


# Machine Learning-based Classification of Hardware Trojans in FPGAs Implementing RISC-V Cores

Stefano Ribes<sup>1</sup>, Fabio Malatesta<sup>2</sup>, Grazia Garzo<sup>3</sup> and Alessandro Palumbo<sup>4</sup> <sup>a</sup>

<sup>1</sup>*Department of Computer Science and Engineering, Chalmers University of Technology, Sweden*

<sup>3</sup>*University of Siena, Italy*

<sup>4</sup>*CentraleSupélec, Inria, Univ Rennes, CNRS, IRISA, France*

<sup>1</sup>*ribes@chalmers.se*, <sup>2</sup>*fabio.malatesta27@gmail.com*, <sup>3</sup>*g.garzo@student.unisi.it*, <sup>4</sup>*alessandro.palumbo@centralesupelec.fr*

**Keywords:** Hardware Security, Machine Learning, Hardware Trojans, Feature Importance, FPGA, RISC-V.

**Abstract:** Hardware Trojans (HTs) pose a severe threat to integrated circuits, potentially compromising electronic devices, exposing sensitive data, or inducing malfunction. Detecting such malicious modifications is particularly challenging in complex systems and commercial CPUs, where they can occur at various design stages, from initial HDL coding to the final hardware implementation. This paper introduces a machine learning-based strategy for the detection and classification of HTs within RISC-V soft cores implemented in Field-Programmable Gate Arrays (FPGAs). Our approach comprises a systematic methodology for comprehensive data collection and estimation from FPGA bitstreams, enabling us to extract insights ranging from hardware performance counters to intricate metrics like design clock frequency and power consumption. Our ML models achieve perfect accuracy scores when analyzing features related to both synthesis, implementation results, and performance counters. We also address the challenge of identifying HTs solely through performance counters, highlighting the limitations of this approach. Additionally, our work emphasizes the significance of Implementation Features (IFs), particularly circuit timing, in achieving high accuracy in HT detection.


## 1 INTRODUCTION

The threat of malicious undesired modifications in integrated circuits is especially relevant in the current global economy, where complex electronic devices, such as commercial CPUs, are often designed, manufactured and sold by different companies in different countries (DIGITIMES, 2012). CAD tools used throughout their design cycle can be produced by other companies, typically independently from the design and manufacturing processes. On top of that, the IP building blocks included in such devices are often purchased from yet different parties (Roy et al., 2008), (Potkonjak, 2010). As several actors have access to the production flow, there are multiple stages where a malicious hardware modification, known as Hardware Trojan (HT), may undergo (Rostami et al., 2013). Hence, it is challenging to demonstrate that the integrity of a manufactured product of such complexity has not been compromised. Even minor, malicious, hardware changes can have devastating effects, such as allowing an attacker to acquire supervisor

privileges, obtain a cryptographic key, or even execute malicious software, as demonstrated in (Jin et al., 2012), (Tsoutsos and Maniatakis, 2014), (Wang et al., 2012). A notable example of HT giving supervisor mode permissions, the *Rosenbridge backdoor*, was found in 2018 in a commercial Via Technologies C3 processor (Domas, 2018). These results seriously question the security of any modern system and call for a reliable and efficient way of identifying compromised CPU devices, ideally at the end-user level (Nikiema et al., 2023), (Cassano et al., 2022). Being not only able to identify but also to *classify* a malicious HT in a device could allow the end-user to gain insights on the threat and counter-attack it either in the chip itself, if necessary when a viable solution, or when developing defense mechanisms for future designs.

Incorporating Machine Learning (ML) into the realm of integrated circuit security presents a promising direction for addressing the challenge of detecting and classifying HTs, potentially overcoming traditional methods for identifying malicious modifications. ML algorithms, powered by advanced algorithms and computational capabilities, offer a data-

---

<sup>a</sup>  <https://orcid.org/0000-0002-0034-6189>

driven approach to this problem. By analyzing intricate patterns and anomalies in the behavior of integrated circuits, ML models can enhance our ability to uncover hidden threats.

## 2 CONTRIBUTIONS

We summarize our contributions to the field of HT detection and classification as follows:

- A systematic methodology for an extensive collection and estimation of numerical data from FPGA bitstreams: our proposed approach includes the extraction of valuable insights from the soft RISC-V microprocessor, spanning various data sources, including hardware performance counters, and extending to more complex-to-acquire metrics such as, for instance, design clock frequency and power consumption.
- ML model performance and robustness: we demonstrate the high performance and robustness of our ML model to address the unexplored challenge of detecting and classifying HTs introduced by an untrusted CAD tool in a trusted IP core.
- Comprehensive feature evaluation: we conduct an extensive exploration of feature importance in ML-based HT detection and classification. Our experiments suggest a critical role of features derived from the synthesis and implementation process in achieving high models' performance. We further leverage Uniform Manifold Approximation and Projection (UMAP) (McInnes et al., 2018) to visually confirm the significance of bitstream-related input features, such as the number of used LUTs in the design or the circuit's worst negative slack.

## 3 RELATED WORK

Most of the existing work on HT focuses on *system-level* implementation methodologies that allow the user to obtain a trusted system even when containing untrusted components (Šišeković et al., 2019), (Shila et al., 2015), (Basak et al., 2017). One way to do this is by running trustworthy software on systems with an untrusted CPU (Dubeuf et al., 2013), (Bloom et al., 2009), (Arikan et al., 2022).

During the last two decades, several techniques to limit HTs effects by detecting them before system deployment have been proposed. They generally consist of *circuit-level* approaches that aim at detecting HTs at design time via logic testing (Chuan et al., 2017),

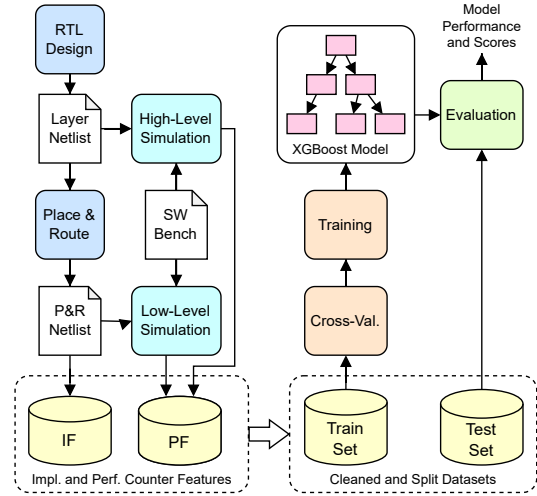


Figure 1: Overview of the data collection pipeline and models' training and evaluation.

formal property verification (Zhang et al., 2015), side-channel analysis (Liu et al., 2017), structural and behavioral analysis (Salmani and Tehranipoor, 2013), (Salmani and Tehranipoor, 2012), (Palumbo et al., 2021), and machine learning (Dong et al., 2019), (Huang et al., 2020).

Regarding possible attacking fronts, CAD tools can become a threat to FPGA-based systems security (Sunkavilli et al., 2021b), (Zhang and Qu, 2019), (Duncan et al., 2019). In particular, they may interfere directly with the FPGA configuration by injecting HTs in the produced bitstream (Ender et al., 2019), (Sunkavilli et al., 2021a). To the best of our knowledge, detecting HTs introduced by an untrusted CAD tool in a trusted IP core has been explored in a limited manner in prior research. In order to prevent malicious CAD tools, a *security rule checking* paradigm has recently been proposed for driving the adoption of CAD tools supporting the trustworthiness and security of systems throughout the whole design process (Xiao et al., 2016). For checking the trustworthiness of CAD tools, in (Palumbo et al., 2022) is proposed a ML-based solution to detect the presence of HTs in FPGA bitstreams.

## 4 METHODOLOGY

This section outlines our methodology, which covers data collection and cleaning, followed by ML model selection and implementation for classifying HTs.

## 4.1 System Setup for Data Collection

Figure 1 shows the followed steps to define, train and evaluate the ML-based models for classifying HTs. On the left side of the figure, we report the FPGA-related flow, which aims at extracting and gather relevant features from the RISC-V bitstreams. Our feature extraction process started with the implementation of the RTL design of a soft 3-stage pipeline RISC-V Ibex processor, which was further modified to include one of four potential HT models described in Trust-Hub (Shakya et al., 2017), *i.e.*, `Clk_Mod`, `Critical`, `MitM`, and `Fetch`. In our detection and classification modeling, we categorized HTs as either “triggered” or “not-triggered”, with the exception of the `Critical` HT, which operates continuously. The selected HT models are detailed as follows:

- `Clk_Mod`: This model corresponds to the B15-T100 model in the Trust-Hub platform. It encompasses an HT capable of slowing down the microprocessor’s clock, thereby impacting program throughput and potentially causing a denial-of-service.
- `Critical`: Inspired by models S35932-T300 and MAC10GE-T100 from Trust-Hub, this HT model introduces additional logic into the combinatorial path. It alters the microprocessor’s timing behavior and performance. It operates as an always-on HT and can potentially lead to a denial of service. This HT model has been positioned between the core’s instruction bus and its instruction memory.
- `MitM`: Drawing inspiration from the PIC16F84-T100 Trust-Hub model, the `MitM` HT simulates a man-in-the-middle scenario between the microprocessor and the instruction memory. It requires a specific trigger event to activate, and once triggered, it modifies fetched instructions, coercing the execution of a malicious program and resulting in changes to the microprocessor’s functionality.
- `Fetch`: The `Fetch` HT model is akin to the B19-T300 model proposed in the Trust-Hub platform. It interferes with the microprocessor’s fetching activity, altering the address of loaded instructions. Similar to `MitM`, it forces the execution of a malicious program, thereby modifying the microprocessor’s behavior and functionality.

Following the completion of the RTL design phase, we proceeded to execute synthesis, place & route (P&R), and implementation utilizing the Xilinx Vivado toolchain, specifically version 2018.2. We targeted a Xilinx XC7Z020-1CLG484C Zynq-7000 FPGA. In order to generate a more diverse and realist

Table 1: The used Vivado settings for generating the bitstreams implementing the altered soft RISC-V microprocessor.

Synthesis Settings	Implementation Settings	Logic Opt.
Vivado default	Vivado default	On
Vivado default	Vivado default	Off
Flow Area Opt. (medium)	Ultra Low Fast Methodology	On
Flow Area Alternate Routability	Ultra Low Fast Methodology	On

set of data, we defined four distinct combinations of synthesis and implementation settings, as detailed in Table 1, during the bitstream generation phase.

Once our designs were ready, we performed a set of tests with a software testbench, both during the synthesis phase, simulating their layer netlist, and during the P&R phase, simulating their post-P&R netlist. During simulation, we collected a diverse array of features, which can be categorized into two main groups, as detailed in Table 2. The first subset of features is derived from hardware performance counters integrated into the design. This includes metrics like the number of executed cycles, the count of retired instructions, and more. These measurements are obtained while running selected benchmarks, which consist of Coremark, Median, Multiply, Rsort, and Towers, similarly to what considered in the approach outlined in (Palumbo et al., 2022).

A second set of collected data reports design characteristics extracted from the FPGA synthesis and implementation phases. These characteristics include parameters such as design area, quantified in terms of the number of Look-Up Tables (LUTs) and Flip-Flops (FFs), as well as timing metrics like worst negative slack. Furthermore, power consumption and device temperature trends were also recorded as part of this data collection.

We subsequently categorize the first group of features as “Performance Features” (PFs) and the second group as “Implementation Features” (IFs). Our data collection process resulted in a total of 160 data samples: four distinct HT types with triggering and non-triggering conditions, as well as a HT-free scenario, each combined with four different Vivado settings and five distinct benchmarks.

## 4.2 Features Extraction from the End Users’ Perspective

In an industrial context, extracting feature data from a bitstream that implements a microprocessor, as detailed in Table 2, is a practical and achievable process, despite being challenging. In this section, we propose and discuss a series of methods for collecting the aforementioned data features from a given FPGA bitstream.

Table 2: Features extracted from the considered dataset.

Feature ID	Description
<b>Performance Features (PFs)</b>	
Cycles	Number of clock cycles to execute the program
InstrRet	Number of instructions retired in the program
LSUs	Total waiting cycles to access data memory
FetchWait	Total waiting cycles before instruction fetch
Loads	Number of executed load instructions
Stores	Number of executed store instructions
Jumps	Number of executed jump instructions
CondBran	Number of executed conditional branches
ComprIns	Number of executed compressed instruction
TakCBran	Number of taken conditional branches
MulWait	Cycles for multiplication operation completion
DivdWait	Cycles for division operation completion
Benchmark	Program under execution (text label)
<b>Implementation Features (IFs)</b>	
LUTs	Final number of LUTs in the design
FFs	Final number of FFs in the design
AvgDynPow	Avg. dynamic power consumption [W]
AvgTotPower	Avg. total power consumption [W]
Timing	Worst negative slack (the circuit critical path) [ns]
Temperature	Temperature trend

#### 4.2.1 Performance Features Extraction

PFs comprise the values written by the microprocessor into its registers during program execution. These values can be accessed through versatile interfaces such as AXI, UART, I<sup>2</sup>C, or dedicated debugging interfaces. Virtually all modern microprocessors offer dedicated registers for storing predefined and configured hardware performance counters, eliminating the need for additional hardware modifications to acquire this data.

#### 4.2.2 Timing and Number of LUTs and FFs Estimation

The synthesis and implementation of the circuit on the FPGA via Vivado provide essential information, including the count of Look-Up Tables (LUTs), Flip-Flops (FFs), and timing characteristics. Vivado furnishes these metrics directly once the implementation process is completed. In case users do not have access to the Vivado reports about detailed timing analysis, an external tunable clock generator can be leveraged for retrieving such information, *i.e.*, the circuit’s critical path slack. By incrementally increasing the clock frequency until the system experiences its first failure, it becomes possible to discern when the critical path’s timing is breached. This frequency corresponds to the maximum achievable circuit clock frequency, and its reciprocal defines the worst negative slack. A similar discussion can be made for LUTs and FFs: an estimate of their number can be extracted through reverse engineering of the circuit logic netlist. However, the precise contents of programmed LUTs and FFs still remains a challenging and impractical en-

deavor (Benz et al., 2012).

#### 4.2.3 Power Consumption Estimation

Vivado provides a reliable method for estimating circuit power consumption through post-implementation simulations. This estimation is grounded in the fidelity of the system’s behavior in post-implementation test benches, which are considered the most trustworthy as they exclusively focus on real signals synthesized and implemented during the process. By configuring Vivado settings, one can generate “SAIF” (Switching Activity Interchange Format) files that collect power trace data via test-bench simulations.

#### 4.2.4 Temperature Estimations

Temperature estimations have been obtained through the Vivado XADC monitor. Such a monitor dumps the temperature trace over program execution. We considered the mathematical integral value of the dumped temperature traces as a feature. The Zynq-7000 board features a temperature sensor linked to XADC signals, enabling the monitoring of FPGA runtime temperature trends. To ensure independent measurements, we adopted the following procedure:

1. Powering on the board.
2. Loading the bitstream of the desired implementation after 20 minutes of board activation.
3. Keeping the reset active for 1 minute and then deactivating it to initiate program execution.
4. Powering off the board and repeating the entire process after a 30-minute interval upon powering on the board.

This meticulous approach ensures reliable temperature data, regardless of Vivado tool implementation positional choices, and facilitates accurate temperature trend estimation.

### 4.3 Datasets Generation

We here outline the process of generating datasets for training and testing our ML models.

#### 4.3.1 Combining Non- and Triggered HTs

Measurements and data collection are done either after a selected HT has been turned on, *i.e.*, triggered, or when left inactive, *i.e.*, non-triggered (*Critical* trojans cannot be turned off). We include both triggered and non-triggered HTs in the initial dataset before subsequent splits. By including both triggered and non-

triggered HTs in the initial dataset we provide a more realistic view over the real data distribution.

### 4.3.2 Diverse Distribution of HT-free Samples

We offer two dataset options:

- *Uniform distribution* (80-20 split): we maintain an 80-20 random split between train and test sets, respectively, while ensuring equal class representation, serving as a performance baseline.
- *Realistic distribution* (80-20 split with 50% HT-free samples in test set): this scenario mirrors real-world conditions, emphasizing the practical challenge of distinguishing HTs from HT-free instances.

### 4.3.3 Exploring Model Robustness

To explore the minimal data requirements for achieving high accuracy, we conducted a series of experiments where the test set remained fixed (*i.e.*, 20% of the data), and we progressively reduced the size of the training set. This process involves sampling a subset of the training data and training separate models on these subsets. Through this investigation, we better understand the models' robustness and their ability to maintain accuracy scores, *i.e.*, generalize to new data, under data scarcity, providing valuable information for practical deployment scenarios.

## 4.4 Model Training

In this sub-section, we delve into our choice of model, our validation strategy, and our exploration of the impact of input features on model performance.

### 4.4.1 Selection of XGBoost

For our machine learning framework, we opt for XGBoost (Chen et al., 2015), a state-of-the-art model recognized for its effectiveness in handling tabular data with relatively limited sample sizes.

### 4.4.2 Cross-Validation for Robustness

To ensure the robustness and reliability of our chosen model, we employ  $k$ -fold cross-validation with  $k$  set to 5 during the training phase on specific datasets. Cross-validation enhances our confidence in model performance by evaluating it across multiple folds or subsets of the dataset. This rigorous validation approach helps us measure how well the model generalizes to unseen data, providing a robust assessment of its capabilities.

### 4.4.3 Feature Impact Analysis

Understanding the influence of input features on our model's performance is essential. To this end, we conduct an in-depth analysis of feature impact on HT detection and classification. We explore three distinct scenarios:

- Both PFs and IFs as inputs: in this configuration, we leverage both performance features and implementation features as inputs to the model.
- PFs only: since IFs can be challenging to acquire, especially when synthesis and implementation reports are unavailable, this scenario isolates the contribution of performance-related metrics.
- PFs plus one IF: here, we introduce a single implementation feature alongside PFs to investigate the individual influence of an implementation-related metric on model performance.

Through these analyses, we gain additional insights into the relative importance of different feature categories, helping us optimize the model's performance for HT-related tasks.

## 5 RESULTS AND DISCUSSION

We here present the results of our HT classification and detection accuracy experiments. The experiments were conducted with varying training percentages and dataset compositions, as detailed in the methodology section.

### 5.1 HT Classification and Detection

Figure 2a shows the HT accuracy when we systematically adjusted the training percentage, ranging from 80% down to 8%, while maintaining a fixed 20% test set size. When the test set featured a uniform class distribution with 20% HT-free samples, the classification accuracy remained consistently high at 100%, even as the training set size decreased. The detection accuracy followed a similar pattern, staying at 100% until a training set size of 56%, where it slightly dropped to 95% but remained robust. However, in the more realistic scenario with 50% HT-free samples in the test set, the classification accuracy remained at 100% with a training set size of 72% and 64%. The detection accuracy also stayed at 100% for these configurations. As the training set size reduced further, the classification and detection accuracy scores demonstrated some variation, but the model's performance remained notably strong. These results high-

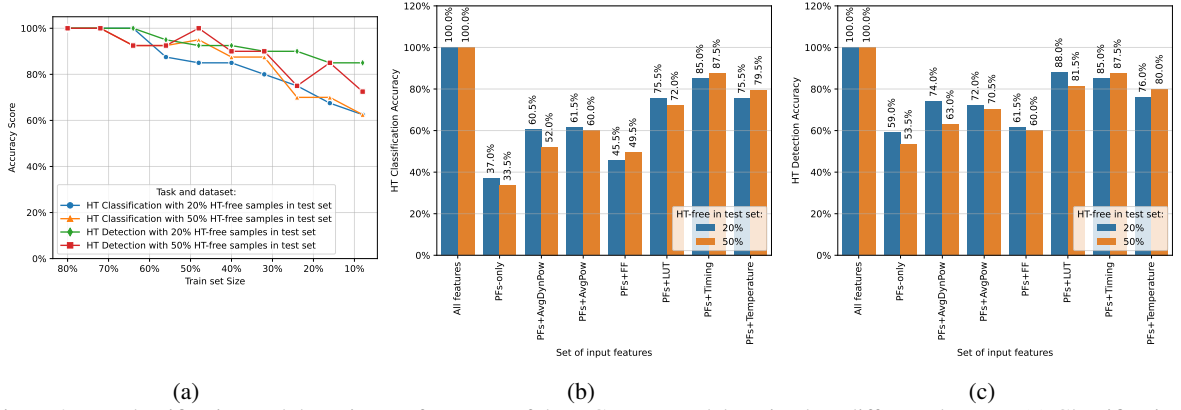


Figure 2: HT classification and detection performance of the XGBoost models trained on different datasets. (a) Classification and detection scores when varying training set size. (b) HT classification accuracy and (c) HT detection accuracy with different input features and percentages of HT-free samples in the test set.

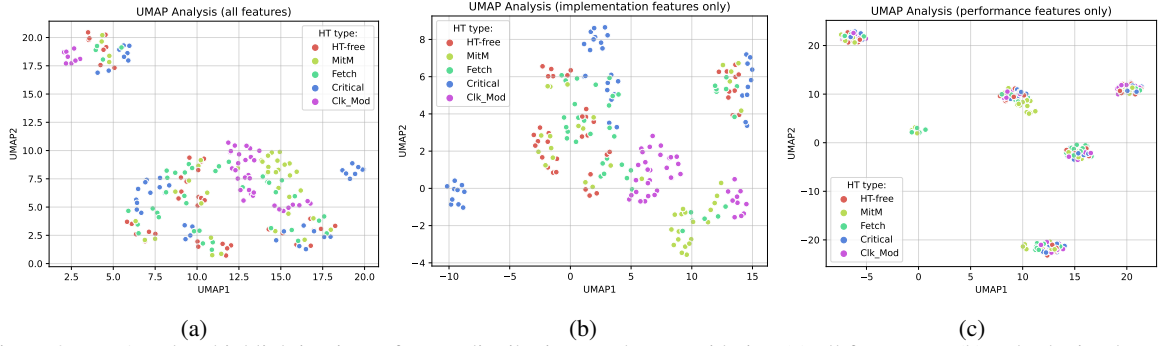


Figure 3: UMAP plots highlighting input feature distributions. When considering (a) all features or (b) only the implementation ones, the data points appear spread out and separable. On the other hand, (c) performance features alone are clumped together in UMAP space, suggesting the need of higher efforts in classifying HT classes.

light the model’s resilience in achieving high accuracy even with limited training data.

### 5.1.1 Performance with Limited Input Features

Figures 2b and 2c report the HT classification and detection accuracy scores with different input features and diverse percentages of HT-free samples in the test set (20% or 50%). Firstly, when considering all features in the model, regardless of the proportion of HT-free samples in the test set, the classification and detection accuracy scores consistently reach 100%. However, the performance of models relying solely on PFs exhibits limitations. In scenarios featuring 20% HT-free samples in the test set, PFs-only models yield relatively lower accuracy scores, with classification accuracy dropping to 37% and detection accuracy to 59% on the uniform test set (33.5% and 53.5% accuracy with 50% HT-free test samples). This suggests that PFs alone are insufficient for achieving high performance in both classification and detection tasks. When we augment PFs with one additional feature, such as the average dynamic power (AvgDynPow),

average total power (AvgTotPow), FFs, LUT, timing, and temperature, the model’s accuracy scores generally improve. Notably, models incorporating information about timing consistently perform well, with classification and detection accuracy scores reaching 85% and 100%, respectively, when 50% HT-free samples are present in the test set. This highlights the significance of timing-related features in distinguishing between HT and HT-free instances.

Overall, our findings highlight the crucial role of feature selection in HT detection and classification tasks. While PFs alone may fall short, the inclusion of timing features proves instrumental in enhancing the model’s accuracy and effectiveness, especially in scenarios where HT-free samples are more prevalent.

## 5.2 Features Importance and Distribution

The analysis of feature importance in our XGBoost model provides an additional understanding of the discriminative power of different input fea-

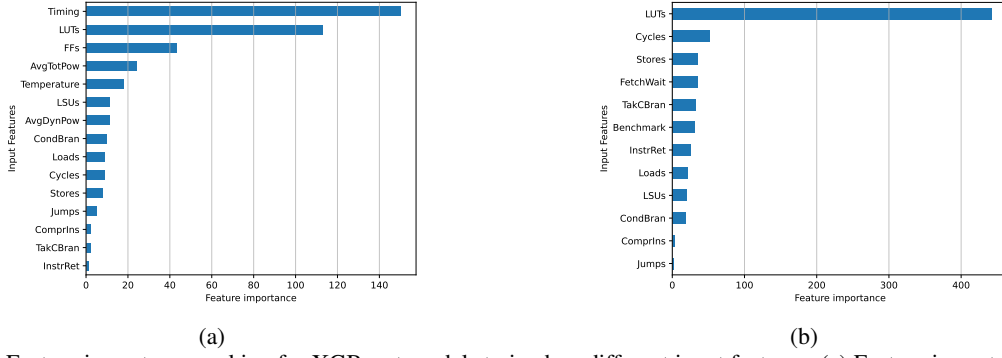


Figure 4: Feature importance ranking for XGBoost models trained on different input features. (a) Feature importance scores for XGBoost trained on all features. (b) Feature importance for XGBoost trained on PFs+LUTs.

tures. When utilizing all input features, IFs emerge as the most influential, with “timing” ranking highest, achieving a score of approximately 150, followed by LUTs at around 115, as showed in Figure 4a. However, if we focus on PFs and LUTs as input features, for example, as highlighted in Figure 4b, the importance of LUTs becomes particularly prominent, with a score of around 450, followed by Cycles at 60. This shift highlights the significance of LUTs when PFs are the primary input features.

To reinforce the argument regarding the importance of IFs, we can employ a Uniform Manifold Approximation and Projection (UMAP) analysis (McInnes et al., 2018), which helps visualize the distribution of input features in a reduced-dimensional space. In Figures 3a and 3b, which consider all features or only the implementation ones, respectively, we can see that data points are well-spread and separable, illustrating the effectiveness of IFs in feature discrimination. However, in Figure 3c, where only PFs are used, data points cluster together in UMAP space, indicating the need for more substantial efforts in classifying HT classes when relying solely on PFs.

## 6 CONCLUSIONS

In this work we provided a comprehensive and detailed methodology for Hardware Trojan (HT) detection and classification using machine learning. Key takeaways include the critical role of Implementation Features (IFs), with timing, *i.e.*, circuit worst negative slack, as the standout factor, in distinguishing HTs from HT-free instances. When leveraging all input features, our XGBoost model consistently achieves classification and detection accuracy scores of 100%. However, we showed that using Performance Features (PFs) alone is insufficient for achieving high accuracy, especially with a more realistic test data distribu-

tion. To overcome this, augmenting PFs with at least one IFs significantly improves model performance. UMAP analysis visually confirmed the effectiveness of IFs in feature discrimination. Overall, our findings emphasize the importance of IFs and the resilience of XGBoost as a robust HT detection and classification in integrated circuits.

## REFERENCES

- Arıkan, K., Palumbo, A., Cassano, L., Reviriego, P., Pontarelli, S., Bianchi, G., Ergin, O., and Ottavi, M. (2022). Processor security: Detecting microarchitectural attacks via count-min sketches. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(7):938–951.
- Basak, A., Bhunia, S., Tkacik, T., and Ray, S. (2017). Security assurance for system-on-chip designs with untrusted ips. *IEEE Transactions on Information Forensics and Security*, 12(7):1515–1528.
- Benz, F., Seffrin, A., and Huss, S. A. (2012). Bil: A tool-chain for bitstream reverse-engineering. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 735–738.
- Bloom, G., Narahari, B., and Simha, R. (2009). Os support for detecting trojan circuit attacks. In *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 100–103.
- Cassano, L., Mascio, S. D., Palumbo, A., Menicucci, A., Furano, G., Bianchi, G., and Ottavi, M. (2022). Is risc-v ready for space? a security perspective. In *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6.
- Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., et al. (2015). XGBoost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4.
- Chuan, X., Yan, Y., and Zhang, Y. (2017). An efficient triggering method of hardware Trojan in AES cryptographic circuit. In *Proc. Int. Conf. Integrated Circuits and Microsystems*, pages 91–95.



- DIGITIMES (2012). Trends in the global IC design service market. <http://www.digitimes.com/news/a20120313RS400.html?chid=2>.
- Domas, C. (2018). Hardware backdoors in x86 cpus. <https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPU-wp.pdf>.
- Dong, C., Chen, J., Guo, W., and Zou, J. (2019). A machine-learning-based hardware-trojan detection approach for chips in the internet of things. *International Journal of Distributed Sensor Networks*, 15(12):1550147719888098.
- Dubeuf, J., Hély, D., and Karri, R. (2013). Run-time detection of hardware trojans: The processor protection unit. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6.
- Duncan, A., Rahman, F., Lukefahr, A., Farahmandi, F., and Tehranipoor, M. (2019). Fpga bitstream security: A day in the life. In *2019 IEEE International Test Conference (ITC)*, pages 1–10.
- Ender, M., Swierczynski, P., Wallat, S., Wilhelm, M., Knopp, P. M., and Paar, C. (2019). Insights into the mind of a trojan designer: the challenge to integrate a trojan into the bitstream. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 112–119.
- Huang, Z., Wang, Q., Chen, Y., and Jiang, X. (2020). A survey on machine learning against hardware trojan attacks: Recent advances and challenges. *IEEE Access*, 8:10796–10826.
- Jin, Y., Maniatakos, M., and Makris, Y. (2012). Exposing vulnerabilities of untrusted computing platforms. In *Proc. Int. Conf. Computer Design*, pages 131–134.
- Liu, Y., Zhao, Y., He, J., Liu, A., and Xin, R. (2017). Scca: Side-channel correlation analysis for detecting hardware trojan. In *Proc. Int. Conf. Anti-counterfeiting, Security, and Identification*, pages 196–200.
- McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.
- Nikiema, P. R., Palumbo, A., Aasma, A., Cassano, L., Kritikakou, A., Kulmala, A., Lukkarila, J., Ottavi, M., Psiakis, R., and Traiola, M. (2023). Towards dependable risc-v cores for edge computing devices. In *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7.
- Palumbo, A., Cassano, L., Luzzi, B., Hernández, J. A., Reviriego, P., Bianchi, G., and Ottavi, M. (2022). Is your fpga bitstream hardware trojan-free? machine learning can provide an answer. *Journal of Systems Architecture*, 128:102543.
- Palumbo, A., Cassano, L., Reviriego, P., Bianchi, G., and Ottavi, M. (2021). A lightweight security checking module to protect microprocessors against hardware trojan horses. In *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6.
- Potkonjak, M. (2010). Synthesis of trustable ics using untrusted cad tools. In *Proceedings of the 47th Design Automation Conference*, pages 633–634.
- Rostami, M., Koushanfar, F., Rajendran, J., and Karri, R. (2013). Hardware security: Threat models and metrics. In *Proc. Int. Conf. Computer-Aided Design*, pages 819–823.
- Roy, J. A., Koushanfar, F., and Markov, I. L. (2008). Extended abstract: Circuit cad tools as a security threat. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*.
- Salmani, H. and Tehranipoor, M. (2012). Layout-aware switching activity localization to enhance hardware trojan detection. *IEEE Trans. Information Forensics and Security*, 7(1):76–87.
- Salmani, H. and Tehranipoor, M. (2013). Analyzing circuit vulnerability to hardware trojan insertion at the behavioral level. In *Proc. Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 190–195.
- Shakya, B., He, T., Salmani, H., Forte, D., Bhunia, S., and Tehranipoor, M. (2017). Benchmarking of hardware trojans and maliciously affected circuits. *Journal of Hardware and Systems Security*, 1(1):85–102.
- Shila, D. M., Venugopalan, V., and Patterson, C. D. (2015). Fides: Enhancing trust in reconfigurable based hardware systems. In *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7.
- Sunkavilli, S., Zhang, Z., and Yu, Q. (2021a). Analysis of attack surfaces and practical attack examples in open source fpga cad tools. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 504–509.
- Sunkavilli, S., Zhang, Z., and Yu, Q. (2021b). New security threats on fpgas: From fpga design tools perspective. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 278–283.
- Tsoutsos, N. G. and Maniatakos, M. (2014). Fabrication attacks: Zero-overhead malicious modifications enabling modern microprocessor privilege escalation. *IEEE Trans. Emerging Topics in Computing*, 2(1):81–93.
- Šišeković, D., Merchant, F., Leupers, R., Ascheid, G., and Kogreiss, S. (2019). Control-lock: Securing processor cores against software-controlled hardware trojans. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI, GLSVLSI '19*, pages 27–32.
- Wang, X., Mal-Sarkar, T., Krishna, A., Narasimhan, S., and Bhunia, S. (2012). Software exploitable hardware trojans in embedded processor. In *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 55–58. IEEE.
- Xiao, K., Nahiyan, A., and Tehranipoor, M. (2016). Security rule checking in ic design. *Computer*, 49(8):54–61.
- Zhang, J. and Qu, G. (2019). Recent attacks and defenses on fpga-based systems. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12(3):1–24.
- Zhang, J., Yuan, F., Wei, L., Liu, Y., and Xu, Q. (2015). Veritrust: Verification for hardware trust. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 34(7):1148–1161.