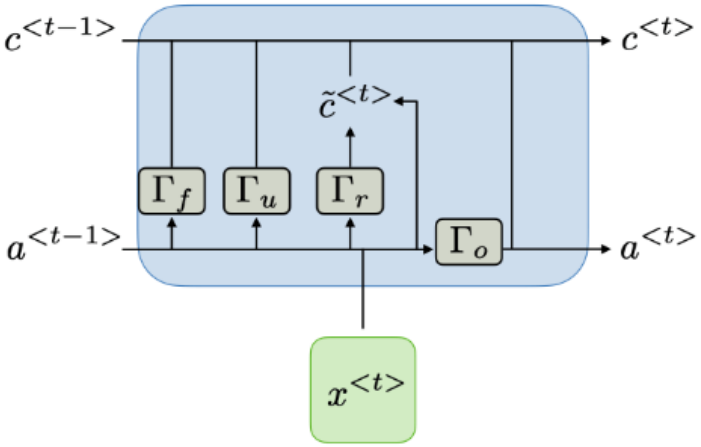

Figure 10. Cell for LSTM architecture.

3. Related Work

3.1. Machine Learning Techniques in Hardware Trojan Detection

HT detection techniques are typically used to verify the presence of unwanted components in ICs, primarily focusing on reverse engineering [24–26], analysis of circuit features [27], and side-channel analysis [28]. Machine learning-based approaches enhance this process by classifying or clustering IC-related features and improving HT detection accuracy. Reverse engineering necessitates repackaging ICs to obtain microscopic images of each layer. Although this method is destructive, it demonstrates high HT detection accuracy rates. In the studies [29,30], a classifier was created, using successive one-class

SVM and K-means techniques to distinguish between expected and suspicious structures in an IC autonomously. The rare triggered HTs are challenging to detect, using traditional testing methods. Hence, circuit features extracted from gate-level netlists, such as switching activity and net features, can be quantified and analyzed as key metrics for identifying suspicious nets or gates. Zhou et al. introduced a method for detecting sequential hardware trojans (HTs) in 3PIP cores, utilizing a structural feature-matching approach [31]. A functional and structural features learning-based method has been proposed [27] for untrusted hardware third party IPs. Integrating machine learning with side-channel analysis will address shortcomings and enhance the signal-to-noise ratio (SNR). Side-channel-based trojan detection techniques that utilize machine learning consider intrinsic circuit parameters like area, power, delay, and electromagnetic radiation, to predict the presence of trojans. Recent work on side-channel analysis has used support vector machines (SVMs) [26], K-nearest neighbors (K-NNs) [32], decision tree (DT) [33], naive Bayesian (NB) [32], deep learning (DL) [34], etc. Sumarsono and Masters [35] proposed a long short-term memory autoencoder (LSTM-AE) for hardware Trojan (HT) detection. While effective against always-on and condition-based Trojans, due to their ability to identify anomalous behavior over time, their method requires the HT to alter the circuit's behavior actively.

3.2. HT Detection in FPGA Bitstreams Using Machine Learning

The runtime reconfigurability of FPGAs provides for rapid prototyping during deployment. It has been a key feature for its usage in multiple emerging application domains, such as cloud computing, edge computing, internet of things (IoT) networks, and aerospace, among others. Despite the flexibility afforded by runtime reconfiguration, however, modern FPGAs are vulnerable to remote attacks [36–39] in such applications. Furthermore, adversaries are able to gain access to the configuration bitstream, since, in many FPGAs, the bitstream is stored in vulnerable external non-volatile memory. One of the critical hardware security vulnerabilities relating to bitstream access and modification by adversaries is the insertion of hardware trojans (HT). Several methods of HT insertion in FPGA bitstreams have been investigated in the literature. Chakraborty et al. [40] demonstrated an attack in which a ring oscillator HT was inserted into an unencrypted Virtex II bitstream, whereby the power consumption rapidly increased, leading to a higher operating temperature and consequent degradation of lifespan. A bitstream manipulation attack, where an HT was inserted to generate malicious ciphertexts to compromise the encryption algorithms AES, DES, and 3DES, was developed by Swierczynski et al. [41]. The inserted HT targeted the substitution box (S-box) used to obfuscate the relationship between the key and ciphertext. In [42], Swierczynski et al. demonstrated an HT insertion scheme wherein the encryption of a high-security USB flash drive, the Kingston DataTraveler 5000, was compromised. For the attack, specific instances of S-boxes used in the encryption were altered and replaced with an identity function, thereby converting the encryption scheme to a linear function.

4. Methodology

4.1. Data Set Generation

Due to the lack of standardized benchmark datasets for FPGAs, we had to create data tailored to our machine learning model, as there are diverse architectures across different FPGA families. We used dynamic partial reconfiguration to generate circuit profiles, incorporating various optimization techniques [43]. This approach maximized layout coverage and provided valuable training data for our model. The details are illustrated in Table 1.

To make our data set for HT profiles, we generated malicious bitstreams using combinational hardware trojans. We modified the design by adding additional logic gates to the non-critical path while keeping the size of the HT within 2–5% of the total circuit size. We followed the same procedure for generating malicious bitstreams as presented in Table 1 [1]. In our study, we used the *.bit* file format. In this study, we employed the entire bitstream based on the rationale outlined in [18]. It has been observed that certain FPGA

applications may not utilize partial reconfiguration, thereby retaining exclusive access to resources to accelerate the current application. In such cases, ensuring the integrity of the entire bitstream becomes imperative.

Table 1. Process flow for generating bitstreams using dynamic partial reconfiguration.

Steps	Description
1. Input Verilog and XDC file.	Provide the Verilog hardware description language (HDL) file describing the FPGA design and the XDC file containing timing constraints.
2. Synthesize design.	Convert the Verilog code into a netlist, a low-level representation of the circuit.
3. Enable dynamic function exchange.	Configure the FPGA to allow for partial reconfiguration.
4. Perform partial reconfiguration in selected floorplan.	Define the area of the FPGA where the reconfiguration will occur and implement the desired changes.
5. Generate full bitstream in .bit format.	Create a complete configuration file containing instructions for the entire FPGA, including both static and reconfigured areas.

4.2. Preprocessing FPGA Bitstream for RNN and LSTM Models

Our proposed method involves two main stages: preprocessing and applying DL algorithms (RNN and LSTM). Using a Xilinx 7-series FPGA (Artix-7) as a case study, we parse the bitstream, based on information from the Xilinx UG470 user guide, focusing only on sections relevant to the circuit configuration. For the current work, in order to generate the datasets for the RNN and LSTM models, multiple bitstreams were generated by means of dynamic partial reconfiguration. The bitstreams generated conformed to various optimization parameters and, therefore, were of varying lengths. As a result, the bitstreams required preprocessing before feeding them to the RNN and LSTM models. Among the multiple bitstream file formats available, the *.bit* file format, shown in Figure 11 is used, as the required preprocessing is easier to perform in this format. The preprocessing algorithm used in this work is given in Figure 12. As can be seen from Figure 12, the first step after reading in the bitstream is removing consecutive rows of zeros. For the circuits used in the current work, the corresponding bitstreams were sparse, since much of the configuration logic resources in the FPGA remained unused. Removing consecutive rows of zeros from the bitstream does not warp the positional information of the logic resources preceding and succeeding a specific consecutive row of zeros. Since the positional information of the logic resources is preserved, the removal of consecutive rows of zeros acts as a compression scheme for the bitstream.

After undergoing this compression, these bitstreams are converted to a hexadecimal value, thereby transforming the bitstream to an encoded sequence. Due to the various optimization parameters used in partial reconfiguration, the encoded bitstream sequences are of varying lengths. Zero padding is, henceforth, used to ensure the uniform size of the encoded bitstream sequences. Such zero padding is achieved by finding the largest encoded sequence length and using it to extend the size of the remaining bitstream sequences to that required sequence length. Both malicious (containing HT) and benign bitstreams are converted into encoded sequences. The preprocessing stage is adaptable to different FPGA bitstreams by referencing the appropriate manufacturer datasheets.

After preprocessing, the encoded bitstream sequences are fed into the RNN and LSTM models for supervised learning, as shown in Figure 13. The encoded bitstream sequences are split equally, to generate the training and validation sets, with each set containing both benign and malicious encoded sequences. The training set is labeled and used to

train RNN and LSTM models via supervised learning. In the current work, during the parameter selection stage before training, an additional step was carried out, which is referred to as *step size* selection. The *step size* can be considered analogous to a resolution. Each row of the bitstream, as shown in Figure 11, is 128 bits wide and a step is equivalent to a certain number of bits considered together. The more bits in a step, the lower the resolution and number of data points available for the RNN and LSTM models. Once training is complete, the trained RNN and LSTM models are validated, using the test set to obtain the performance metrics, such as accuracy, precision, recall, and F1 score.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000	00	09	0f	f0	0f	f0	0f	f0	0f	f0	00	00	01	61	00	2d
00000010	43	69	72	63	75	69	74	36	32	38	38	3b	55	73	65	72
00000020	49	44	3d	30	58	46	46	46	46	46	46	46	46	3b	56	65
00000030	72	73	69	6f	6e	3d	32	30	31	39	2e	31	00	62	00	0d
00000040	37	61	37	35	74	69	66	67	67	34	38	34	00	63	00	0b
00000050	32	30	32	32	2f	31	31	2f	31	34	00	64	00	09	31	36
00000060	3a	30	38	3a	32	31	00	65	00	3a	60	7c	ff	ff	ff	ff
00000070	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
00000080	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	00	00	00	bb
00000090	11	22	00	44	ff	ff	ff	ff	ff	ff	ff	ff	aa	99	55	66
000000a0	20	00	00	00	30	02	20	01	00	00	00	00	30	02	00	01
000000b0	00	00	00	00	30	00	80	01	00	00	00	00	20	00	00	00
000000c0	30	00	80	01	00	00	00	07	20	00	00	00	20	00	00	00

Figure 11. The .bit file format.

The FPGA configuration bitstream encodes detailed information about the circuit design, including both electrical and logical resources. Each logical component or segment of the circuit leaves a distinct footprint in the bitstream, which persists even when the bitstream is encrypted. As a result, any third-party IP, regardless of its encryption status, will imprint its footprint on the bitstream.

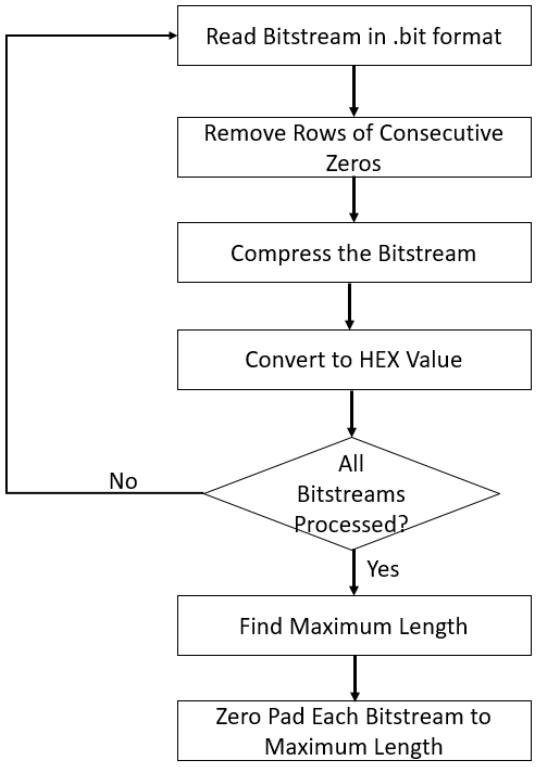

Figure 12. Data preprocessing algorithm.

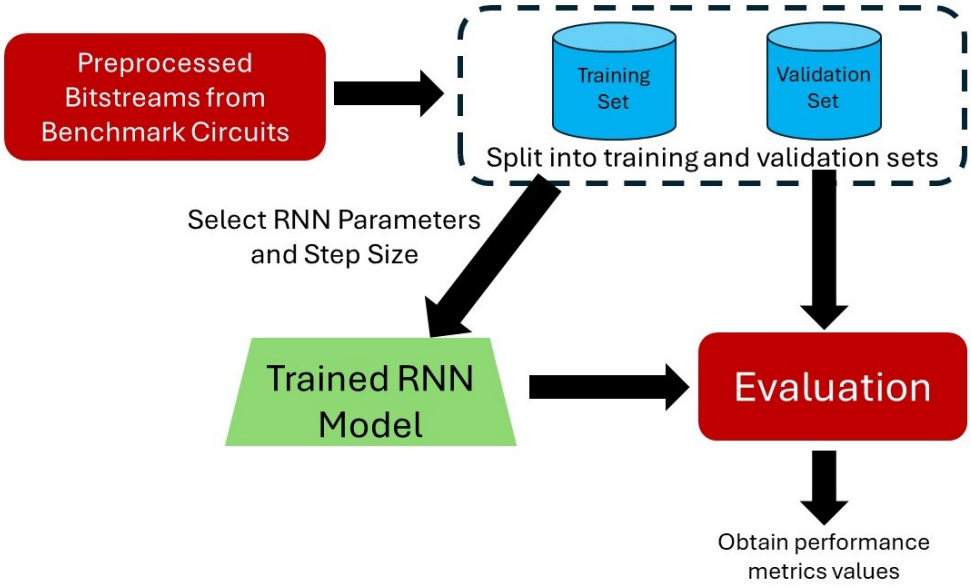

Figure 13. Training RNN models on preprocessed bitstreams.

5. Results

5.1. Combinational Trojan Detection

In this work, we used ISCAS85 benchmark circuits c432, c499, c1355, and c6288 to validate our proposed method. We generated 800 bitstreams for each circuit, half of which were benign, while the rest were malicious. Both RNN and LSTM were used for validation. For each of these models, step sizes of 8, 16, 32, and 64 were used. Both models were trained for 30 epochs, and the loss and accuracy for both training and validation is reported. The simulations were carried out in Python, using the TensorFlow package.

The basic RNN model used the following parameters:

- Two bidirectional SimpleRNN layers with 64 and 32 units, respectively, where the first layer processed input sequences bidirectionally, while the second layer processed output sequences bidirectionally.
- Dropout layers with a dropout rate of 0.5 after each SimpleRNN layer, to prevent overfitting.
- A dense output layer with a sigmoid activation function that facilitated binary classification.

Figure 14 shows the training and validation accuracy for step size 16 in the basic RNN. Tables 2–5 show the performance metrics for step sizes 8, 16, 32, and 64, respectively. The results for the RNN demonstrated only moderate efficacy in differentiating bitstreams that were compromised by HTs from those that were not. The performance trend for the RNN remained, on average, consistent. This was particularly evident at step size 64, when zeros in precision, F1-score, and recall indicated that the model could not generate meaningful results.

The LSTM model used the following parameters:

- A bidirectional LSTM layer with 64 units/nodes along with a dropout layer with a rate = 0.5;
- A bidirectional LSTM layer with 32 units/nodes along with a dropout layer with a rate = 0.5;
- A dense layer with 1 unit/node and a 'sigmoid' activation function.

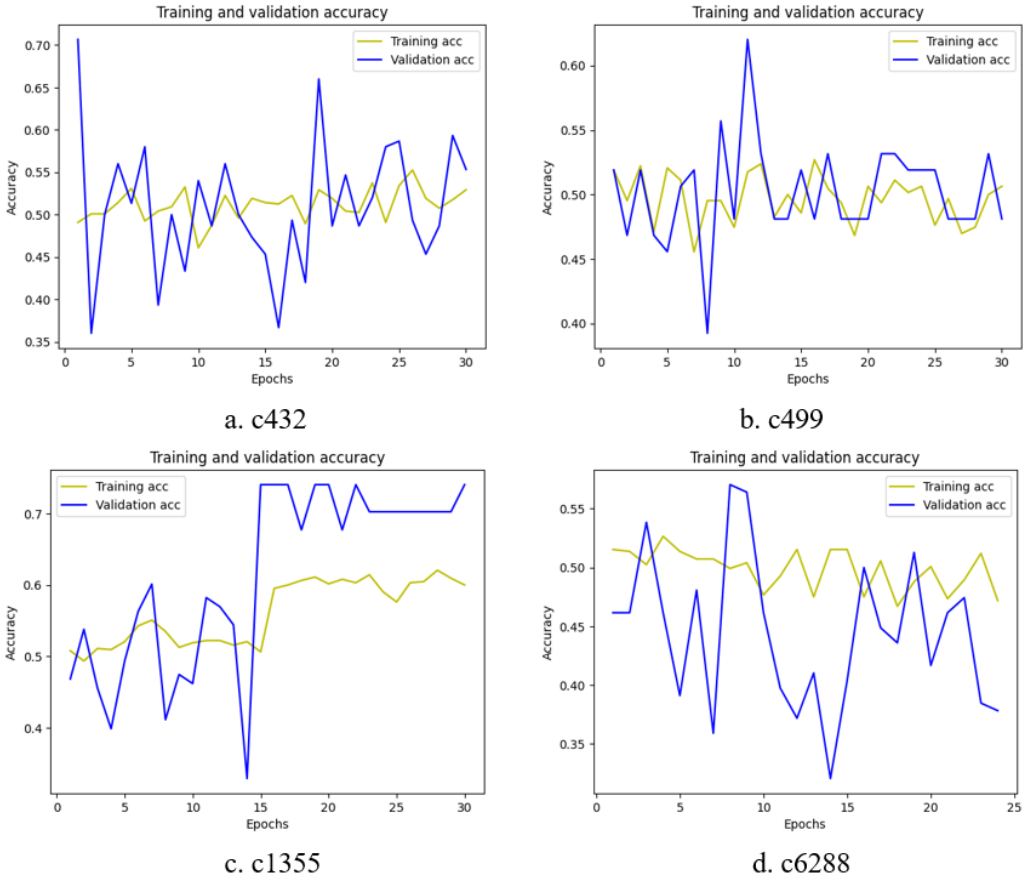

Figure 14. Training and validation accuracy over training epochs for RNN with step size 16.

As illustrated in Figure 15, varying the step size can significantly impact performance. Here, we see the training and validation accuracy achieved by an LSTM using a step size of 16. The results reveal the significant impact of the preprocessing step size on the LSTM model performance. From the results shown in Tables 2–5 and Figure 16, it is clear that the performance metrics deteriorated with increasing step size. Specifically, accuracy and loss function both improved as the step size increased, reaching an optimal step size at 16. However, performance degraded at step sizes of 32 and 64. The performance degradation observed at higher step sizes can be attributed to the reduced number of data points presented to the LSTM model. This reduction in resolution occurred because larger segments of the FPGA bitstream (as depicted in Figure 11) were incorporated into a single data point during preprocessing. As a result, underfitting could occur, which degraded the performance. Figure 16 captures the overall trend succinctly.

Table 2. Performance metrics for step size 8.

Benchmarks	Accuracy		Precision		Recall		F1Score	
	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM
c432	0.52	0.95	0.48	0.97	0.66	0.92	0.56	0.95
c499	0.49	0.88	1.00	0.81	0.74	0.98	0.85	0.88
c1355	0.64	0.98	0.71	0.96	0.49	1.0	0.58	0.98
c6288	0.44	0.93	0.00	0.96	0.00	0.92	0.00	0.94

Table 3. Performance metrics for step size 16.

Benchmark	Accuracy		Precision		Recall		F1Score	
	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM
c432	0.46	0.92	0.45	0.97	0.84	0.84	0.59	0.90
c499	0.49	0.93	0.49	0.97	1.0	0.88	0.66	0.93
c1355	0.60	0.93	0.61	0.93	0.60	0.93	0.61	0.93
c6288	0.41	0.96	0.39	0.94	0.20	0.91	0.26	0.92

Table 4. Performance metrics for step size 32.

Benchmark	Accuracy		Precision		Recall		F1Score	
	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM
c432	0.55	0.55	0.00	0.00	0.00	0.00	0.00	0.00
c499	0.56	0.93	0.54	0.97	0.60	0.88	0.57	0.93
c1355	0.49	0.91	0.00	0.93	0.00	0.89	0.00	0.91
c6288	0.44	0.82	0.00	0.94	0.00	0.80	0.00	0.70

Table 5. Performance metrics for step size 64.

Benchmark	Accuracy		Precision		Recall		F1Score	
	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM
c432	0.55	0.55	0.00	0.00	0.00	0.00	0.00	0.00
c499	0.51	0.51	0.00	0.00	0.00	0.00	0.00	0.00
c1355	0.49	0.49	0.00	0.00	0.00	0.00	0.00	0.00
c6288	0.47	0.47	0.00	0.00	0.00	0.00	0.00	0.00

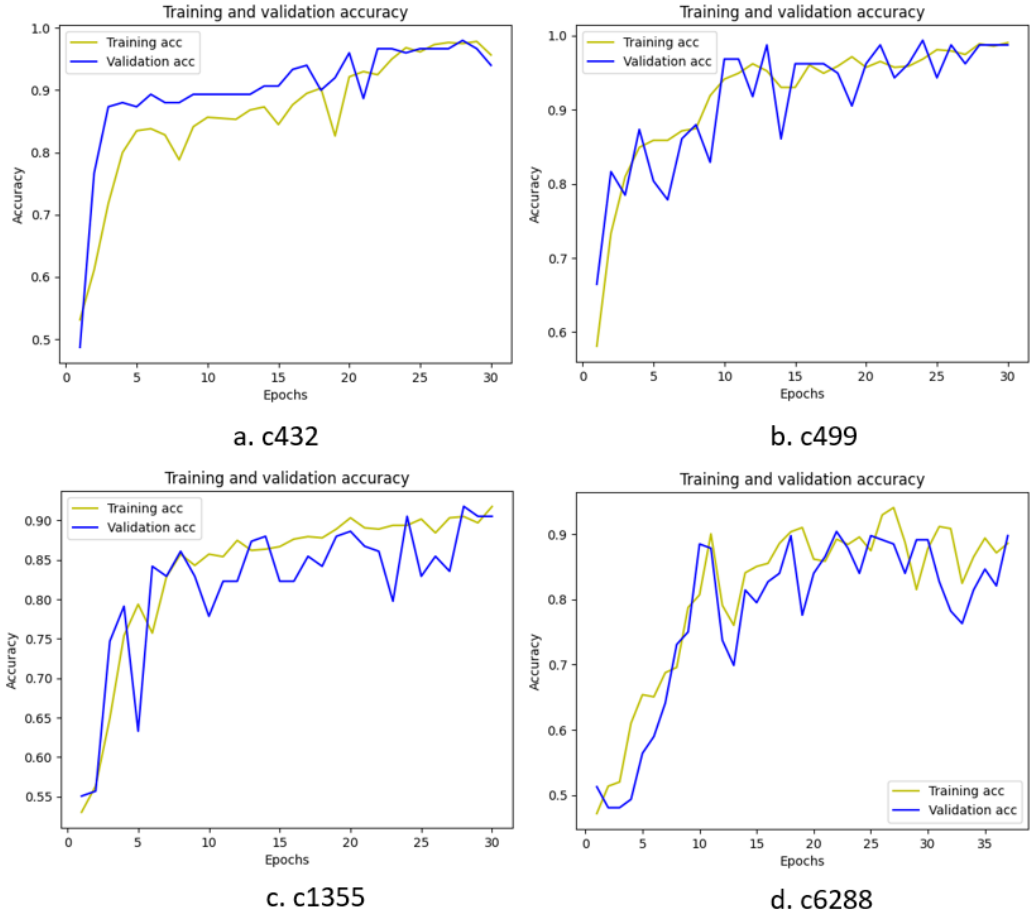

Figure 15. Training and validation accuracy over training epochs for LSTM with step size 16.

Our initial experimentation utilized an RNN for HT detection in FPGA bitstreams. However, RNNs exhibit possible overfitting tendencies, suggesting a potential mismatch between their architecture and the inherent long-range dependencies within the bitstream data. Consequently, we investigated an LSTM model, which was specifically designed to address vanishing gradients and effectively capture long-term dependencies. The LSTM networks achieved satisfactory results, demonstrating their suitability for HT detection within the range of the optimum step size. As shown in Figure 16, the overall accuracy for step size 8 and 16 was 93.5%:

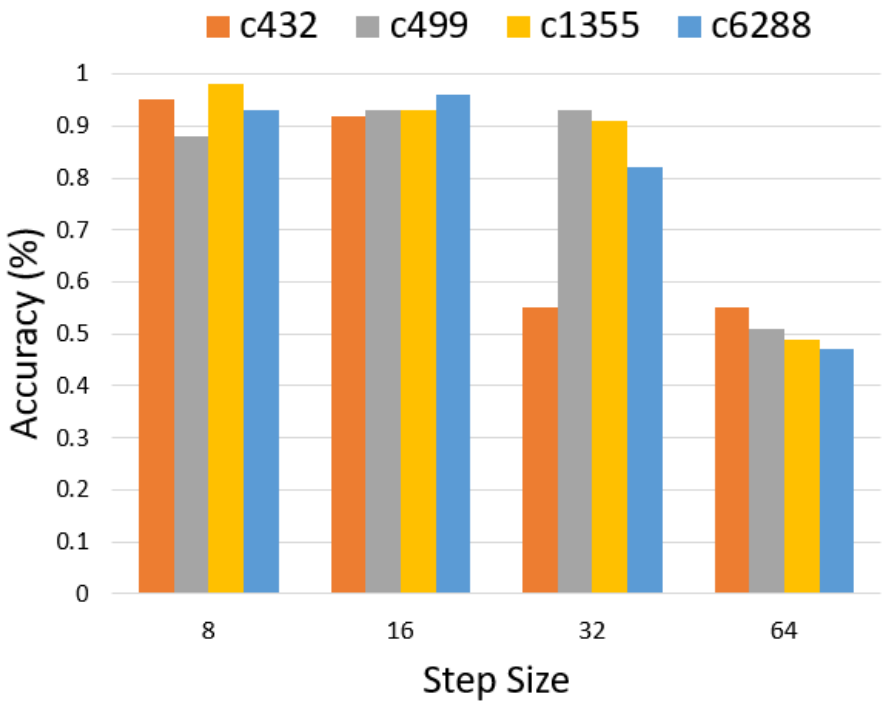

Figure 16. Step size and accuracy trends for LSTM.

5.2. Case Study: Multi-Tenant FPGA Ring Oscillator-Based Trojan

Multi-tenant FPGAs are gaining traction in cloud computing, due to their increased utilization and flexibility. However, this platform introduces security risks from malicious users in cloud environments as the power distribution network is connected between the victim and the adversary. Attackers can exploit voltage variations within the target FPGA by employing voltage sensors like ring oscillators (RO) situated at their end [44]. This vulnerability facilitates voltage-based attacks, including denial of service (DoS) [45]. Moreover, simultaneous activation of a network of ROs at a specific frequency can disrupt the on-board voltage regulator and cause the FPGA to malfunction [45,46]. A study by Zhao et al. highlights remote power side-channel attacks initiated from FPGA fabric [45]. This approach involves using an FPGA RO circuit to steal RSA keys from a CPU, posing a significant threat to multi-tenant cloud security. The use of non-combinational ROs to attack cloud FPGAs presents a significant security challenge, as these malicious circuits evade detection during design rule checks [47]. To address the detection of RO-based HTs, we conducted a case study on latched RO designs, to validate the effectiveness of our proposed method.

Figure 17 shows an example of a latched ring oscillator (RO) circuit. In this design, the output from the final RO stage is fed back as the input to a latch placed within the loop. A latch divides the circuit into two combinational loops, making the overall design non-combinational, which is challenging to detect.

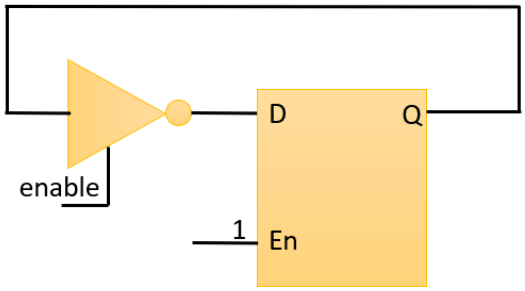

Figure 17. Example of latched RO.

Leveraging the same data collection process detailed in Section 4.1, we investigated HT detection for latched RO trojans. All the experimental parameters, including the number of benign and malicious bitstreams, the step size, and the training/test data split, were maintained as described in Section 5.1, for consistency. As shown in Figure 18, the training and validation accuracy curves for the latched RO HT exhibited a similar trend to the combinational HT discussed previously. The performance metrics for c17 is shown in Figure 19 for step sizes 8 and 16. Notably, performance degradation was observed beyond a step size of 16, potentially due to a reduction in resolution within the training data. This aligns with the underfitting trend observed for combinational HTs at larger step sizes (Section 5.1), suggesting that both HT types share a similar susceptibility to insufficient training data resolution.

In the current work, our focus was on establishing proof of concept and experimentally validating our proposed method. While we have not yet applied it to system-level configurations, our approach is readily scalable to that level.

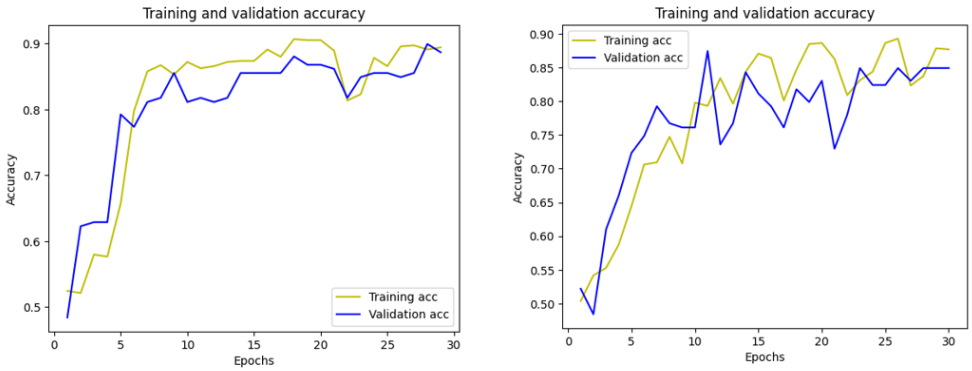

Figure 18. Training and validation accuracy vs. step size for c17 benchmark.

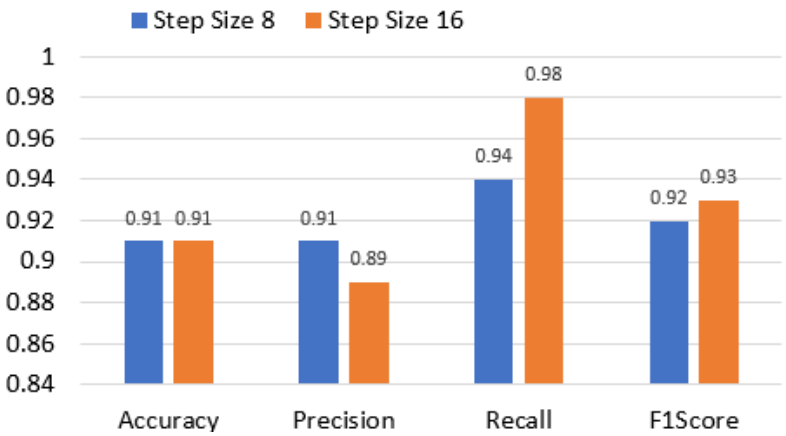

Figure 19. Comparison of c17 performance metrics for step sizes 8 and 16.

5.3. Comparison with Relevant Work

In Table 6, we compare our method with the recent relevant work described in [48]. It is notable that in that work the experiments were carried out on ASIC, while the current work focused on FPGA. The method in [48] relies on feature selection, specifically using circuit path information, which is a resource-intensive and time-consuming process. This approach is particularly challenging for low-power devices, which have limited processing capabilities and power resources. Our proposed method eliminates the need for feature selection by directly utilizing bitstreams. This simplification reduces computational overhead and conserves power. Additionally, in detecting combinational trojans, our method achieved accuracy of 93.50% compared to the 84.00% true positive rate (TPR) of the reference method. This improvement in accuracy translates to better detection rates.

Table 6. Comparison with relevant work.

Reference Article	Type of IC Used	Feature Selection Needed	Feature Selection	HT Type	Learning Model	Evaluation Metric
[48]	ASIC	✓	Circuit path information	Combinational	LSTM	TPR: 84.00%
Proposed Method	FPGA	*X*	Not needed	Combinational	LSTM	Accuracy: 93.50%
[48]	ASIC	✓	Circuit path information	Sequential	LSTM	TPR-99.00%
Proposed Method	FPGA	*X*	Not needed	Sequential	LSTM	Accuracy: 91.00%

One disadvantage of our proposed method is that it will not detect parametric trojans [49]. Parametric trojans manipulate the parameters of existing components, such as voltage levels, timing, or power consumption, rather than adding new logic or altering the circuit configuration. As these changes do not leave a distinct physical footprint in the bitstream, they remain undetectable by our current approach, which relies on identifying footprints in the bitstream to generate a behavioral profile.

6. Conclusions

Field programmable gate arrays (FPGAs) have found extensive usage in multiple application domains, due to their inherent run-time reconfiguration capabilities. However, reconfiguration is achieved more often with a trade-off in security—in particular, the security of the configuration bitstream. Hardware trojan insertions are prevalent threats to FPGA security, and one of the key ways for hardware trojan (HT) insertion is through bitstream manipulation. In this work, we propose a deep learning recurrent neural network (RNN)-based approach for HT dectection. Notably, our approach does not require HT activation, it can bypass extensive reverse engineering, and it does not encounter data loss in the transformation of bitstreams into an intermediate format. Basic RNN and LSTM models are used with varying resolutions. Our experimental results showed that the resolution has a direct correlation with the model's performance. Furthermore, the LSTM model outperforms the basic RNN model within the optimum range of resolution. The overall detection accuracy across different HT types averaged 93.5%, showcasing the effectiveness of our proposed approach.

In future work, we aim to develop a more optimized approach that will enable us to create a comprehensive database, including TrustHub benchmarks, to further validate and refine our method.

Author Contributions: Conceptualization, J.D. and W.D.; methodology, J.D. and W.D.; software, J.D., V.M. and A.C.; validation, J.D., W.D., V.M. and A.C.; formal analysis, J.D. and W.D.; resources, J.D., W.D., V.M. and A.C.; data curation, J.D., V.M. and A.C.; writing—original draft preparation, J.D. and W.D.; writing—review and editing, J.D. and W.D.; visualization, J.D.; supervision, J.D.; project administration, J.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Rajput, S.; Dofe, J.; Danesh, W. Automating Hardware Trojan Detection Using Unsupervised Learning: A Case Study of FPGA. In Proceedings of the 2023 24th International Symposium on Quality Electronic Design (ISQED), San Francisco, CA, USA, 5–7 April 2023 ; pp. 1–6. [CrossRef]
2. Elnawawy, M.; Farhan, A.; Nabulsi, A.A.; Al-Ali, A.; Sagahyroon, A. Role of FPGA in Internet of Things Applications. In Proceedings of the 2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), Ajman, United Arab Emirates, 10–12 December 2019; pp. 1–6. [CrossRef]
3. Fahmy, S.A.; Vipin, K.; Shreejith, S. Virtualized FPGA Accelerators for Efficient Cloud Computing. In Proceedings of the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, Canada, 30 November–3 December 2015; pp. 430–435. [CrossRef]
4. Zeitouni, S.; Vliegen, J.; Frassetto, T.; Koch, D.; Sadeghi, A.R.; Mentens, N. Trusted Configuration in Cloud FPGAs. In Proceedings of the 2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Orlando, FL, USA, 9–12 May 2021; pp. 233–241. [CrossRef]
5. Monmasson, E.; Hilairet, M.; Spagnuolo, G.; Cirstea, M.N. System-on-Chip FPGA Devices for Complex Electrical Energy Systems Control. *IEEE Ind. Electron. Mag.* **2022**, *16*, 53–64. [CrossRef]
6. Abdelfattah, M.S.; Bitar, A.; Betz, V. Design and Applications for Embedded Networks-on-Chip on FPGAs. *IEEE Trans. Comput.* **2017**, *66*, 1008–1021. [CrossRef]
7. Mattioli, M. FPGAs in Client Compute Hardware: Despite Certain Challenges, FPGAs Provide Security and Performance Benefits over ASICs. *Queue* **2022**, *19*, 66–88. [CrossRef]
8. Koch, D.; Ziener, D.; Hannig, F. FPGA Versus Software Programming: Why, When, and How? In *FPGAs for Software Programmers*; Koch, D., Hannig, F., Ziener, D., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 1–21. [CrossRef]
9. Deep Dive into Alibaba Cloud F3 FPGA as a Service Instances. 2018. Available online: https://www.alibabacloud.com/blog/deep-dive-into-alibaba-cloud-f3-fpga-as-a-service-instances_594057 (accessed on 22 March 2024).
10. Amazon EC2 F1 Instances. 2024. Available online: https://aws.amazon.com/ec2/instance-types/f1 (accessed on 20 March 2024).
11. Magyari, A.; Chen, Y. Review of state-of-the-art FPGA applications in IoT Networks. *Sensors* **2022**, *22*, 7496. [CrossRef]
12. Krautter, J.; Gnad, D.R.E.; Tahoori, M.B. Mitigating Electrical-Level Attacks towards Secure Multi-Tenant FPGAs in the Cloud. *ACM Trans. Reconfig. Technol. Syst.* **2019**, *12*, 1–26. [CrossRef]
13. Sunkavilli, S.; Zhang, Z.; Yu, Q. New Security Threats on FPGAs: From FPGA Design Tools Perspective. In Proceedings of the 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL, USA, 7–9 July 2021; pp. 278–283. [CrossRef]
14. Zhang, T.; Wang, J.; Guo, S.; Chen, Z. A Comprehensive FPGA Reverse Engineering Tool-Chain: From Bitstream to RTL Code. *IEEE Access* **2019**, *7*, 38379–38389. [CrossRef]
15. Trimberger, S.; McNeil, S. Security of FPGAs in data centers. In Proceedings of the 2017 IEEE 2nd International Verification and Security Workshop (IVSW), Rhodes Island, Greece, 3–5 July 2017; pp. 117–122. [CrossRef]
16. Cho, M.; Jang, J.; Seo, Y.; Jeong, S.; Chung, S.; Kwon, T. Towards Bidirectional LUT-level Detection of Hardware Trojans. *Comput. Secur.* **2021**, *104*, 102223. [CrossRef]
17. Krieg, C.; Wolf, C.; Jantsch, A. Malicious LUT: A stealthy FPGA Trojan injected and triggered by the design flow. In Proceedings of the 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, USA, 7–10 November 2016; pp. 1–8. [CrossRef]
18. Elnaggar, R.; Chaudhuri, J.; Karri, R.; Chakrabarty, K. Learning Malicious Circuits in FPGA Bitstreams. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2022**, *42*, 726–739. [CrossRef]
19. Bhunia, S.; Abramovici, M.; Agrawal, D.; Bradley, P.; Hsiao, M.S.; Plusquellic, J.; Tehranipoor, M. Protection Against Hardware Trojan Attacks: Towards a Comprehensive Solution. *IEEE Des. Test* **2013**, *30*, 6–17. [CrossRef]
20. Mal-Sarkar, S.; Krishna, A.; Ghosh, A.; Bhunia, S. Hardware trojan attacks in FPGA devices: Threat analysis and effective counter measures. In Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI, Houston, TX, USA, 21–23 May 2014. [CrossRef]
21. Altherr, R. Unpacking Xilinx 7-Series Bitstreams: Part 1. 2018. Available online: https://www.kc8apf.net/2018/05/unpacking-xilinx-7-series-bitstreams-part-1 (accessed on 30 December 2023).

22. AMD Adaptive Computing Documentation Portal—docs.xilinx.com. Available online: https://docs.xilinx.com/r/en-US/ug470_7Series_Config (accessed on 30 October 2023).
23. Afshine Amidi, S.A. Recurrent Neural Networks Cheatsheet. Available online: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks (accessed on 10 November 2023).
24. Torrance, R.; James, D. The state-of-the-art in semiconductor reverse engineering. In Proceedings of the 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), San Diego, CA, USA, 5–9 June 2011; pp. 333–338.
25. Shiyanovskii, Y.; Wolff, F.; Rajendran, A.; Papachristou, C.; Weyer, D.; Clay, W. Process reliability based trojans through NBTI and HCI effects. In Proceedings of the 2010 NASA/ESA Conference on Adaptive Hardware and Systems, Anaheim, CA, USA, 15–18 June 2010; pp. 215–222. [CrossRef]
26. Bao, C.; Forte, D.; Srivastava, A. On Reverse Engineering-Based Hardware Trojan Detection. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2016**, *35*, 49–57. [CrossRef]
27. Hoque, T.; Cruz, J.; Chakraborty, P.; Bhunia, S. Hardware IP Trust Validation: Learn (the Untrustworthy), and Verify. In Proceedings of the 2018 IEEE International Test Conference (ITC), Harbin, China, 15–17 August 2018; pp. 1–10. [CrossRef]
28. Nguyen, L.N.; Cheng, C.L.; Prvulovic, M.; Zajić, A. Creating a Backscattering Side Channel to Enable Detection of Dormant Hardware Trojans. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 1561–1574. [CrossRef]
29. Bao, C.; Forte, D.; Srivastava, A. On application of one-class SVM to reverse engineering-based hardware Trojan detection. In Proceedings of the Fifteenth International Symposium on Quality Electronic Design, Santa Clara, CA, USA, 3–5 March 2014; pp. 47–54. [CrossRef]
30. Bao, C.; Xie, Y.; Liu, Y.; Srivastava, A. Reverse Engineering-Based Hardware Trojan Detection. In *The Hardware Trojan War: Attacks, Myths, and Defenses*; Bhunia, S., Tehranipoor, M.M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 269–288. [CrossRef]
31. Zhou, E.R.; Li, S.Q.; Chen, J.H.; Ni, L.; Zhao, Z.X.; Li, J. A Novel Detection Method for Hardware Trojan in Third Party IP Cores. In Proceedings of the 2016 International Conference on Information System and Artificial Intelligence (ISAI), Hong Kong, China, 24–26 June 2016; pp. 528–532. [CrossRef]
32. Lodhi, F.K.; Hasan, S.R.; Hasan, O.; Awwadl, F. Power profiling of microcontroller's instruction set for runtime hardware Trojans detection without golden circuit models. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 294–297. [CrossRef]
33. Huang, Z.; Wang, Q.; Chen, Y.; Jiang, X. A Survey on Machine Learning Against Hardware Trojan Attacks: Recent Advances and Challenges. *IEEE Access* **2020**, *8*, 10796–10826. [CrossRef]
34. Vakil, A.; Behnia, F.; Mirzaeian, A.; Homayoun, H.; Karimi, N.; Sasan, A. LASCA: Learning Assisted Side Channel Delay Analysis for Hardware Trojan Detection. In Proceedings of the 2020 21st International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 25–26 March 2020; pp. 40–45. [CrossRef]
35. Sumarsono, A.; Masters, Z. Application of LSTM Auto Encoder in Hardware Trojan Detection. In Proceedings of the 2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 8–11 March 2023; pp. 566–571. [CrossRef]
36. Kataria, J.; Housley, R.; Pantoga, J.; Cui, A. Defeating Cisco Trust Anchor: A Case-Study of Recent Advancements in Direct FPGA Bitstream Manipulation. In Proceedings of the 13th USENIX Conference on Offensive Technologies, Santa Clara, CA, USA, 12–13 August 2019; WOOT'19; p. 5.
37. Chakraborty, R.S. ProTro: A Probabilistic Counter Based Hardware Trojan Attack on FPGA Based MACSec Enabled Ethernet Switch. In *Proceedings of the Security, Privacy, and Applied Cryptography Engineering: 9th International Conference SPACE 2019, Gandhinagar, India, 3–7 December 2019*; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11947, p. 159.
38. Krautter, J.; Gnad, D.R.; Tahoori, M.B. FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 44–68. [CrossRef]
39. Johnson, A.P.; Patranabis, S.; Chakraborty, R.S.; Mukhopadhyay, D. Remote dynamic partial reconfiguration: A threat to Internet-of-Things and embedded security applications. *Microprocess. Microsyst.* **2017**, *52*, 131–144. [CrossRef]
40. Chakraborty, R.S.; Saha, I.; Palchaudhuri, A.; Naik, G.K. Hardware Trojan insertion by direct modification of FPGA configuration bitstream. *IEEE Des. Test* **2013**, *30*, 45–54. [CrossRef]
41. Swierczynski, P.; Fyrbiak, M.; Koppe, P.; Paar, C. FPGA Trojans through detecting and weakening of cryptographic primitives. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2015**, *34*, 1236–1249. [CrossRef]
42. Swierczynski, P.; Fyrbiak, M.; Koppe, P.; Moradi, A.; Paar, C. Interdiction in practice—Hardware Trojan against a high-security USB flash drive. *J. Cryptogr. Eng.* **2017**, *7*, 199–211. [CrossRef]
43. Heiner, J.; Sellers, B.; Wirthlin, M.; Kalb, J. FPGA partial reconfiguration via configuration scrubbing. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, 31 August–2 September 2009; pp. 99–104. [CrossRef]
44. Chaudhuri, J.; Chakrabarty, K. Diagnosis of Malicious Bitstreams in Cloud Computing FPGAs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2023**, *42*, 3651–3664. [CrossRef]
45. Zhao, M.; Suh, G.E. FPGA-Based Remote Power Side-Channel Attacks. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–23 May 2018; pp. 229–244. [CrossRef]

46. Gnad, D.R.E.; Oboril, F.; Tahoori, M.B. Voltage drop-based fault attacks on FPGAs using valid bitstreams. In Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 4–8 September 2017; pp. 1–7. [CrossRef]
47. Sugawara, T.; Sakiyama, K.; Nashimoto, S.; Suzuki, D.; Nagatsuka, T. Oscillator without a Combinatorial Loop and its Threat to FPGA in Data Center. *Electron. Lett.* **2019**, *55*, 640–642. [CrossRef]
48. Yu, S.; Gu, C.; Liu, W.; O'Neill, M. Deep Learning-Based Hardware Trojan Detection With Block-Based Netlist Information Extraction. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 1837–1853. [CrossRef]
49. Kumar, R.; Jovanovic, P.; Burleson, W.; Polian, I. Parametric Trojans for Fault-Injection Attacks on Cryptographic Hardware. In Proceedings of the 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, Busan, Republic of Korea, 23 September 2014; pp. 18–28. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.