Timekeepers: ML-Driven SDF Analysis for Power-Wasters Detection in FPGAs

MOHAMED FATHY, ITEC, Karlsruhe Institute of Technology, Karlsruhe, Germany, Electronics, German University in Cairo, New Cairo City, Egypt, and Si-Vision LLC, Cairo, Egypt
HASSAN NASSAR, Karlsruhe Institute of Technology, Karlsruhe, Germany
MOHAMED ABD EL GHANY, Electronics, German University in Cairo, New Cairo City, Egypt and Si-Vision LLC, Cairo, Egypt
JÖRG HENKEL, Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany

As the integration of FPGAs into cloud computing platforms accelerates, the risk of fault injection attacks - especially through power-wasting designs - becomes increasingly critical. Malicious tenants can upload FPGA designs that, under specific input stimuli, generate excessive power consumption, jeopardizing the integrity of the shared power delivery network (PDN) and enabling denial-of-service or side-channel attacks. Traditional detection techniques relying on netlist and bitstream analysis struggle with generalization and can be evaded through circuit obfuscation and seemingly benign designs. In contrast to these netlist-based approaches, we introduce Timekeepers, a novel detection method that utilizes Standard Delay Format (SDF) timing data combined with machine learning to detect anomalous power behavior in synthesized FPGA designs.

Our method trains a decision tree classifier on SDF files generated from both benign and malicious designs, focusing on timing characteristics such as propagation delays and setup/hold violations to identify power wasters at the primitive level. By abstracting away from circuit connectivity and emphasizing timing patterns, our framework is both scalable and robust across different FPGA architectures. The classifier independently evaluates each FPGA component and aggregates the results using a threshold-based voting system to improve detection granularity and reduce false positives. Timekeepers achieves 99.6% accuracy and demonstrates superior performance compared to state-of-the-art solutions. Furthermore, our approach is platform-agnostic and does not require access to netlists or bitstreams, preserving intellectual property confidentiality while enhancing pre-deployment security checks.

CCS Concepts: • **Hardware → Reconfigurable logic and FPGAs**; • **Security and privacy → Hardware attacks and countermeasures**.

Additional Key Words and Phrases: Multi-Tenant FPGAs, Accelerated Clouds, Security

1 Introduction

FPGAs are widely used to accelerate various applications, such as cryptography, artificial intelligence, financial services, and real-time signal processing. Therefore, they are rapidly being integrated into cloud environments [1]. Cloud FPGAs offer user-customizable hardware acceleration that enables the runtime reconfiguration of computing resources to adapt to the current application, a feature that is lacking in ASICs.

Authors' Contact Information: Mohamed Fathy, ITEC, Karlsruhe Institute of Technology, Karlsruhe, BW, Germany and Electronics, German University in Cairo, New Cairo City, Cairo Governorate, Egypt and Si-Vision LLC, Cairo, Cairo Governorate, Egypt; e-mail: mohamed.fathy@partner.kit.edu; Hassan Nassar, Karlsruhe Institute of Technology, Karlsruhe, BW, Germany; e-mail: hassan.nassar@kit.edu; Mohamed Abd El Ghany, Electronics, German University in Cairo, New Cairo City, Cairo Governorate, Egypt and Si-Vision LLC, Cairo, Cairo Governorate, Egypt; e-mail: mohamed.abdel-ghany@guc.edu.eg; Jörg Henkel, Computer Science, Karlsruhe Institute of Technology, Karlsruhe, BW, Germany; e-mail: henkel@kit.edu.

ACM 1558-3465/2025/8-ART
https://doi.org/10.1145/3761809

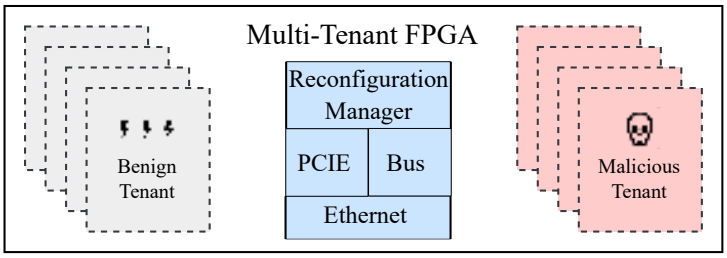


Fig. 1. Multi-tenant systems and their threats. Several tenants reside on the FPGA. All of them rely on the static design for reconfiguration, memory communication and communication to the outer world. If one or several of the tenants are malicious, they can inject faults and cause harm to the system.

Virtualization and multi-tenancy are key concepts of cloud computing; they are already applied to CPUs and GPUs. In contrast to CPUs and GPUs, the adoption of multi-tenant FPGAs in cloud systems while being heavily investigated [2, 3] is so far missing. The reason is that the presence of multiple tenants and resource sharing within cloud FPGA platforms raises significant security issues. The systems could harbor benign and malicious tenants (as illustrated in Fig. 1), which can pose threats to the shared physical chip. Malicious tenants can exploit the shared electrical level infrastructure, e.g., the Power Delivery Network (PDN), to carry out side channel or fault injection attacks that could extract sensitive data from benign tenants or compromise their computations [4, 5]. Current security solutions, including bitstream encryption, static verification, and access control policies, offer a degree of protection, but are often rigid, computationally intensive, or unable to identify complex runtime threats [6].

Power wasters are a particularly serious threat as they have been shown to cause serious financial damage to cloud providers [7]. These attacks often employ circuits with high switching activity that stress the PDN of the FPGA, inducing voltage droops that can corrupt the computation of neighboring tenants or cause a denial of service. Unlike aggressive or clearly malicious hardware constructs, power wasters frequently resemble functional logic, such as SHA cores or memory access patterns, that is common in legitimate workloads. This makes it increasingly difficult to flag these designs during bitstream-level or functional verification, especially when they are mixed with benign components [8] or spread over multiple tenants [9].

In this work, we offer significant insight into the limitations of current power-waster detection mechanisms. They focus on detecting malicious constructs within the circuit that attackers can find ways to hide, or they can use newer creative power wasters. However, while power wasters have evolved and use several different circuits, they all share a common feature. To waste such amounts of power, high switch activity has to be achieved, often violating timing constraints. Therefore, instead of analyzing the circuit's interconnections, analyzing the timing behavior of the circuit can provide more useful information.

Contrary to the state-of-the-art, we present a Standard Delay Format (SDF)-based power-wasters detection mechanism that utilizes timing analysis along with machine learning models to identify malicious circuits on FPGA. By analyzing setup and hold timing violations sourced from the SDF file, which contains precise delay information for circuit paths, the framework identifies discrepancies caused by the inclusion of power wasters. The key contributions of this work are:

- We are the first to take advantage of timing behavior of circuits implemented on FPGAs to identify power wasters; unlike state-of-the-art solutions looking at the netlist, the timing violations cannot be obfuscated.
- We provide a classification model based on SDF files that, unlike the state of the art, does not look in the netlist in any form and hence, it preserves the privacy of the users private data.
- Unlike the state-of-the-art, as our solution looks only at the timing behavior of circuits, it is generalizable and does not require preprocessing that is specific to a certain FPGA model.

The remainder of the paper is structured as follows. Important background material is provided in Section 2. In Section 3, the proposed technique is explained in depth, including the procedures to extract and evaluate timing violations. The evaluation process is described in Section 4. Section 5 concludes the report by summarizing the results and discussing possible directions for further investigation.

2 Background

Cloud FPGAs represent a powerful computing model that combines the advantages of hardware acceleration while being adaptable and customizable at runtime. They offer high-performance computing capabilities for various applications without the need for costly physical FPGA installations by providing on-demand access to configurable hardware [10]. Although this transformation has opened up new opportunities in AI, cryptography, and real-time data processing, it has also raised significant concerns related to security and reliability. The multitenancy characteristic of cloud FPGAs optimizes resource utilization, but exposes shared hardware to risks such as fault injection and side-channel attacks. To implement FPGA-based cloud services securely and efficiently, it is essential to understand these concerns and their implications.

2.1 Dynamic Partial Reconfiguration

Dynamic Partial Reconfiguration (DPR) is a key enabler of flexible and efficient FPGA utilization, especially in scenarios involving reconfigurable computing and multi-tenant systems. It allows a portion of the FPGA fabric to be reconfigured at runtime without disrupting the operation of the remaining static logic. This architectural capability has been widely adopted in domains such as reconfigurable processors [11, 12] and domain-specific hardware accelerators [13, 14], where runtime adaptability is essential for performance and power efficiency.

In a typical DPR design, the FPGA is partitioned into a fixed static region and one or more partially reconfigurable regions (PRRs). Each PRR can support multiple distinct configurations, enabling the on-demand deployment of tenant-specific logic. All configurations targeting a given PRR must conform to a predefined interface specification, ensuring compatibility with the static system shell. For each design variant, both a partial bitstream and an accompanying "blank" configuration, used to reduce idle power consumption, are generated as part of the implementation flow. These reconfiguration assets can be loaded at runtime via external interfaces such as JTAG or through internal reconfiguration mechanisms such as the Internal Configuration Access Port (ICAP), managed by a runtime configuration controller [15].

In multi-tenant cloud FPGAs, DPR is central to enabling resource sharing while preserving isolation and flexibility. Cloud providers often deploy a generic static shell containing reconfiguration logic and management interfaces, with multiple PRRs left blank at deployment time. When tenant requests arrive, the system populates the PRRs with the corresponding partial bitstreams, customizing the FPGA fabric to each workload while maintaining the integrity of the static system infrastructure. This approach not only improves FPGA resource utilization, but also simplifies the isolation and control of tenant designs. However, it also introduces additional security risks, as dynamically loaded designs may include stealthy power-wasting circuits or timing faults that exploit the shared physical substrate.

2.2 Cloud FPGAs

Cloud FPGAs [16] have revolutionized cloud computing by offering on-demand access to reconfigurable hardware, eliminating the high costs of owning FPGA systems. Through FPGA-as-a-Service (FaaS) [10], providers like AWS, Microsoft Azure, and Alibaba Cloud enable users to allocate and program FPGA resources efficiently. This flexibility has made it suitable for machine learning, cryptography, finance, and bioinformatics.

The strength of cloud FPGAs lies in their ability to merge software-defined flexibility with hardware acceleration [17]. Unlike GPUs and ASICs, which are optimized for specific tasks, FPGAs can be dynamically

reprogrammed to adapt to evolving computational needs, enhancing performance and energy efficiency. DPR further expands this adaptability by allowing real-time hardware modifications without disrupting other tasks. This feature is crucial for sharing the FPGA between different workloads that change over time.

Cloud FPGAs also introduce multi-tenancy, where multiple users share the same FPGA hardware, improving resource utilization and cost efficiency. While current commercial offerings of multi-tenant FPGAs are limited, the concept is actively explored by cloud service providers and research platforms. For example, AMD-HACC have supported a previous work on potential issues in multi-tenant settings [18], and IBM has proposed architectural support for multi-tenancy in FPGAs [19]. These efforts highlight growing interest in resource sharing for cost efficiency.

As multi-tenancy becomes more viable, this shared model raises serious security concerns. Malicious tenants can exploit FPGA-level vulnerabilities to interfere with or extract sensitive data from co-tenants. One significant attack is address-redirection [20]. In this attack, a malicious tenant injects faults to manipulate memory mappings. This manipulation is used to configure a bitstream to the wrong FPGA region, potentially compromising computations.

Another critical threat is remote side-channel attacks [21]. In these attacks, malicious tenants use ring oscillators (ROs) to analyze power variations and infer co-tenant activities. This poses risks to cryptographic applications and AI models. Furthermore, remote fault injection [22] can disrupt computations by triggering subtle timing errors. This is particularly dangerous for safety-critical purposes, e.g. medical diagnostics.

Configuration-based attacks also pose significant risks, targeting reconfiguration interfaces such as ICAP [23]. Attackers may alter configuration data, causing persistent faults. Although there are security mechanisms such as bitstream encryption and CRC checks, sophisticated attackers may bypass these protections [20].

To mitigate these threats, researchers and cloud providers are implementing security measures such as bitstream integrity verification, real-time monitoring, and hardware-based access control. The use of trusted execution environments (TEEs) [24] is also being explored to create secure enclaves for sensitive computations. However, they have been shown to be vulnerable to attacks that exploit the FPGA fabric [25] where the accelerators implemented on the FPGA can be used to leak data.

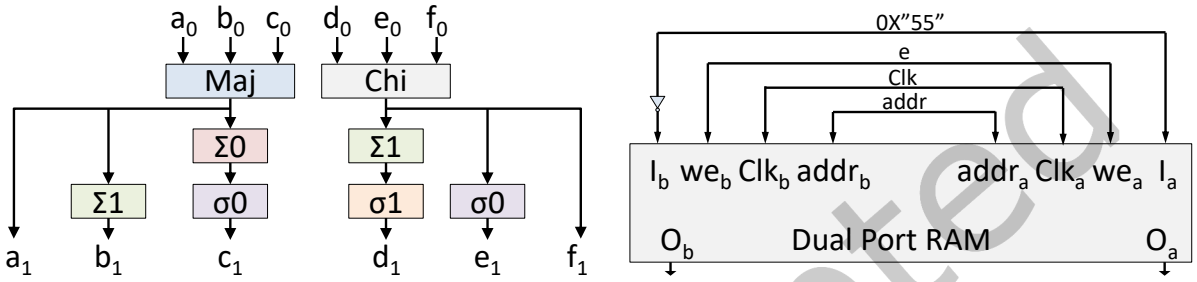
2.3 Power Wasters-Fault Injection

Multi-tenant cloud FPGAs are vulnerable to power-based fault injection attacks, as users or *'tenants'* who are malicious can exploit shared PDNs to interfere with co-located benign tenants [26]. Although FPGA vendors implement logical isolation between different tenants by restricting direct access to other user's logic and interconnect resources, the PDN remains a shared attack surface that adversaries can manipulate. Such attacks involve deploying power-wasting circuits, such as ring oscillators (ROs), or other high-switching activity components, which create voltage fluctuations across the FPGA fabric. These fluctuations can induce faults in victim circuits, leading to reliability issues, denial-of-service (DoS) attacks, and even side-channel information leakage. A previous work [27] has demonstrated that by activating thousands of ROs, an attacker can cause excessive voltage droops. These droops trigger a global reset in FPGAs, forcing the device to reload its bitstream and rendering all co-tenants temporarily inoperable [28]. Beyond simple DoS attacks, controlled power manipulation can inject delay faults into sensitive cryptographic modules, such as AES cores, allowing attackers to compromise encryption outputs and extract secret keys [29]. Similarly, malicious ROs placed near true random number generators (TRNGs) have been shown to degrade randomness by altering circuit timing, leading to predictable output and weakening security protocols [27]. In particular, these attacks have been replicated in real-world cloud environments, such as Amazon EC2 F1 instances [7], demonstrating that remote power-based side channel attacks are feasible in commercial cloud FPGAs.

Beyond direct circuit faults, adversaries can also use voltage fluctuations as an information leakage channel. Since power consumption in FPGA circuits is highly correlated with computation, especially in cryptographic

applications, attackers can deploy on-chip voltage sensors to capture power traces and recover secret information. For instance, researchers [30] have successfully used RO- and time-to-digital converter (TDC)-based voltage sensors. They extracted encryption keys from AES and RSA cores via side-channel analysis [5, 31].

Mitigating such attacks is possible at runtime. One approach is to integrate distributed voltage sensors across the FPGA fabric to detect abnormal power consumption patterns and identify potential attackers [32]. Additionally, clock edge suppression techniques can help stabilize voltage fluctuations by dynamically adjusting clock timing in response to detected power changes [33].



(a) SHA-based power waster (b) RAM-based power waster

Fig. 2. Different types of seemingly benign power wasters from the state of the art.

Pre-deployment bitstream analysis can also be used to flag and block suspicious designs that include excessive ROs or other power-wasting circuits before they are loaded onto the FPGA [34]. However, these defenses must be carefully designed to avoid negatively impacting performance or legitimate high-power applications. This becomes significantly harder with seemingly benign types of power waster such as SHA and RAM-based power wasters [8, 35, 36] shown in Fig. 2. Such power wasters do not use self-oscillating ROs but induce timing faults on a large scale with high switching activities.

2.4 Timing simulation and SDF Files

Timing simulation is crucial for a comprehensive analysis of digital circuits [37]. Unlike functional simulation, which only evaluates a circuit's logical validity without considering the specifics of its physical realization, timing simulation incorporates circuit delays. This plays a vital role in identifying potential timing violations that could threaten circuit performance by effectively taking into account these delay factors. Such violations include issues with setup and hold times, clock skew, race conditions, and propagation delays problems that could lead to metastability or serious functionality problems, particularly in high-speed circuits.

Standard Delay Format (SDF) is a well-known file format that includes comprehensive timing information for smooth integration with simulation tools [37]. SDF files provide module route delays, connection delays, and cell delays. This enables the back-annotation of precise timing information into gate-level simulations, enabling designers to represent real-world circuit behavior. Furthermore, SDF files include necessary timing constraints such as skew, period, and path delay constraints, creating a framework to improve circuit performance by reducing timing variations.

An SDF file is organized in a specific manner, starting with a header section that contains key metadata such as the timescale and version of the file. The main part of the SDF file is the delay section [37], which outlines in detail the timing values of the interconnects and logic components. For example, a snippet that specifies delays for interconnects could look like this:

$$(INTERCONNECTin1top/test - reg - D(0.0027 :: 0.0028)(0.0029 :: 0.0030)),$$

where “in1” represents the source port, “top/test-reg/D” represents the destination, and the two sets of figures indicate the delays for rising and falling signals. Likewise, cell delays are represented with the IOPATH construct, as demonstrated in:

$$(IOPATH - I5 - O - (23.0 : 39.0 : 39.0)(23.0 : 39.0 : 39.0)),$$

which illustrates the delay from input I5 to output O, including minimum, typical, and maximum delay values. Another example is:

$$(IOPATH - CLK - Q - (0.2864 :: 0.2865)(0.3616 :: 0.3617)),$$

which represents the propagation delay from the clock input (CLK) to the output (Q) of a flip-flop, providing highly detailed timing information necessary for accurate simulations.

In addition to fundamental delay descriptions, SDF files contain a variety of important timing checks to ensure adherence to strict performance standards. These checks [37] cover the setup time, the hold time, the recovery time, the removal time, and the pulse width constraints. For instance, the SETUPHOLD statement specifies the setup and hold time requirements for flip-flops:

$$(SETUPHOLD(posedgeD)(posedgeCLK)(0.5805 :: 0.5805)(-0.1512 :: -0.1512)),$$

which establishes timing margins critical for avoiding data corruption and ensuring correct operation of the flip-flop. Moreover, SDF files feature conditional path delay constructs that enable dynamic assignment of timing values based on particular circuit conditions. A statement such as:

$$(COND - A\&\&!B - (IOPATHSY(0.7427 :: 0.7427)(0.6942 :: 0.6942))),$$

demonstrates this functionality, where the delay between input S and output Y of a multiplexer changes depending on the states of inputs A and B.

In gate-level simulations, SDF files include only delay information, not netlist data, ensuring that simulated circuit behavior closely reflects actual performance. By incorporating delay values from standard cell libraries and design constraints such as Xilinx Design Constraints (XDC), SDF files allow simulation tools to detect any timing violations early in post-layout analysis. This quick detection minimizes costly last-minute design changes and promotes efficient circuit design. Additionally, by supporting multicorner analysis with minimum, typical, and maximum delay values, SDF files enable designers to assess circuit performance under varying process, voltage, and temperature conditions.

2.5 Related works

State-of-the-art techniques explored a variety of machine-learning methods for detecting malicious FPGA bitstreams in multi-tenant cloud environments, where security risks arise from the sharing of hardware resources. The first technique applied is MaliGNNoma [38], which is a graph-oriented learning system. MaliGNNoma models the relationships between bitstreams and hardware interfaces, allowing for precise differentiation between legitimate and harmful circuits. Using a graph-based representation, MaliGNNoma enhances the model’s ability to generalize across different FPGA architectures and to recognize complex power-waster patterns. However, MaliGNNoma has limited coverage of attack types; glitch-induction attacks [7], BRAM-based power-wasters [36], shift-register-based power-wasters [35], and Reed-Solomon-based power-wasters [8] are not covered.

Meta-Scanner [8] introduces a metadata-based detection framework designed to quickly adjust to evolving power wasters by learning from a small number of malicious bitstream instances. It extracts features by comparing an empty bitstream with the design bitstream to identify resemblances. However, such a method has a shortcoming as it cannot accurately classify weak versions of power-wasters, relying on a three-category classifier (red, green, yellow) instead of two (malicious and benign). It uses red for clearly malicious designs, green for clearly benign designs, and yellow for designs that may contain weak power wasters or benign designs that resemble them.

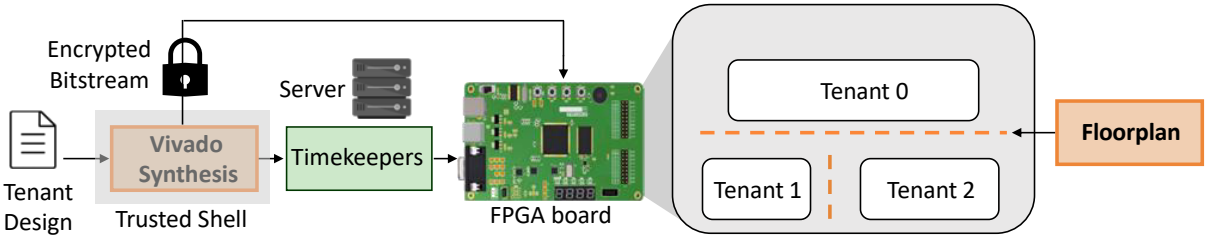


Fig. 3. Proposed detection scheme. The tenant design gets synthesized on a trusted shell. The server then checks the generated SDF file to check if power wasters are present in the design or not before deploying it on the FPGA. In case no powerwasters are detected, the encrypted bitstream coming from the trusted shell is uploaded directly to the FPGA.

Another proposed method adopts a CNN-based framework [39], which transforms FPGA bitstreams into grayscale image formats to uncover malicious patterns. It trains a convolutional neural network on a balanced dataset of benign and malicious bitstreams. However, the classification through CNNs is inadequate for generalization across diverse FPGA architectures even when they are from the same vendor. Furthermore, they fail to support debugging, compressed, or partial bitstreams [8], which are common formats for enhancing upload efficiency or design debugging.

A fourth approach [40] presents a two-stage machine learning workflow that employs Python-based tools to extract features, reduce dimensionality through truncated singular value decomposition, and classify the bitstream with Random Forest (RF) and Support Vector Machines (SVM). However, it has a significant false positive rate (FPR) of up to 30.4%. Moreover, the classifier requires retraining for every FPGA model, which increases dataset generation and model fine-tuning workload.

Despite progress in machine learning-based security methods, these approaches require significant preprocessing, a non-trivial task. They are heavily dependent on extracted features rather than direct circuit analysis, leaving them vulnerable to dataset biases, adversarial threats, and false positives. This vulnerability arises because they do not explicitly search for the timing violations that always occur in power-wasters. Moreover, whether these solutions work at the netlist or bitstream level, the risk remains that the netlist can be leaked, violating the privacy of the user design, because the bitstream can be reverted to a netlist [41]. Our SDF-based security solution relies on comprehensive timing information that usually exists in SDF files, including propagation delays and setup/hold violations, to directly indicate malicious circuit-level changes. Table 1 summarize the limitations of the state-of-the-art and how our solution surpasses them

Table 1. Comparing our solution to the state of the art.

Metric	Ours	Ref. [8]	Ref. [38]	Ref. [39]	Ref. [40]
RO-based Attacks	✓	✓	✓	✓	✓
Hidden Attacks	✓	✓	✓	✗	✓
Cryptographic Bening-based Attacks	✓	✓	✓	✗	✗
Non-Cryptographic Benign-based Attacks	✓	✓	✗	✗	✗
Short circuit Attacks	✓	✓	✗	✗	✗
Partial Bitstreams	✓	✓	✓	✗	✗
Diagnosis Bitstreams	✓	✗	✓	✗	✗
Compressed Bitstreams	✓	✗	✓	✗	✗
Netlist-independent	✓	✗	✗	✗	✗

3 Timekeepres: Our Novel Detection Mechanism

We introduce a timing-based analysis using SDF files to detect power wasters in designs of malicious tenants in multi-tenant cloud FPGAs. Timekeepers analyzes circuit timing for self-oscillations, glitch attacks, and overclocking, offering a privacy-preserving approach that avoids examining netlists. It employs an efficient DT model for interpretation and uses a threshold-based method to evaluate individual FPGA components, such as LUTs, BRAMs, and DSPs, before determining the overall design classification. Timekeepers provides a platform-agnostic solution that adapts to diverse FPGA platforms without requiring changes to structural analysis tools, since timing characteristics reflect circuit performance rather than specific layout details. Consequently, our method remains effective even if attackers attempt to hide their power wasters.

Figure 3 shows an overview of our method. A Cloud Service Provider typically requires the submission of rtl source code or at maximum a synthesized netlist [7, 8] but not directly a bitstream. This allows te CSP to run DRC checks on the design. In order not to leak the tenant design it gets synthesized and/or implemented on a trusted shell as explained in [24]. During this step, one additional simple tcl script is needed to generate the SDF file needed for Timekeepers. The server then checks the generated SDF file to check if power wasters are present in the design or not before deploying it on the FPGA in case it finds it to be benign. Note that the SDF file is the only thing the server checks; the bitstream from the trusted shell is encrypted and cannot be read by the server directly.

3.1 Threat model

The presence of multi-tenant FPGAs in cloud environments (see Section 2.2) raises serious security issues, as attackers may exploit shared resources to compromise system integrity, reveal private data, or disrupt service availability using power wasters. Overclocking, intentional glitch production, or self-oscillations can violate timing constraints and produce unpredictable circuit behavior. Since these timing violations typically result directly from power wasters, our strategy targets timing-based analysis instead of netlist checks, which risk exposing structural details and sensitive design information.

We assume a realistic adversary in a cloud FPGA environment where users submit synthesized netlists to a CSP for implementation and bitstream generation. The attacker is a legitimate tenant who embeds power-wasting logic into their design. These attacks have been validated in practice under agreements with cloud vendors [7, 18], where the designs successfully bypassed standard design rule checks and caused measurable disruption or crashes.

In our threat model, the cloud provider scans FPGA designs submitted by tenants before deployment, ensuring that no design exhibiting malicious timing behavior is allowed on shared infrastructure. By extracting timing-related features from SDF files and analyzing them with our decision tree classifier, our model detects harmful alterations—such as abnormal path constraints, setup/hold violations, or propagation delays—that reliably indicate vulnerabilities without disclosing private design details.

3.2 Classifier proposal

A common feature of power wasters is their impact on circuit timing. Malicious modifications typically appear as self-oscillating loops, induced glitches, or overclocking, each of which disrupts the expected timing performance. These attacks manipulate timing constraints to induce errors, leak sensitive information, or destabilize the system. Unlike netlist-based assessments, which can overlook hidden modifications, timing violations, such as unexpected delays or setup / hold time violations, are inherently difficult to mask. Thus, a better detection method would focus on extracting timing-related properties that consistently reveal such malicious changes.

Recognizing the limitations of state-of-the-art detection methods relying on detailed circuit analysis, we propose an approach that relies on SDF files to capture vital timing attributes without exposing private design details. SDF files record propagation delays, setup and hold times, and path-specific constraints, all of which can

be directly affected by malicious modifications like self-oscillation, glitch attacks, and overclocking. To detect malicious circuit behavior with high precision and interpretability, we employ a Decision Tree (DT) classifier trained on timing features extracted from Standard Delay Format (SDF) files. The DT model was chosen due to its transparent hierarchical structure, which allows it to uncover complex relationships between individual timing anomalies and broader malicious behaviors such as glitch injection, overclocking, or self-oscillation. The classifier is configured with the following parameters: a maximum depth of 25 to balance complexity and generalization, a minimum of 5 samples required to split an internal node (min_samples_split=5), and a minimum of 2 samples per leaf node (min_samples_leaf=2). The gini criterion is used to measure impurity during node splits, and a fixed random_state=42 ensures consistent and reproducible results. Each SDF entry, which represents an FPGA primitive like a LUT, BRAM, or DSP, contributes a set of timing features, including propagation delays, setup/hold violations, removal/recovery time.

Algorithm 1 Threshold-based Malicious Behavior Detection from SDF Timing Data

Input: SDF_data // list of timing entries for each FPGA component
Input: threshold // fraction of components that must exhibit anomalies
Output: classification // "malicious" or "benign"
malicious_count ← 0
total_components ← length(SDF_data)
for each entry in SDF_data **do** // each entry corresponds to a specific component (e.g., LUT, BRAM, DSP)
    anomalies ← analyze_timing(entry) // assess timing characteristics: delays, setup/hold violations, etc.
    if anomalies indicate self-oscillation, glitch injection, or overclocking **then**
        malicious_flag ← 1
    else
        malicious_flag ← 0
    end if
    malicious_count ← $malicious\_count$+ malicious_flag // count flagged components
end for
fraction_malicious ← $\frac{malicious\_count}{total\_components}$
if fraction_malicious ≥ threshold **then**
    classification ← "malicious"
else
    classification ← "benign"
end if
return classification

Given that analyzing the entire circuit may obscure subtle malicious alterations, our methodology emphasizes the separate evaluation of each individual data point within the SDF file. Each entry corresponds to a specific FPGA component, such as LUTs, BRAMs, or DSP units. By carefully analyzing the timing characteristics of each component, we can detect subtle inconsistencies that could otherwise remain hidden within an overall benign design. This detailed examination reveals subtle patterns of self-oscillation, glitch injection, or overclocking, common hallmarks of power wasters, thereby enhancing our capacity to pinpoint vulnerable FPGA elements with greater precision.

Finally, we independently evaluate each FPGA primitive and then aggregate the findings using a threshold-based method. The final classification is determined by the fraction of primitives exhibiting malicious behavior rather than by a single anomaly, minimizing false positives and distinguishing real threats from minor timing

variations. This threshold method can be adapted to accommodate various attack strategies and FPGA architectures, ultimately enhancing the robustness and precision of our detection approach. Algorithm 1 details the steps of our classifier.

3.3 Dataset generation

To evaluate the effectiveness and generalizability of our proposed detection model, we curated a dataset comprising 24 unique base RTL circuits, of which 11 were malicious and 13 benign. From these base circuits, we generated 112 distinct FPGA configurations, or "tenants," by applying circuit-specific modifications. For malicious circuits, we varied the number and placement of hardware Trojans (e.g., oscillators, glitches, or timing loops). For benign configurations, we introduced variations in logic structure or composed multiple submodules to simulate realistic, non-malicious multi-tenant designs. Each configuration was then synthesized and simulated to extract its corresponding SDF file, and the resulting dataset included thousands of SDF entries (one per FPGA primitive such as LUTs, BRAMs, and DSPs), which formed the training and testing samples for our model. Importantly, we used a leave-one-circuit-out (LOCO) validation strategy: In each fold, we excluded all entries associated with one base circuit from training and used them exclusively for testing. This strict separation ensures that the model is evaluated on truly unseen circuits, rather than on slight variations of known ones, thereby testing its ability to generalize beyond the training data. The dataset was approximately balanced in terms of the number of samples derived from malicious and benign configurations, ensuring that the classifier did not develop a bias toward one class.

3.3.1 Malicious Circuits representation. The malicious designs in our dataset were designed to take advantage of hardware-level weaknesses, primarily by using ring oscillators and combinational loops to create high power usage [42]. We also included more sophisticated power wasters, including synchronous flip-flop-based oscillators [35], BRAM-based power wasters [36], and glitch amplification power wasters that generate excessive switching activity and increase power consumption [43]. Furthermore, we included four power wasters that are based on benign circuits, AES, DES, SHA, and Reed-Solomon based power wasters [8].

3.3.2 Bengin Circuits representation. For the benign designs, we assembled various recognized benchmark circuits to facilitate an unbiased and realistic assessment of our detection method. These benchmarks were obtained from the Groundhog [44], ISCAS [45], and IWLS [46] hardware benchmark collections and cryptographic implementations such as SHA and RSA encryption [8]. Moreover, we incorporated a dual core RISC-V processor design [47], providing a complex, resource-heavy circuit suitable for evaluating our model's ability to differentiate between authentic high-power circuits and harmful power-intensive attacks.

3.3.3 SDF data extraction. To create our dataset, we first synthesized, placed, and routed circuit designs inside the FPGA's partial reconfiguration region. In the tenant areas, several users can share hardware resources dynamically, simulating actual cloud FPGA configurations. We used Vivado for the synthesis and implementation, ensuring that the partial reconfiguration requirements were followed. Following the completion of the synthesis and placement, we extracted SDF data that recorded the timing characteristics of each implemented design under various operating conditions. We used the following command to streamline the extraction process using Vivado's TCL scripting interface: $write\_sdf output\_path/design\_name.sdf$.

To facilitate feature extraction and preprocessing, we systematically organized the SDF files for each of the 24 fundamental FPGA designs into a designated dataset repository following extraction. By examining these SDF files and arranging the raw timing information, we can prepare essential circuit behaviors as features for machine learning training and evaluation. By using advanced timing components, we create a dataset that effectively reflects circuit performance across various conditions, aiding in the detection of unusual variations that might indicate malicious activity. The final dataset of circuits and their samples are shown in Table 2.

Table 2. The Dataset Generated

Type	Design	#	Circuits
Malicious	AES-based	5	
	DES-based	5	
	Reed-Solomon-based	5	
	SHA-based [8]	5	
	Hidden attacks	10	
	Latch-based	5	
	MUX-based	5	
	BRAM-based	5	
	Shift-Register-based	5	
	Glitch-based	5	
Benign	ISCAS	10	
	Groundhog	3	
	IWLS	20	
	OpenCores	9	
	Own Designs	10	
	RISC-V	5	

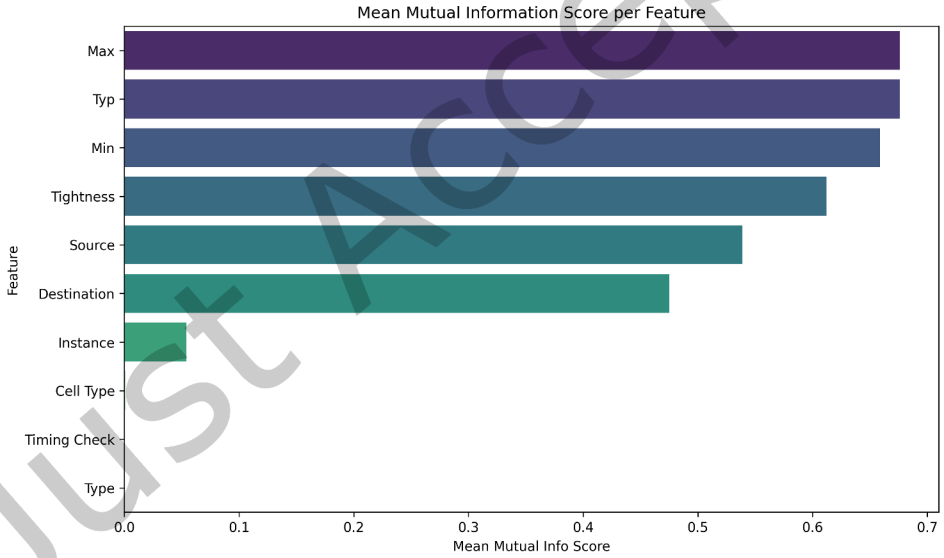


Fig. 4. Average mutual information score per feature across all datasets.

Our dataset uses standard benchmarks and publicly available attack circuits. Real tenant-submitted designs are inaccessible due to CSP confidentiality, a limitation common across the literature [8, 38, 40]. To approximate deployment conditions, we run each attack alongside at least one benign tenant, introducing inter-tenant noise. This background activity does not hinder detection and may in fact increase PDN stress, making power wasters more detectable. Additionally, since our analysis relies solely on the tenant's SDF within its partial region, timing variations from co-tenants do not affect our results.

3.3.4 Raw Data preprocessing. Upon generating the SDF files, we collected significant timing data by identifying essential patterns within the information. This led to the creation of two primary output files for every circuit: (i) a delay file that documented propagation delays with minimum, typical, and maximum values, along with information regarding the source and destination of each signal, and (ii) a timing check file that categorized constraints like setup, hold, recovery, removal, and pulse width violations. Feature selection was crucial to ensure that only the most relevant characteristics, such as timing check, type and delay values, were preserved. Furthermore, we included a significant feature: the "tightness" of the minimum-to-maximum delay values, based on the assumption that tighter constraints suggest a narrower transformation window, which could raise the risk of timing violations. It can be seen in Figure 4 the ranking of the features extracted from the SDF file where the top 5 features were only used to train the model.

3.3.5 Dataset categorize. Our first dataset showed an imbalance: some circuit classes were quite overrepresented, so the classification results might be skewed. We used the Synthetic Minority Over-sampling Technique (SMOTE) [48] to make extra samples for underrepresented groups to encourage a balanced dataset. SMOTE was exclusively introduced in the training data to allow for an unbiased assessment. Thus, the testing data remained intact. SMOTE was performed in smaller steps given the size of the dataset to avoid too much computation and memory use.

We used a leave-one-circuit-out validation approach that was used in [8] to verify the model and to ensure that every circuit was examined with new data. This approach assigns one circuit for each validation and 23 for training, to be repeated across all circuits. Using this method, we can assess the model's generalizability as well as how accurately it found previously unidentified dangerous concepts. Crucially, each circuit was added many times in the dataset to represent different design changes and attack severity, therefore establishing a strong evaluation basis. Consequently, our finalized dataset comprised 24 training and 24 testing datasets, each containing multiple variations of 23 circuits, while the validation set featured 112 unique circuit configurations.

3.3.6 Setup resource management. Since using traditional CSV storage would have led to huge file sizes, which could reach several gigabytes due to the large number of timing attributes and circuit variations, we chose the Parquet format [49]. Parquet provides an efficient columnar storage model unlike row-based text formats, considerably reducing file sizes while improving read/write performance. Given that it allowed us to analyze circuits efficiently without incurring significant computing costs, this optimization was necessary to handle timing data on a large scale.

This systematic dataset creation allowed us to create a comprehensive and flexible assessment tool to identify time-based attacks in FPGA designs. Our strategy covers a wide range of attack methods. It includes a strict validation process that raises the reliability of our detection system, making it well-suited to real-world applications in cloud FPGA settings.

4 Evaluation

Our design implementation was carried out using Vivado 2019.1 on the Xilinx Zynq UltraScale+ ZCU102 FPGA board, a well-suited high-performance platform for testing and evaluation. To analyze the extracted timing information, we developed Timekeepers in Python and executed it on a 12th Gen Intel® Core™ i9-12900 processor with 128 GiB of RAM, ensuring sufficient computational power for efficient data processing. The machine learning model was trained on a pre-balanced dataset that provided a well-distributed representation of circuit features. The training process followed a structured methodology to ensure accuracy and consistency. At the beginning of the training process, 23 of the 24 circuits were included, while one circuit was excluded to serve as an unseen dataset for validation. This leave-one-circuit-out approach was repeated for all circuits and allowed us to evaluate the model's ability to generalize. Note that leaving out one circuit means that we exclude all samples including

this circuit from the 112 samples included in our dataset. Of the 23 circuits, two datasets were prepared: a balanced dataset for training and a separate dataset for testing.

Furthermore, we applied 5-fold cross-validation to the training dataset before finalizing the model. This divides the training dataset into five equal parts, where four subsets are used for training and the fifth is reserved for evaluation. The process is repeated five times, ensuring that each subset serves as a test set once. During each independent iteration, six key performance metrics were recorded: precision, precision, recall, F1 score, false negative rate (FNR), and false positive rate (FPR).

To assess the predictive performance of the model, we used multiple evaluation metrics. Accuracy provided an overall measure of correct classifications, but given the potential imbalance in the dataset, additional metrics such as precision, recall, and F1-score were used. Precision measures the proportion of correctly identified malicious circuits, minimizing false positives, while recall evaluates the model's ability to detect actual malicious circuits, reducing false negatives. The F1-score balances these two metrics, providing a comprehensive evaluation of classification performance. In addition, the false negative rate (FNR) and the false positive rate (FPR) were included to measure the robustness of the model and compare it with the latest techniques.

To further validate the performance of the model, we generated learning curves that illustrate how the model generalizes as training data increases. The accuracy learning curve examines the gap between the training and validation accuracy, identifying potential overfitting or underfitting. A separate learning curve for the F1-score evaluates how well the model balances precision and recall as more data is used for training. Standard deviations of accuracy and F1-score were computed to assess prediction consistency.

Once the model was trained and cross-validated, it was evaluated on the testing dataset. Instead of directly classifying circuits, the model generated probability scores for each data point, which were analyzed using a thresholding technique. Several threshold values were tested to determine the optimal decision boundary, and classification performance was assessed at each threshold using accuracy, precision, recall, F1-score, FPR, FNR, and the area under the receiver operating characteristic curve (AUC-ROC). Fidelity was also computed to measure classification consistency across the dataset.

Finally, following the threshold selection, the model was validated on the previously excluded circuit to assess its ability to generalize beyond the training dataset. The same evaluation metrics were applied, ensuring that the model maintained high predictive accuracy even when encountering completely new circuits.

4.1 Cross-Validation Performance

As mentioned in Section 3 we use a DT model as our classifier. The model was configured with a maximum depth of 25, a minimum of five samples required to split a node, and at least two samples per leaf, using the Gini impurity criterion to determine the splits. These hyperparameters were fine-tuned through trial and error to strike an optimal balance between complexity and performance. A more systematic approach to hyperparameter optimization would involve using Grid Search [50] from the Scikit-Learn module, which systematically explores a predefined set of parameter values and selects the best combination through cross-validation.

The final performance metrics were obtained by averaging the results in all cross-validation iterations. As shown in Fig. 5, the mean values of these metrics were calculated for each training dataset, with one of the 24 circuit types omitted during training to serve as an unseen validation dataset (more details in Section 4.3). The column labeled "Excluded Circuit" in Fig. 5 specifies which circuit type was excluded from training. The high overall scores—99.58% accuracy, 99.7% precision, 99.45% recall, and 99.58% F1-score—demonstrate the model's effectiveness. The model training process takes approximately 7 to 9 minutes, while executing the full five-fold cross-validation procedure requires between 1 hour and 15 minutes to 2 hours and 30 minutes.

To gain deeper insights into model behavior, we analyzed learning curves, which illustrate how performance evolves as the amount of training data increases. Learning curves are crucial for determining the optimal dataset

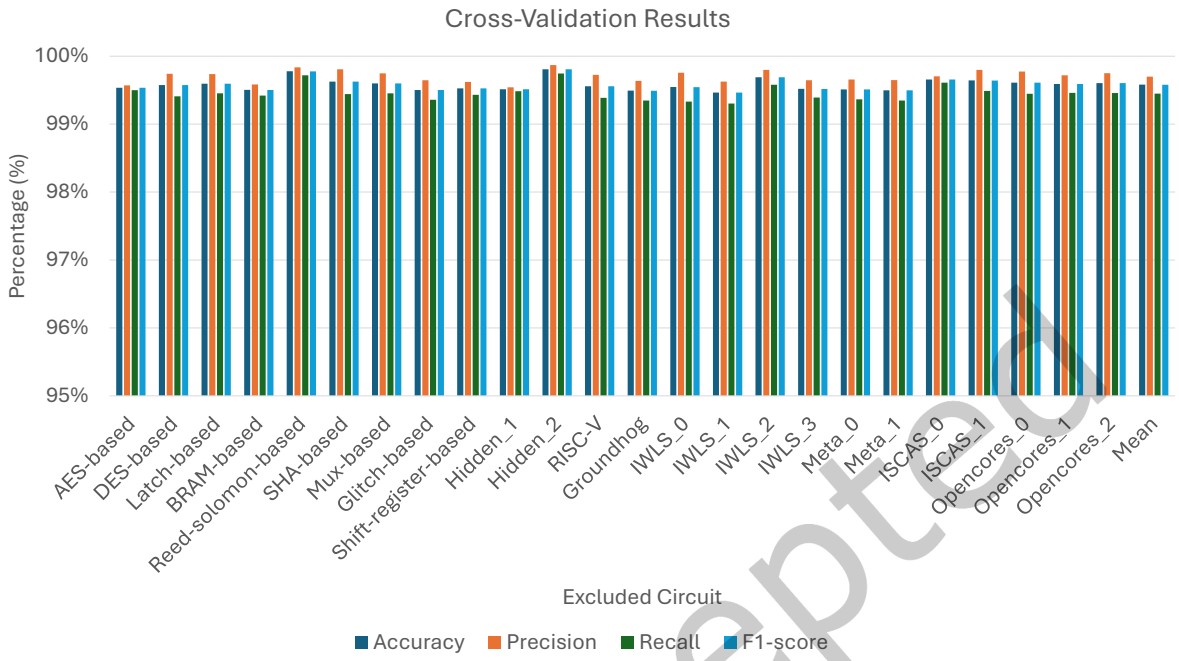


Fig. 5. Cross-validation results for different sub-datasets. In general all metrics consistently stayed over 99%

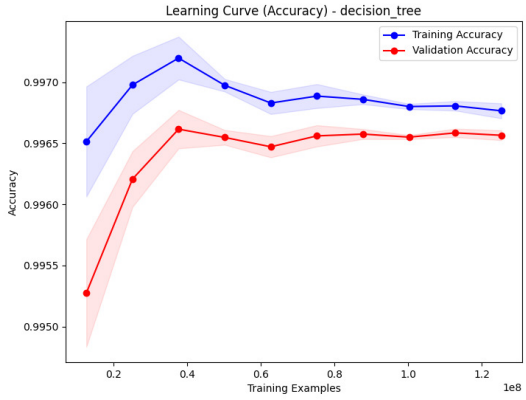
size necessary for achieving peak performance while avoiding overfitting or underfitting. We generated learning curves for each dataset and evaluated them against accuracy and F1-score. Figure 6 shows a sample of the results for different benign and malicious circuits. The observed patterns suggest that as the validation score gradually improves while the training score stabilizes, the two curves converge to a similar value. The absence of a significant gap between them indicates that the model neither overfits (where the training score is high but validation lags) nor underfits (where both scores remain low), confirming its generalization capability.

To assess the convergence speed of the model, we examined how quickly the validation metrics stabilize and approach the training metrics. Unlike typical scenarios where performance increases steeply from a lower baseline, our accuracy and F1-score values start above 99.5%, leaving a very narrow range for visible improvement. The maximum validation accuracy reached approximately 99.65%, with a gradual rise observed until around 30% of the training samples were used. After that point, both the training and validation curves showed minimal change and began to converge more tightly, indicating rapid stabilization. This suggests that our model achieves near-peak performance early in training and continues to generalize well as more data is added.

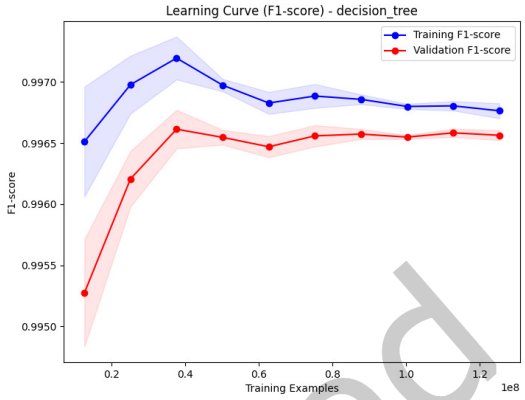
Therefore, instead of defining convergence by a percentage of the final metric value (which is nearly saturated), we evaluate convergence as the point where the validation curve flattens and the gap with the training curve becomes negligible — which in our case, occurred at approximately 30–40% of the training data. These observations confirm not only the model's high performance but also its fast convergence speed and efficient learning behavior. The same patterns hold for other circuits; we are not showing them to avoid redundancy. Note that as this is just the training with the cross-validation we did not include the thresholds mechanism yet for Timekeepers.

4.2 Performance on testing dataset

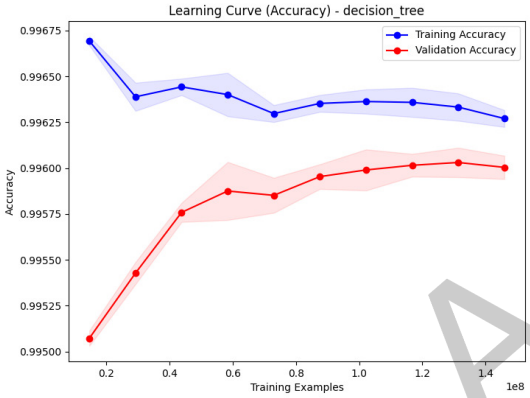
Next we assess the model's ability to generalize to previously unseen circuits. For each of the 24 iterations of our evaluation, we conducted tests on the separate test dataset that, while distinct from the training set, remained



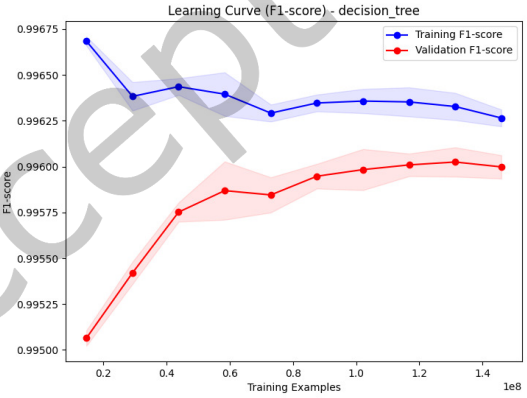
(a) Accuracy for dataset with $ISCAS_0$ excluded



(b) F1-score for dataset with $ISCAS_0$ excluded



(c) Accuracy for dataset with Mux-based excluded



(d) F1-score for dataset with Mux-based excluded

Fig. 6. Learning curve for different sub-datasets excluded. No over or under fitting occurs.

within the same domain. Since the model was trained on similar data distributions, achieving high performance on these test sets suggests that it has effectively generalized and can reliably classify circuits as either benign or malicious.

The results, summarized in Table 3, reinforce the stability of the model, demonstrating consistently high performance in all key metrics. To enhance the depth of our evaluation, we supplemented standard metrics, accuracy, precision, recall, and F1 score, with additional measures, including AUC-ROC, fidelity, false negative rate (FNR) and false positive rate (FPR). In particular, the model achieved an AUC-ROC score of 0.999, which signifies its exceptional ability to distinguish between malicious and benign circuits. A near-perfect ROC value indicates that the model effectively separates the two classes, minimizing the likelihood of misclassifications. The consistently low FNR confirms that malicious circuits are rarely mislabeled as benign, which is crucial for security-sensitive applications where unidentified rogue circuits could introduce severe vulnerabilities. Furthermore, the low FPR ensures that benign circuits are not mistakenly classified as malicious, thus preventing unnecessary

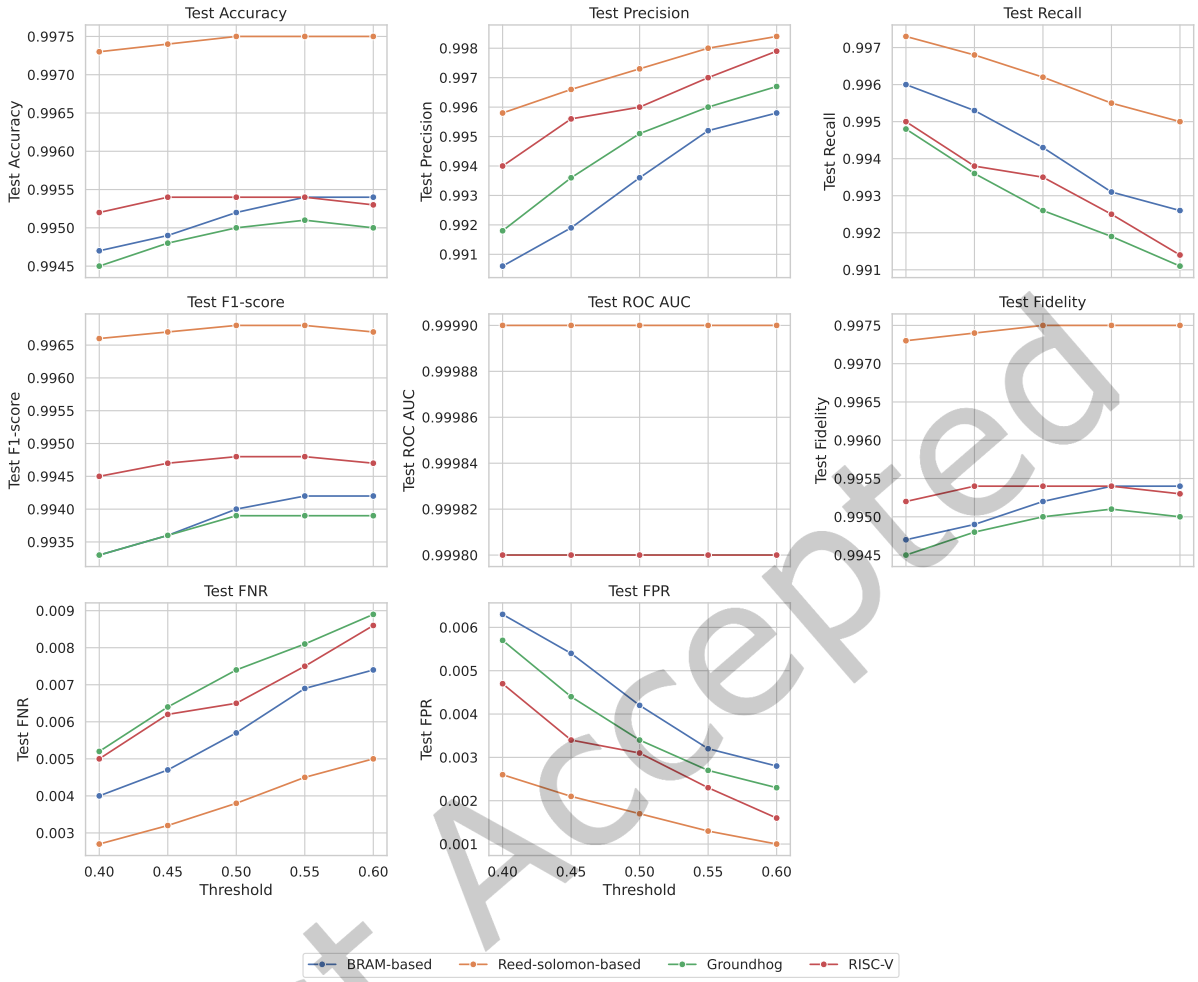


Fig. 7. Variation of test metrics across different thresholds for all circuit types.

interventions or false alarms. The fidelity score remains high at 99.6%, reflecting the model's strong ability to accurately predict true classifications and reinforcing its reliability.

As this is the testing dataset, we empirically derived the best threshold for each of the subsets. We aim to have all metrics perform above 0.95. From this analysis, we determined that the mean classification threshold is 0.548. This means that a circuit is classified as malicious if more than 54.8% of its extracted SDF-based features indicate malicious behavior. It can also be seen from Figure Fig. 8 the mean results for the category of the proposed circuit. With an accuracy of 99.6%, precision of 99.7%, recall of 99.4%, F1-score of 99.5%, and fidelity of 99.6%, the overall testing results further validate the model's robustness and reliability. The maximum was 0.6, going higher than this led to an increase of the FPR.

The test metrics Fig. 7 presents a comprehensive view of how the model performs on testing data across different thresholds for each circuit type. It covers performance metrics including accuracy, precision, recall, F1-score, ROC AUC, fidelity, false negative and positive rates, and predicted malicious percentage. This figure

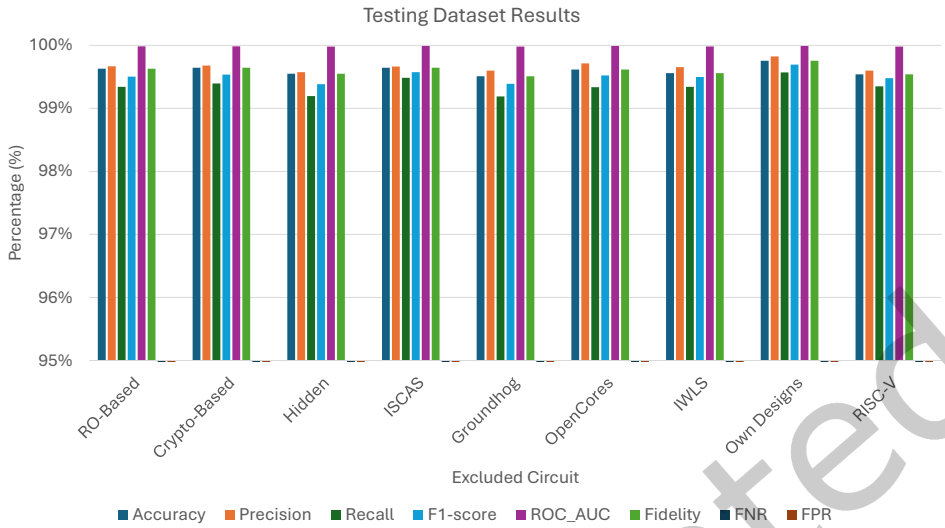


Fig. 8. Testing dataset results for different Circuit types

serves as a confirmation of the model's ability to generalize well under optimal thresholds derived from validation. In particular, circuits that maintain high precision and recall simultaneously suggest strong discriminative power, while those exhibiting divergent curves reflect imbalanced misclassification risks. The ROC AUC trends reinforce the robustness of the classification boundaries, and fidelity indicates the degree of agreement between the model predictions and the actual results.

4.3 Performance on "Unseen" Validation dataset

To further assess the model's ability to generalize beyond the training and testing datasets, we extended our analysis to evaluate its performance on entirely or partially novel circuits. This step was critical in determining how well the model performs when encountering circuits that were never included in its training phase. As previously mentioned, our dataset contained 24 circuit types, with 23 used for training and testing in each iteration while leaving one circuit completely unseen for validation. This process was repeated for all 24 circuits to ensure that each one was omitted at least once. The objective was to determine whether the model could accurately infer the delay and timing constraints of the unseen circuit and classify it as malicious or benign. To accomplish this, we used two thresholding strategies: (i) the mean threshold of 54.8% and (ii) the maximum threshold of 60%. Using the maximum threshold, we achieved the best results.

The validation metrics figure Fig. 9 illustrates how various evaluation scores (accuracy, precision, recall, F1-score, fidelity, false negative rate, and false positive rate) vary with different threshold values for each circuit. As shown, the behavior of these metrics in response to threshold tuning is circuit-dependent, highlighting the trade-offs that exist between sensitivity and specificity. For example, increasing the threshold often reduces the false positive rate, but may increase the false negative rate, especially for circuits with less separable distributions. The F1-score and fidelity curves help identify the threshold values that achieve a balanced trade-off, where both positive and negative classifications are handled effectively. Moreover, the precision remaining stable in some circuits across thresholds may indicate robustness in detecting malicious behaviors, while sharp changes in recall expose thresholds where detection performance sharply drops. The pred_perc curve offers an interpretable

Table 3. Evaluation of Testing Datasets

Excluded Circuit	Best Threshold	Accuracy	Precision	Recall	F1-score	ROC	Fidelity	FNR	FPR
AES-based	0.6	0.996	0.996	0.994	0.995	0.999	0.996	0.006	0.002
DES-based	0.55	0.996	0.996	0.992	0.994	0.999	0.996	0.008	0.002
Latch-based	0.55	0.996	0.997	0.993	0.995	0.999	0.996	0.007	0.002
BRAM-based	0.6	0.995	0.996	0.993	0.994	0.999	0.995	0.007	0.003
Reed-solomon-based	0.5	0.998	0.997	0.996	0.997	0.999	0.998	0.004	0.002
SHA-based	0.55	0.996	0.998	0.993	0.995	0.999	0.996	0.007	0.002
Mux-based	0.55	0.996	0.997	0.993	0.995	0.999	0.996	0.007	0.002
Glitch-based	0.55	0.995	0.996	0.992	0.994	0.999	0.995	0.008	0.003
Shift-register-based	0.6	0.996	0.996	0.992	0.994	0.999	0.996	0.008	0.002
Hidden_1	0.6	0.996	0.996	0.992	0.994	0.999	0.996	0.008	0.002
Hidden_2	0.55	0.998	0.998	0.997	0.997	0.999	0.998	0.004	0.001
RISC-V	0.5	0.995	0.996	0.994	0.995	0.999	0.995	0.006	0.003
Groundhog	0.55	0.995	0.996	0.992	0.994	0.999	0.995	0.008	0.003
IWLS_0	0.5	0.996	0.997	0.993	0.995	0.999	0.996	0.007	0.002
IWLS_1	0.5	0.995	0.995	0.992	0.994	0.999	0.995	0.008	0.004
IWLS_2	0.55	0.997	0.998	0.996	0.997	0.999	0.997	0.004	0.002
IWLS_3	0.55	0.995	0.996	0.993	0.994	0.999	0.995	0.007	0.003
Meta_0	0.6	0.995	0.997	0.991	0.994	0.999	0.995	0.009	0.002
Meta_1	0.5	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.000
ISCAS_0	0.5	0.997	0.996	0.996	0.996	0.999	0.997	0.004	0.003
ISCAS_1	0.55	0.996	0.997	0.994	0.996	0.999	0.996	0.006	0.002
Opencores_0	0.55	0.996	0.997	0.993	0.995	0.999	0.996	0.007	0.002
Opencores_1	0.55	0.996	0.997	0.994	0.995	0.999	0.996	0.006	0.002
Opencores_2	0.55	0.996	0.997	0.993	0.995	0.999	0.996	0.007	0.002
Mean	0.548	0.996	0.997	0.994	0.995	0.999	0.996	0.048	0.044

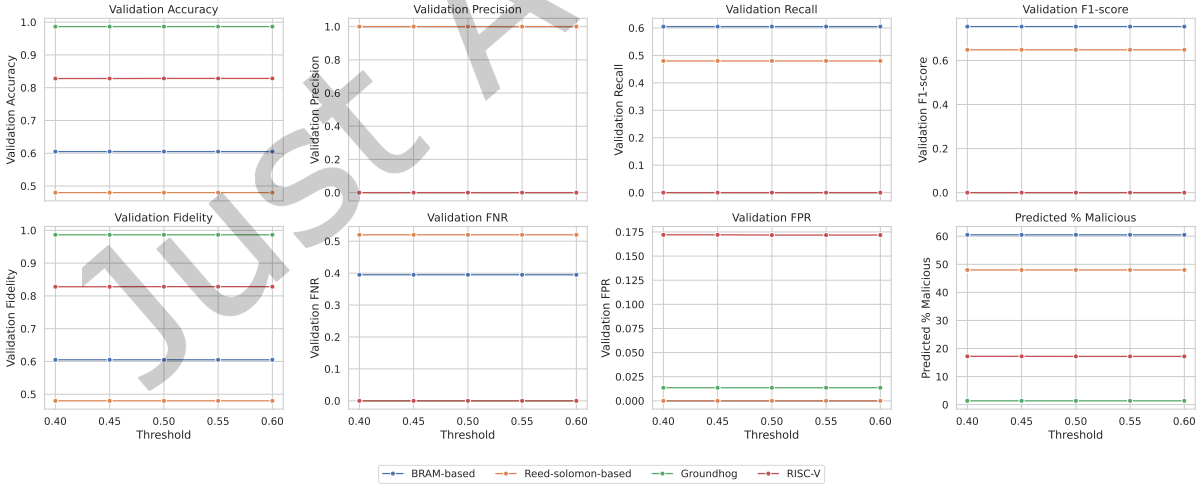


Fig. 9. Variation of validation metrics across different thresholds for all circuit types.

indication of how aggressively the model flags malicious behavior, which is critical in sensitive applications like

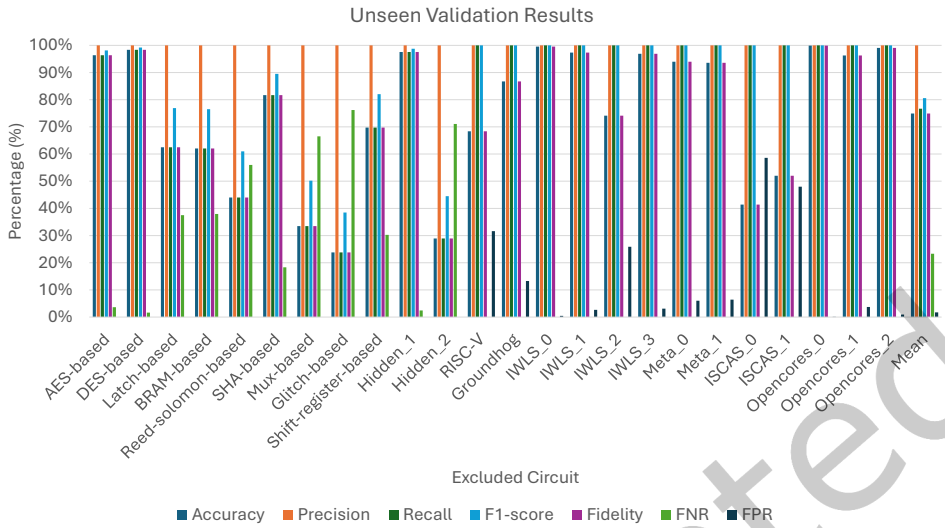


Fig. 10. Validation Metrics for unseen dataset

hardware security. These patterns provide valuable insight into how circuit-specific characteristics influence model generalization and where optimal operating points lie for validation.

We applied the same scoring metrics used during the testing phase. The results, presented in Fig. 10, provide key insights. In general, the model demonstrated near-perfect performance in many circuits, and most metrics, particularly precision, consistently reached 100%. The precision was consistently high, affirming the model's reliability in detecting threats with positive predictions. However, recall varied, averaging 76.7%, affecting F1-scores and fidelities. This aligns with previous findings in [8] where outliers also existed.

Table 4. Classification metrics for different circuits on Virtex 7 FPGA

Circuit	Accuracy	Precision	Recall	F1-score	Fidelity	FNR	FPR	Prediction	Malicious %
AES-Based	0.6681	1	0.6681	0.7867	0.6681	0.332	0	Malicious 2/4	66.81%
Latch-Based	0.6877	1	0.6877	0.8124	0.6877	0.3123	0	Malicious	68.77%
Shift-register-based	0.7826	1	0.7826	0.8780	0.7826	0.2174	0	Malicious	78.26%
Hidden attacks0	0.9928	1	0.9928	0.9964	0.9928	0.0072	0	Malicious	99.28%
BRAM-Based	0.5656	1	0.5656	0.7225	0.5656	0.4344	0	Malicious	56.56%
Glitch-Based	0.2116	1	0.2116	0.3435	0.2116	0.7884	0	Benign	21.16%
Hidden attack1	0.4686	1	0.4686	0.6256	0.4686	0.5314	0	Benign	46.86%
SHA-Based	0.3321	1	0.3321	0.4529	0.3321	0.6679	0	Benign 1/2	33.21%
Reed-Solomon-Based	0.4206	1	0.4206	0.5731	0.4206	0.5794	0	Benign 1/2	42.06%
DES-Based	0.9750	1	0.9750	0.9873	0.9750	0.0250	0	Malicious	97.5%
Groundhog	0.9871	1	1	1	0.9871	0	0.0129	Benign	1.29%
IWLS2	0.5946	1	1	1	0.5946	0	0.4054	Benign	40.54%
IWLS3	0.7043	1	1	1	0.7043	0	0.2957	Benign	29.57%
IWLS0	0.8153	1	1	1	0.8153	0	0.1847	Benign	18.47%
IWLS1	0.7569	1	1	1	0.7569	0	0.2431	Benign	24.31%
RISC-V	0.5737	1	1	1	0.5737	0	0.4263	Benign	42.63%
ISCAS0	0.5930	1	1	1	0.5930	0	0.4070	Benign 3/4	40.70%
ISCAS1	0.6375	1	1	1	0.6375	0	0.3625	Benign	36.25%
Own Designs0	0.5425	1	1	1	0.5425	0	0.4575	Benign 3/4	45.75%
Own Designs1	0.5076	1	1	1	0.5076	0	0.4924	Benign 2/4	49.24%

To evaluate the portability and robustness of our classifier, we conducted a cross-platform test using circuits implemented on the Virtex-7 FPGA. Specifically, we rebuilt 54 of the original 112 circuits—previously used for evaluation on the Zynq FPGA—on the Virtex-7 platform and extracted corresponding SDF files. The testing methodology remained consistent with the unseen circuit validation approach: for each circuit, the model was trained on all remaining circuits built on the Zynq FPGA, excluding the one under test. The results, summarized in Table Table 4, present the average classification metrics for each circuit, including accuracy, precision, recall, F1-score, fidelity, false negative rate (FNR), and false positive rate (FPR). The classifier's performance on Virtex-7 circuits closely mirrors the results observed on Zynq-based circuits, demonstrating consistent generalization across FPGA platforms. Notably, the classifier achieved an overall prediction accuracy of 70.3% while using 54.8% as a threshold for analysing the Malicious Designs, effectively identifying the nature of each tested circuit.

4.4 State of the art comparison

Table 5. Comparison with the state of the art (cross-validation)

Metrics	Num of Folds	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	FPR (%)	FNR (%)
Ours	5	99.5%	99.7%	99.4%	99.5%	0.53%	0.29%
Meta-Scanner [8]	10	98%*	98.4%	97.6%	97.8%	0.75%	1.8%
CNN [39]	5	99.2%	99.2%*	99.2%*	99.2%*	99.2%*	99.2%*

To ensure a fair and consistent evaluation of our proposed detection model, all experiments in this study were carried out using the same dataset, generated from 24 base RTL designs. We adopted a leave-one-circuit-out (LOCO) validation strategy, in which each circuit was completely excluded from training and reserved solely for testing, to evaluate the model's ability to generalize to entirely unseen designs. This approach guarantees that each model variation and threshold setting was assessed on a consistent and identical dataset structure. For comparison with state-of-the-art methods, we refer to the reported performance metrics in the existing literature, as we were unable to reproduce their experiments due to the lack of publicly available datasets. However, we aligned our dataset composition with theirs by including a diverse set of circuits, such as ISCAS benchmarks and AES variants, which are commonly used in hardware security studies. Although there may be minor differences in the specifics of the dataset, this alignment allows for a meaningful comparative analysis that highlights the robustness and advantages of our SDF-based detection approach relative to net-list-based methods.

To estimate the missing scoring metrics consistently across all models, we adopted standard classification metric formulas and made a common assumption that the total number of test samples is 2000, with an equal split of 1000 benign and 1000 malicious samples. The metrics were calculated using the following standard definitions: Precision = $\frac{TP}{TP+FP}$, Recall = $\frac{TP}{TP+FN}$, F1-score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision+Recall}}$, FNR = $\frac{FN}{FN+TP}$, and FPR = $\frac{FP}{FP+TN}$. For estimating ROC, we used the approximation: ROC $\approx 1 - \frac{\text{FPR+FNR}}{2}$. Fidelity was derived using the formula:

$$\text{Fidelity}^{-} = \frac{1}{D} \sum_{i=1}^{D} \left| \mathbb{1}(\hat{y}_i = y_i) - \mathbb{1}(\hat{y}_i^{G_S} = y_i) \right|,$$

where D is the number of test samples, and $\mathbb{1}$ is the indicator function. These assumptions and formulas enabled consistent and fair estimation of the unreported metrics across all models.

As summarized in Table 5, our method demonstrates superior performance compared to two of the top models in the field, even while using a reduced number of K-folds during cross-validation. This is a notable advantage, as it suggests that our model achieves higher efficiency and generalization with fewer training iterations, reducing computational complexity while maintaining strong predictive accuracy.

Table 6. Comparison with the state of the art (testing)

Metrics	Ours	Meta-Scanner [8]	MaliGNNoma [38]	CNN [39]	Learning [40]
Machine	i9 + 125GiB RAM	Ryzen 6-core	Xeon 10-core	–	–
Model type	Decision trees	RF/DT	GNN	CNN	RF + SVM
Training time (min)	7:26	2 (DT)	6h	–	< 2 (RF/SVM)
prediction time	0.098s	5ms	0.05s	–	8ms
Data type	Timing parameters	Bitstream	Netlist graph	Bitstream	Bitstream
Dataset size	112	475	115	314	229
Accuracy	99.6%	97.9%	98.24%	96.4%	84%
Precision	99.7%	100%	97.88%	95.76%*	72.4%
Recall	99.4%	96.3%	95.8%	97.1%*	95.5%*
F1-score	99.5%	98.12%*	98.19%	96.42%*	80.7%
FNR	4.8%	2.1%	4.2%*	2.9%*	4.5%
FPR	4.4%	1.5%	2.07%*	4.29%	30.4%
ROC	99.9%	98.2%*	96.87%*	96.41%*	82.55%*

When assessing performance on previously unseen datasets where the data was not explicitly used for training but shares structural similarities with the training set, we ensured that our evaluation incorporated all key performance metrics employed in leading methodologies. This approach enabled a standardized, direct comparison, allowing us to fairly assess how well our classifier operates under similar conditions.

To ensure a uniform and meaningful comparison across different models in our study, we extended the reported evaluation metrics by estimating the missing ones using standard classification formulas. Since most of the papers only reported partial metrics (e.g., Accuracy without Precision or Recall), we assumed a consistent evaluation set consisting of 4000 samples—split evenly between malicious and benign cases—while respecting the fact that the "dataset size" in the table refers to the number of distinct circuits, not individual data points. For missing values such as Precision, Recall, F1-score, False Positive Rate (FPR), False Negative Rate (FNR), and ROC, we derived estimates using well-established metric relationships. For example, we computed F1-scores using the harmonic mean of Precision and Recall, while Recall and Precision were reconstructed where one was missing but the F1-score was available. When both Recall and FPR or FNR were known, we inferred the ROC score using the standard approximation ROC $=1-((FPR+FNR)/2)$. These estimations were made cautiously to ensure consistency with the originally reported metrics, enabling a complete, fair, and comparative performance landscape across all the evaluated methods. As presented in Table 6, our model surpasses all leading classifiers across most evaluation metrics, reinforcing its effectiveness in identifying malicious circuits and it also falls in the same timing range as the state of the art for training and predicting the circuit nature.

Among comparable models, MaliGNNoma and Meta-Scanner exhibit results closest to ours, with minor differences in performance. However, our approach outperforms both in most metrics, showcasing its enhanced ability to generalize across diverse circuit structures. A key distinguishing factor of our method is the inclusion of the Receiver Operating Characteristic (ROC) metric, which has been absent from all previous works. By integrating ROC analysis, we provide a more in-depth evaluation of our model's discriminative power, further affirming its reliability in distinguishing between malicious and benign circuits. This additional evaluation strengthens our argument that our model is not only competitive but also introduces a novel perspective to the field by offering a more comprehensive performance assessment—an aspect previously overlooked in state-of-the-art methods. Moreover, our solution is the only one not using sensitive data from the netlist nor the bitstream to detect the existence of power wasters so it maintains same security level without sacrificing privacy of the designs of the tenants.

Finally, we compare our validation against "unseen" circuits to the performance of Meta-Scanner [8]. It is the only work that conducted such an experiment; others relied only on cross-validation. As Table 7 shows, we have comparable accuracy to them and slightly better recall.

Table 7. Comparison with the state of the art (leave one circuit out)

Metrics	Ours	Meta-Scanner [8]
Accuracy	74.9%	77.3%
Recall	76.7%	67.5%

4.5 Future work

Future directions for this research offer numerous opportunities to improve detection accuracy, adaptability, and practical deployment in real-world FPGA environments. One promising avenue is the adoption of more advanced machine learning techniques—such as deep neural networks, ensemble learning, or hybrid models—that can better capture complex relationships in timing data and generalize across diverse circuit designs. Expanding the dataset to include a broader range of FPGA configurations, architectural styles, and usage patterns would significantly enhance the model's robustness, particularly in handling emerging threats in multi-tenant cloud infrastructures.

Existing research in FPGA security has introduced a variety of innovative detection frameworks, each tackling the problem from different angles. MaliGNNoma [38] leverages graph neural networks (GNNs) to identify structural anomalies in FPGA circuits by analyzing netlist topologies, making it particularly effective at capturing unauthorized modifications embedded within circuit connectivity. However, it lacks sensitivity to timing-based behaviors, which can be a key indicator of stealthy attacks like glitch injection or overclocking. Our work focuses on SDF-based timing analysis, capturing setup/hold violations, propagation delays, and path-specific metrics to expose subtle timing anomalies caused by malicious modifications. Integrating these rich features into existing detection schemes as dual classification stages for the proposed design, as primary and secondary stages, can ensure the nature of the proposed design.

In parallel, future work should include building explainable AI tools within the detection framework. These tools would help highlight which parts of a design—specific cells, signals, or timing paths—contributed to the classification decision. Such insights would make the system more transparent, helping designers understand and address flagged issues, whether they're false positives or real vulnerabilities. By combining powerful classification capabilities with model interpretability and broader coverage of FPGA design variations, this line of research can evolve into a truly scalable and intelligent security solution for cloud-based FPGA systems.

4.6 Discussion

The results presented in this work demonstrate the effectiveness of leveraging timing behavior, as extracted from SDF files, to identify power-wasting FPGA designs in multi-tenant cloud environments. Unlike existing approaches that rely on structural analysis or bitstream-level features, Timekeepers focuses on the one element that all power wasters inevitably affect—timing. By evaluating propagation delays, setup/hold violations, and related timing characteristics, our method generalizes well across diverse attack types and circuit structures, while avoiding exposure of sensitive design information.

The SDF-based detection approach offers multiple advantages. First, it abstracts away from the physical layout or specific resource mappings of the design, making the detection process architecture independent and resilient to obfuscation techniques. Second, it preserves intellectual property (IP) confidentiality, since it does not require access to the netlist or bitstream. This is particularly important in cloud scenarios where tenant privacy is a key concern. Third, Timekeepers enables per-component analysis of timing behavior, allowing for fine-grained detection that can pinpoint localized malicious activity, which might be obscured in full-design level statistics.

While Timekeepers achieved consistently high performance across cross-validation and testing datasets, some performance degradation was observed when validating on entirely unseen circuits. In particular, the recall values were slightly lower, which suggests that certain circuit structures or timing profiles may not be fully

captured during training. This is a common challenge in timing-based analysis, as subtle design variations can affect delay profiles. Nevertheless, the model remained effective in most cases and successfully identified a wide range of power wasters, including those based on cryptographic cores and memory patterns.

It is also worth noting that decision trees were selected for their interpretability and hardware-friendliness. However, more advanced models or hybrid classifiers could be explored to further enhance performance, especially in low-recall edge cases. This would require careful balancing to maintain the efficiency and transparency offered by the current model. Overall, the Timekeepers framework offers a privacy-preserving, scalable, low-overhead, and architecture-agnostic solution for pre-deployment security checks in cloud FPGAs. It complements existing runtime defenses by flagging high-risk designs before they are loaded onto the fabric, and sets the foundation for future work in timing-based hardware security analysis.

Note that we did not run experiments looking at the power utilization when the attack is running. However, we had ground truth experiments where the attack is run on the FPGA and it crashes. Furthermore, in [51] the authors have run experiments showing the exact power usage for different types of power wasters showing that it is high.

5 Conclusion

This work presents a novel SDF-driven security framework for detecting malicious FPGA circuits in cloud environments. By utilizing timing-based analysis and machine learning models, our approach offers a systematic method to evaluate FPGA configurations for potential security risks. The experimental results demonstrate that setup and hold violations provide critical insights into circuit behavior, enabling effective classification of benign and malicious designs. Compared to conventional bitstream verification and encryption-based security measures, our method introduces a non-intrusive, data-driven approach that is adaptable to various FPGA architectures. Furthermore, the integration of machine learning enables automated threat detection, reducing the need for manual intervention and enhancing scalability.

Acknowlwdgements

This work was partially funded by the Federal Ministry of Research, Technology, and Space BMFTR, as part of the MANNHEIM-CeCaS project (grant: 16ME0817) and the DI-EDAI project (grant: 16ME0990K).

References

[1] Ghada Dessouky, Ahmad-Reza Sadeghi, and Shaza Zeitouni. 2021. SoK: Secure FPGA Multi-Tenancy in the Cloud: Challenges and Opportunities. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. 487–506. DOI: http://dx.doi.org/10.1109/EuroSP51992.2021.00040

[2] Michael Guilherme Jordan, Guilherme Korol, Mateus Beck Rutzig, and Antonio Carlos Schneider Beck. 2021. MUTECO: A Framework for Collaborative Allocation in CPU-FPGA Multi-tenant Environments. In *2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*. 1–6. DOI: http://dx.doi.org/10.1109/SBCCI53441.2021.9529992

[3] Xiangmeng Long, Baohua Liu, Feng Jiang, Qingjie Zhang, and Xiaoli Zhi. 2020. FPGA virtualization deployment based on Docker container technology. In *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*. 473–476. DOI: http://dx.doi.org/10.1109/ICMCCE51767.2020.00109

[4] Hassan Nassar, Hanna AlZughbi, Dennis R. E. Gnad, Lars Bauer, Mehdi B. Tahoori, and Jörg Henkel. 2021. LoopBreaker: Disabling Interconnects to Mitigate Voltage-Based Attacks in Multi-Tenant FPGAs. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9. DOI: http://dx.doi.org/10.1109/ICCAD51958.2021.9643485

[5] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi B. Tahoori. 2021. An Inside Job: Remote Power Analysis Attacks on FPGAs. *IEEE Design & Test* 38, 3 (2021), 58–66. DOI: http://dx.doi.org/10.1109/MDAT.2021.3063306

[6] Nithyashankari Gummidipoondi Jayasankaran, Hao Guo, Satwik Patnaik, Jeyavijayan, Rajendran, and Jiang Hu. 2023. Securing Cloud FPGAs Against Power Side-Channel Attacks: A Case Study on Iterative AES. (2023). arXiv:cs.CR/2307.02569 https://arxiv.org/abs/2307.02569

[7] Tuan La, Khoa Pham, Joseph Powell, and Dirk Koch. 2021. Denial-of-Service on FPGA-based Cloud Infrastructures — Attack and Defense. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 3 (Jul. 2021), 441–464. DOI: http://dx.doi.org/10.46586/tches.

v2021.i3.441-464

[8] Hassan Nassar, Jonas Krautter, Lars Bauer, Dennis Gnad, Mehdi Tahoori, and Jörg Henkel. 2024. Meta-Scanner: Detecting Fault Attacks via Scanning FPGA Designs Metadata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (Nov. 2024), 3443–3454. DOI: http://dx.doi.org/10.1109/TCAD.2024.3443769

[9] Hassan Nassar, Philipp Machauer, Dennis R. E. Gnad, Lars Bauer, Mehdi B. Tahoori, and Jörg Henkel. 2024. Covert-Hammer: Coordinating Power-Hammering on Multi-tenant FPGAs via Covert Channels. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '24)*. Association for Computing Machinery, New York, NY, USA, 43. DOI: http://dx.doi.org/10.1145/3626202.3637613

[10] Inna Kolesnyk, Artem Perepelitsyn, and Vitaliy Kulanov. 2017. Providing of FPGA Resources as a Service : Technologies , Deployment and Case-Study. In *Proceedings of the 13th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer, ICTERI, PhD Symposium 2017, Kyiv, Ukraine, May 15-18, 2017 (CEUR Workshop Proceedings)*, Vol. 1844. CEUR-WS.org, 63–68.

[11] Marvin Damschen, Martin Rapp, Lars Bauer, and Jörg Henkel. 2020. *i-Core: A Runtime-Reconfigurable Processor Platform for Cyber-Physical Systems*. Springer International Publishing, Cham, 1–36. DOI: http://dx.doi.org/10.1007/978-3-030-16949-7_1

[12] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E.M. Panainte. 2004. The MOLEN polymorphic processor. *IEEE Trans. Comput.* 53, 11 (2004), 1363–1375. DOI: http://dx.doi.org/10.1109/TC.2004.104

[13] Amr Hassan, Hassan Mostafa, Hossam A. H. Fahmy, and Yehea Ismail. 2017. Exploiting the Dynamic Partial Reconfiguration on NoC-Based FPGA. In *2017 New Generation of CAS (NGCAS)*. 277–280. DOI: http://dx.doi.org/10.1109/NGCAS.2017.78

[14] Khaled Salah. 2017. An area efficient multi-mode memory controller based on dynamic partial reconfiguration. In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 328–331. DOI: http://dx.doi.org/10.1109/IEMCON.2017.8117122

[15] Xilinx 2015. *Partial Reconfiguration User Guide*. Xilinx.

[16] Miriam Leeser, Suranga Handagala, and Michael Zink. 2021. FPGAs in the Cloud. *Computing in Science & Engineering* 23, 6 (Nov. 2021), 72–76. DOI: http://dx.doi.org/10.1109/MCSE.2021.3127288 Conference Name: Computing in Science & Engineering.

[17] Oliver Sinnen Feng Yan, Andreas Koch. A survey on FPGA-based accelerator for ML models. https://arxiv.org/abs/2412.15666. ([n. d.]).

[18] Hassan Nassar, Philipp Machauer, Lars Bauer, Dennis Gnad, Mehdi Tahoori, and Jörg Henkel. 2025. DoS-FPGA: Denial of Service on Cloud FPGAs via Coordinated Power Hammering. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24)*. Association for Computing Machinery, New York, NY, USA, Article 119, 9 pages. DOI: http://dx.doi.org/10.1145/3676536.3676843

[19] Zhuangdi Zhu, Alex X. Liu, Fan Zhang, and Fei Chen. 2021. FPGA Resource Pooling in Cloud Computing. *IEEE Transactions on Cloud Computing* (2021).

[20] Jayeeta Chaudhuri, Hassan Nassar, Dennis R. E. Gnad, Jorg Henkel, Mehdi B. Tahoori, and Krishnendu Chakrabarty. 2025. FLARE: Fault Attack Leveraging Address Reconfiguration Exploits in Multi-Tenant FPGAs. (2025). arXiv:cs.CR/2502.15578 https://arxiv.org/abs/2502.15578

[21] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia, Philippe Loubet Moundi, and Francis Olivier. 2020. Remote side-channel attacks on heterogeneous SoC. In *Smart Card Research and Advanced Applications: 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11–13, 2019, Revised Selected Papers 18*. Springer, 109–125.

[22] Jakub Breier and Xiaolu Hou. 2022. How Practical Are Fault Injection Attacks, Really? *IEEE Access* 10 (2022), 113122–113130. DOI: http://dx.doi.org/10.1109/ACCESS.2022.3217212

[23] Jayeeta Chaudhuri, Hassan Nassar, Dennis R.E. Gnad, Jörg Henkel, Mehdi B. Tahoori, and Krishnendu Chakrabarty. 2024. Hacking the Fabric: Targeting Partial Reconfiguration for Fault Injection in FPGA Fabrics. In *2024 IEEE 33rd Asian Test Symposium (ATS)*. 1–6. DOI: http://dx.doi.org/10.1109/ATS64447.2024.10915413

[24] Shaza Zeitouni, Jo Vliegen, Tommaso Frassetto, Dirk Koch, Ahmad-Reza Sadeghi, and Nele Mentens. 2021. Trusted Configuration in Cloud FPGAs . In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE Computer Society, 233–241. DOI: http://dx.doi.org/10.1109/FCCM51124.2021.00036

[25] Hassan Nassar, Jeferson Gonzalez-Gomez, Varun Manjunath, Lars Bauer, and Jorg Henkel. 2025. Through Fabric: A Cross-world Thermal Covert Channel on TEE-enhanced FPGA-MPSoC Systems. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference (ASPDAC '25)*. Association for Computing Machinery, New York, NY, USA, 461–467. DOI: http://dx.doi.org/10.1145/3658617.3697767

[26] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. 2021. Remote and Stealthy Fault Attacks on Virtualized FPGAs. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1632–1637. DOI: http://dx.doi.org/10.23919/DATE51398.2021.9474140

[27] George Provelengios, Daniel Holcomb, and Russell Tessier. 2020. Power distribution attacks in multitenant FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 12 (2020), 2685–2698.

[28] Jiliang Zhang and Gang Qu. 2019. Recent Attacks and Defenses on FPGA-based Systems. *ACM Trans. Reconfigurable Technol. Syst.* 12, 3, Article 14 (Aug. 2019), 24 pages. DOI: http://dx.doi.org/10.1145/3340557

[29] Jonas Krautter, Dennis RE Gnad, and Mehdi B Tahoori. 2018. FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 44–68.
[30] Amit Mazumder Shuvo, Tao Zhang, Farimah Farahmandi, and Mark Tehranipoor. 2023. A comprehensive survey on non-invasive fault injection attacks. *Cryptology ePrint Archive* (2023).
[31] Mark Zhao and G Edward Suh. 2018. FPGA-based remote power side-channel attacks. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 229–244.
[32] Kenneth M Zick, Meeta Srivastav, Wei Zhang, and Matthew French. 2013. Sensing nanosecond-scale voltage attacks and natural transients in FPGAs. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. 101–104.
[33] Linda L Shen, Ibrahim Ahmed, and Vaughn Betz. 2019. Fast voltage transients on FPGAs: Impact and mitigation strategies. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 271–279.
[34] Jonas Krautter, Dennis RE Gnad, and Mehdi B Tahoori. 2019. Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12, 3 (2019), 1–26.
[35] George Provelengios, Daniel Holcomb, and Russell Tessier. 2020. Power Wasting Circuits for Cloud FPGA Attacks. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 231–235. DOI: http://dx.doi.org/10.1109/FPL50879.2020.00046
[36] Md Mahbub Alam, Shahin Tajik, Fatemeh Ganji, Mark Tehranipoor, and Domenic Forte. 2019. RAM-Jam: Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 48–55. DOI: http://dx.doi.org/10.1109/FDTC.2019.00015
[37] VLSI EXPERT 2019. *Reading SDF files*. VLSI EXPERT.
[38] Lilas Alrahis, Hassan Nassar, Jonas Krautter, Dennis Gnad, Lars Bauer, Jörg Henkel, and Mehdi Tahoori. 2024. MaliGNNoma: GNN-Based Malicious Circuit Classifier for Secure Cloud FPGAs. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, Tysons Corner, VA, USA, 383–393. DOI: http://dx.doi.org/10.1109/HOST55342.2024.10545411
[39] Jayeeta Chaudhuri and Krishnendu Chakrabarty. 2022. Detection of Malicious FPGA Bitstreams using CNN-Based Learning. In *2022 IEEE European Test Symposium (ETS)*. IEEE, Barcelona, Spain, 1–2. DOI: http://dx.doi.org/10.1109/ETS54262.2022.9810438
[40] Rana Elnaggar, Jayeeta Chaudhuri, Ramesh Karri, and Krishnendu Chakrabarty. 2023. Learning Malicious Circuits in FPGA Bitstreams. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 3 (March 2023), 726–739. DOI: http://dx.doi.org/10.1109/TCAD.2022.3190771
[41] Khoa Dang Pham, Edson Horta, and Dirk Koch. 2017. BITMAN: A tool and API for FPGA bitstream manipulations. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 894–897. DOI: http://dx.doi.org/10.23919/DATE.2017.7927114
[42] T. Sugawara, K. Sakiyama, S. Nashimoto, D. Suzuki, and T. Nagatsuka. 2019. Oscillator without a combinatorial loop and its threat to FPGA in data centre. *Electronics Letters* 55, 11 (2019), 640–642. DOI: http://dx.doi.org/10.1049/el.2019.0163 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/el.2019.0163.
[43] Kaspar Matas, Tuan Minh La, Khoa Dang Pham, and Dirk Koch. 2020. Power-hammering through Glitch Amplification – Attacks and Mitigation. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 65–69. DOI: http://dx.doi.org/10.1109/FCCM48280.2020.00018
[44] Peter Jamieson, Tobias Becker, Peter Y. K. Cheung, Wayne Luk, Tero Rissa, and Teemu Pitkänen. 2010. Benchmarking and evaluating reconfigurable architectures targeting the mobile domain. *ACM Trans. Des. Autom. Electron. Syst.* 15, 2, Article 14 (March 2010), 24 pages. DOI: http://dx.doi.org/10.1145/1698759.1698764
[45] F. Brglez, D. Bryan, and K. Kozminski. 1989. Combinational profiles of sequential benchmark circuits. In *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1929–1934 vol.3. DOI: http://dx.doi.org/10.1109/ISCAS.1989.100747
[46] J. T. Pistorius, Mike Hutton, Alan Mishchenko, and Robert K. Brayton. 2007. Benchmarking Method and Designs Targeting Logic Synthesis for FPGAs. In *IWLS*. https://api.semanticscholar.org/CorpusID:14413517
[47] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovic. 2014. The RISC-V instruction set manual, volume I: User-level ISA, version 2.0. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54* (2014), 4.
[48] Imbalanced learn. *Synthetic Minority Oversampling Technique*. Imbalanced learn. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
[49] Apache. *parquet*. Apache.
[50] INRIA. GridsearchCV. https://scikit-learn.org/stable/modules/grid_search.html. ([n. d.]).
[51] Tuan Minh La, Kaspar Matas, Nikola Grunchevski, Khoa Dang Pham, and Dirk Koch. 2020. FPGADefender: Malicious Self-oscillator Scanning for Xilinx UltraScale+ FPGAs. *ACM TRETS* (2020).

Received 7 August 2025; revised 7 August 2025; accepted 11 August 2025