






# Machine Learning-Based Detection of Glitch Attacks in Clock Signal Data

Asier Gamba<sup>1,2</sup> , Durba Chatterjee<sup>1</sup> , Unai Rioja<sup>2</sup> , Igor Armendariz<sup>2</sup> , and Lejla Batina<sup>1</sup> 

<sup>1</sup> Digital Security Group, Radboud University Nijmegen, The Netherlands

<sup>2</sup> Ikerlan Technology Research Centre, Spain

**Abstract.** Voltage fault injection attacks are a particularly powerful threat to secure embedded devices because they exploit brief, hard-to-detect power fluctuations causing errors or bypassing security mechanisms. To counter these attacks, various detectors are employed, but as defenses strengthen, increasingly elusive glitches continue to emerge. Artificial intelligence, with its inherent ability to learn and adapt to complex patterns, presents a promising solution. This research presents an AI-driven voltage fault injection detector that analyzes clock signals directly. We provide a detailed fault characterization of the STM32F410 microcontroller, emphasizing the impact of faults on the clock signal. Our findings reveal how power supply glitches directly impact the clock, correlating closely with the amount of power injected. This led to developing a lightweight Multi-Layer Perceptron model that analyzes clock traces to distinguish between safe executions, glitches that keep the device running but may introduce faults, and glitches that cause the target to reset. While previous fault injection AI applications have primarily focused on parameter optimization and simulation assistance, in this work we use the adaptability of machine learning to create a fault detection model that is specifically adjusted to the hardware that implements it. The developed glitch detector has a high accuracy showing this a promising direction to combat FI attacks on a variety of platform.

**Keywords:** Fault injection · Machine learning · Fault tolerance.

## 1 Introduction

Embedded devices are designed to operate within specific conditions to meet functional and performance requirements. When exposed to conditions outside their normal range, faults can occur. In 1997, Boneh, DeMillo, and Lipton first suggested taking advantage of transient faults to break secure cryptographic algorithm implementations [8]. They show that with one faulty RSA signature one could retrieve the private key and this created a precedent that following years of publications would build on, making fault injection a field of research in their own right. The attack is known as Bellcore attack. Soon after, Biham

and Shamir [7] discovered that the same principle could be applied to extract the secret key of a DES algorithm by comparing the correct encryption results with faulty ones. They named this attack Differential Fault Analysis (DFA), as it used information from comparing differences between numerous correct and faulty cyphertexts.

A fault model is a mathematical description of how a fault could compromise a system's (correct) operation by exploiting vulnerabilities in its implementation. These vulnerabilities arise from the specific design and construction of the hardware components. Any fault injection technique that aligns with the characteristics of a fault model is theoretically capable of successfully exploiting its vulnerabilities. In other words, if a fault injection technique matches the conditions defined by a fault model, it can induce a fault that compromises the system's security. Various real-world conditions can cause chips to function outside their intended parameters, thus leading to faulty behavior. Those conditions can be also provoked by active attackers capable of introducing some sort of glitch in a system. Actually, fault injection attacks replicate one or more of these conditions in a controlled environment to compromise the security of a system and create or exploit a vulnerability, ultimately with the goal of correcting potential fault tolerance deficiencies in the system.

Fault injection attacks can be applied to all cryptographic algorithms. For instance, Differential Fault Analysis (DFA) can be applied to symmetric ciphers like AES [14] by injecting precise bit flips that enable secret key extraction. The required bit flip can be achieved glitching a microcontroller's clock signal [1], but also injecting glitch attacks on the implementation's power source [21]; or even without physical contact using electromagnetic emanation. Dehbaoui et al. [13] performed Fault Injection by focusing an infrared laser beam at a wavelength of 1,064 nm on the decapsulated chip. Colombier et al. [12], applied a voltage pulse on the backside substrate of an integrated circuit using a conductive needle (Body Biasing Injection) and Chancel et al. [11]; demonstrated that one could tamper with the Dynamic Voltage and Frequency Scaling energy management mechanisms embedded within a Trusted Execution Environment [34]. Given that the fault model for many AES implementations only requires a reproducible single bit-flip, it is no surprise that DFA has emerged as a popular attack method. This vulnerability allows attackers to exploit predictable changes in the output caused by specific fault injections, making DFA an effective technique for compromising AES encryption. In addition to bit flips, other popular transient faults are instruction skips [30], memory access disruption [32], or execution failures. This makes them challenging to diagnose, reproduce, and mitigate, especially considering their temporary nature and lack of lasting damage. Moreover, it abstracts the algorithm being executed from its implementation, leaving different approaches to maintain the code execution's integrity against faulty attacks. Several countermeasures have been developed to mitigate these risks, primarily focusing on intrusion detection, algorithmic resistance, and error detection and correction techniques [4].

Algorithmic resistance aims to tolerate certain types of faults by strengthening the algorithms with additional constants, double checks, loop integrity checks, or flow monitoring. Nevertheless, while the source code may be secure, the compiled binary can still be vulnerable due to compiler optimizations. Hence, protecting security implementations against active adversary has to be done on all implementation levels, from algorithms to circuits.

In this article, we focus on voltage glitching intrusion detection, which involves continuously monitoring the system for anomalies that might indicate a fault attack, such as unexpected changes in execution time, power consumption, error rates, or, in this case, utilizing machine learning model to detect anomalies on the system clock signal. Dedicated hardware AI accelerators, model compression techniques, cloud-based solutions, and optimized software frameworks are enabling the growing feasibility of machine learning on embedded devices. Zhang et al. [37] recently proposed a real-time facial expression recognition system using an AI-powered microcontroller assessing four deep learning algorithms. Real-world voltage fault injection attacks often require iterative adjustments of glitch parameters to discover the precise attack values that trigger the desired fault. A system capable of detecting such behavior could implement effective intrusion detection countermeasures, dynamically updating its security model without relying on traditional analog detection methods.

This article introduces an artificial neural network-based approach for detecting potential glitch attacks. By analyzing the system’s clock signal, the presented model is capable of distinguishing between normal operation, glitches that allow continued operation, and those that cause resets. The model is specifically trained for the hardware properties of the device under test (DUT), a Cortex-M4 STM32 Nucleo board. The characterization of the physical properties of the board is algorithm agnostic, as it is based on the dependency between the power supply and the clock signal.

This article offers two key contributions:

- The characterization of the glitches on the targeted hardware. We exhaustively study the impact of voltage glitches on a 32-bit microcontroller’s logic and clock signal. We create a detailed understanding of glitch effects by systematically testing various glitch parameters and monitoring both device outputs and clock signals.
- A neural network-based glitch detection model. As a proof-of-concept, we develop a lightweight neural network that can accurately identify the influence of glitches on the device’s logic using only clock trace data. This lays a solid foundation for future research on voltage fault injection detection and mitigation.

The rest of the paper is organized as follows. In Sect. 2 we give a brief background on fault injection attacks and countermeasures and machine learning. In Sect. 3 we mention previous machine learning applications in the field. In Sect. 4 we detail the experimental setup for voltage fault injection, alongside the measurements made. In Sect. 5 and Sect. 5 we introduce the machine learning

model developed for glitch prediction and we give a comprehensive analysis of its performance. Sect. 7 concludes the paper.

## 2 Background

### 2.1 Voltage Fault Injection

Voltage Fault Injection (VFI) is a specific FI technique used to disrupt the normal operation of integrated circuits by introducing brief disturbances into their power supply and revealing weaknesses in a device’s ability to handle abnormal conditions. The disturbances are known as voltage glitches or simply glitches. When a glitch is injected at a critical moment—brief enough to avoid causing a shutdown but strong enough to induce a malfunction—it can create a fault that exploits weaknesses in a target execution. The waveform of a glitch is typically characterized by its voltage, duration, and timing, as illustrated in Fig. 1.

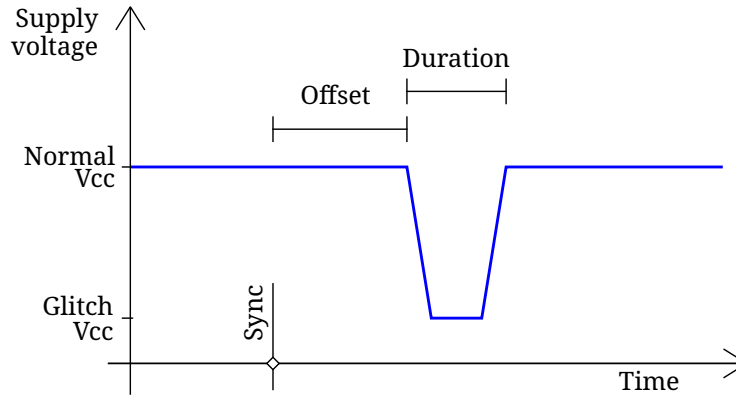


Fig. 1. The parameters of a voltage glitch.

A VFI setup involves systematically introducing glitches into a system and observing its responses to categorize the outcomes. The target can exhibit three main behaviors in response to glitches: NORMAL, where the system functions correctly and ignores the glitch; RESET, where the system ceases to respond entirely; and SUCCESS, where the system behaves differently than expected, indicating that the fault injection was effective. A fault parameter search aimed at identifying faulty behavior typically involves a systematic approach that iterates between normal operation and induced reset glitches. The process begins by executing the system under normal conditions to establish a baseline performance profile. During the search, the system alternates between normal operation and the introduction of these glitches. Each iteration involves monitoring the system’s response to the faults and assessing how they impact outputs, error rates, or overall functionality. By varying the characteristics of the glitches, the search

can pinpoint specific conditions that trigger faulty behavior. With knowledge of the fault parameters, developers can implement mitigation strategies or modify the system architecture to enhance resilience against identified faults.

Over the last two decades, many different fault injection mechanisms followed the path opened by Boneh et al. and established techniques to attack implementations of cryptographic algorithms. Aumüller et al. [3] published the first practical demonstration of the Bellcore attack on RSA using electrical glitches in smartcards with hardware protections disabled. O’Flynn [24] enhanced voltage fault injection by providing high temporal resolution by introducing a crowbar circuit, allowing for precise control over glitch timing. Bozzato et al. [9] proposed an alternative to the crowbar circuit by using a Digital-to-Analog Converter to precisely optimize glitch pulse shapes with genetic algorithms, enhancing the accuracy and effectiveness of fault injection attacks. Recently, Saß et al. [30] extended O’Flynn’s circuit to bypass the ARM’s TrustZone-M security protections with multiple coordinated voltage faults.

## 2.2 Machine Learning

Machine learning (ML) is a subfield of artificial intelligence that empowers computer systems to learn from data and make probabilistic decisions or predictions without explicit programming [5]. This involves the construction of algorithms that can learn from and make predictions or decisions based on data. The most commonly used ML types include supervised learning, where models are trained on labeled datasets to predict outcomes; unsupervised learning, which identifies patterns in unlabeled data; and reinforcement learning, which optimizes decision-making processes through trial and error.

In the context of fault injection, machine learning establishes complex relationships within various fault conditions and learns to approach these insights autonomously. By analyzing data from past fault occurrences and their propagation, ML algorithms have the potential to detect ongoing faulty executions [16] or identify optimal fault parameters that expose vulnerabilities [26].

## 3 Related Work

### 3.1 ML-based Fault Injection Attacks

Fault injection evaluations aim to uncover vulnerabilities in systems by simulating adversarial attacks, which involves navigating a complex parameter space with factors like fault type, location, timing, and duration. Manual or brute-force searching through this space can be computationally expensive and time-consuming. Machine learning algorithms that optimize complex search spaces are well-suited for this task.

Picek et al. first introduced ML-based parameter search optimization on hardware FIA with a genetic algorithm approach [26]. Genetic algorithms are a form of evolutionary optimization technique that perfectly adapts to search

for optimal solutions in complex, well-defined problem spaces such as glitch parameters. In the literature, optimization algorithms have been instrumental in multiple fault injection techniques, including voltage glitching [10, 25, 35], electromagnetic fault injection (EMFI) [22, 23], and laser fault injection [19, 20], showcasing its versatility across different techniques. On the software side, machine learning (ML) has significantly advanced fault analysis, including assisting reverse engineering embedded systems [18], getting to automate bit-flip fault injections in white-box AES implementations [36] or identifying non-relevant information on virtual platforms’ fault propagation, allowing precise countermeasures [28].

### 3.2 ML-based Fault Injection Countermeasures

On the other end of adversarial models, ML applications are also employed to develop countermeasures and fault detection systems. Methods for addressing fault injection attacks fall into two main categories: attack detection, which involves monitoring both hardware and software to spot ongoing attacks, and software vulnerability identification, which focuses on finding potential weaknesses that attackers could exploit.

Unlike adversarial models, these techniques often use supervised learning, where the algorithm is trained on a labeled dataset. In supervised learning, the model learns to classify and distinguish between secure data outcomes and potential security risks based on this training data. As a result, a model can effectively identify and respond to anomalies or threats in real-time, providing immediate responses to potential security breaches. Software approaches to fault behavior detection aim to differentiate between normal and anomalous behavior. Gangolli et al. [16] use real-time monitoring in IoT systems to identify and classify software faults. Meanwhile, Saha et al. [29] analyze ciphertext distributions to detect fault-induced leakage, identifying anomalies produced by fault injection attacks.

Hardware approaches to fault detection often focus on identifying rapid glitches that conventional methods might miss by examining their effects on glitch-sensitive circuits. For example, specialized hardware glitch detectors are designed to catch fast glitches that standard detectors might overlook [17]. However, these detectors require fine-tuning and are typically effective only for specific types of glitches. Facon et al. [15] introduced an AI framework that adjusts each sensor’s threshold individually, leading to notable improvements in EMFI detection. Similarly, Shrivastwa et al. [31] combined digital sensors with artificial intelligence to continuously monitor sensor data and use AI for real-time fault classification and analysis. Monitoring the system clock through a detector circuit that identifies timing violations can also be effective for detecting glitches, which are the basis of voltage and glitch attacks. However, high-speed clock glitching attacks can still be executed successfully, even in systems protected by such detectors [2].

In contrast, the novel approach presented in this work relies exclusively on AI to detect anomalies caused by fault glitches. To the best of our knowledge, no

prior research has directly used machine learning models to monitor an embedded system’s signals for identifying potentially insecure behaviors resulting from fault injection.

## 4 Voltage Fault Injection Testbed

This section presents an experimental fault injection evaluation of a Cortex-M4 STM32F410RB board [33] running at its default 16 MHz. The setup provides insights into microcontroller vulnerability and adversarial strategies for improving fault tolerance and security in embedded systems. To this end, experiments attempt to modify the running program on the DUT by injecting glitches in its power supply mid-operation. Fig. 2 illustrates the setup, which can be considered a standard onboard fault injection system.

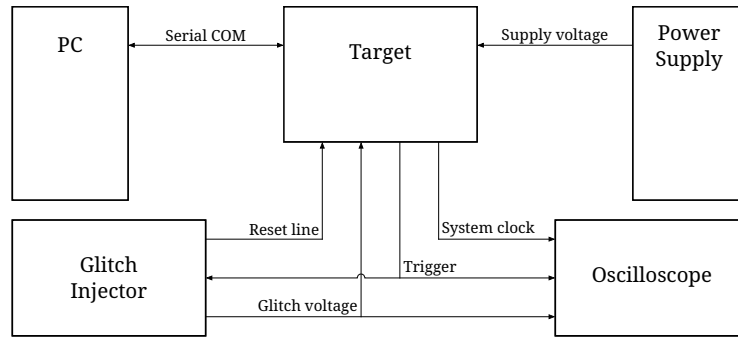


Fig. 2. Voltage fault injection setup.

**Setup configuration** The setup was built on the Riscure fault injection FPGA-Based Inspector platform [27]. It counts with precise timing control signals, high clock frequencies with a dual-core running at 125 MHz, and analog outputs with 4 ns pulse resolution. The complex tasks and algorithms, the attack logic, and the specific communication protocol used by the target are implemented in a custom Java framework that orchestrates fault injection evaluation. Algorithm 1 presents a glitch measurement loop in which glitches are injected into the target through an internal voltage regulator pin. This pin usually regulates voltage for all digital circuits within the chip, except for the backup domain and standby circuitry, and is designed to support an external capacitor for stable operation. However, it also provides an access point for localizing voltage glitches on the chip’s core logic, making it ideal for voltage fault injection. The board is powered by an external supply and connects to the PC via a serial interface and to the glitch injector via an output GPIO (General Purpose Input/Output). The GPIO provides a timing reference for glitch injection to the PC, signaling when

---

**Algorithm 1:** Fault injection sequence

---

```

init fault injection sequence
enable inputPin triggerPin
enable serialCOM uartBoard
enable oscilloscope
load n sets of glitch parameters
for each set of glitch parameters
  reset board
  when triggerPin is high
    sleep offset ns
    supplyVcc  $\leftarrow$  glitch_voltage
    sleep glitch_duration ns
    supplyVcc  $\leftarrow$  normal_supply_voltage
  if uartBoard.receive(r32) then
    if r32 = ones then
      verdict  $\leftarrow$  normal
    else
      verdict  $\leftarrow$  success
  else
    verdict  $\leftarrow$  reset
  oscilloscope.receive(clk_trace)

```

---

targeted instructions are imminent. Serial communication is used to transmit feedback messages, and a fault is identified if the feedback message deviates from the expected response. Additionally, the board outputs its system clock through another GPIO connected to an oscilloscope. The oscilloscope records the clock signal trace for each glitch injection, using the GPIO as a trigger to synchronize with the measurements.

The DUT is modified by removing external capacitors from power supply pins, the reset pin, and the internal voltage regulator— a standard practice to increase susceptibility to voltage fluctuations. Algorithm 2 presents the main on-board code executed. The program clears a 32-bit register, raises a trigger signal, sets all bits to one using four assembly instructions, and sends the register to the PC. If the PC receives a message where all 32 bits are set to one, it confirms that the code has executed as expected without any glitches or errors. However, if the message contains zeros, it means there was a fault during execution. If the PC receives no message at all, it means the injected glitch caused a system restart.

**Fault parameter space** Literature shows that many parameters affect the performance of glitches on their fault-inducing capability, such as timing, power, length, absolute glitch voltage, and even waveform [9].

In our experiments, a glitch is characterized by its offset, voltage, and duration. The *offset* measures the interval between the trigger signal sent by the DUT and the glitch injection and is set to range from 0 to 600 *ns* in 16-*ns* steps. The *glitch voltage* refers to the absolute voltage level reached during a glitch. On



---

**Algorithm 2:** On-board code

---

```

init board
enable outputPin triggerPin
enable serialCOM uartPC
r32  $\leftarrow$  0
triggerPin  $\leftarrow$  high
repeat 4 times for different bits
    r32  $\leftarrow$  flip 8 bits
uartPC.transmit(r32)

```

---

the target voltage supply line, the glitch voltage ranged from 1.1 V (the nominal operating voltage) to 0 V (ground), in 0.1-V steps. Finally, the glitch *duration* determines the duration of the voltage deviation from its nominal value, set between 4 and 88 *ns*, in 16-*ns* steps.

**Injected faults** Here we examine the logical outcomes resulting from different glitch injections, while clock traces are addressed in the next section. We analyze normal executions, faults, and resets in relation to the glitch parameters that caused them. In order to comprehensively visualize the results, a distinct color was assigned to the three possible verdicts: green represents NORMAL behavior, where the program works as expected; yellow shows the glitch parameters that produced a RESET on the target; and red represents the SUCCESS glitch parameters that modified the outcome of the onboard program. These colors represent the logical feedback received from the DUT for each combination of parameters within the specified ranges. Fig. 3 visualizes the verdicts as points on a three-dimensional plot, where the axes represent glitch voltage (*z*), duration (*x*), and timing (*y*) across the specified value ranges.

As shown by the accumulation of yellow dots, combinations of low glitch voltage values and extended glitch durations are likely to trigger a reset in the DUT. Conversely, the code functions as intended when the glitch voltage is close to the nominal supply value of 1.1 V. The most interesting behavior, represented by red dots, occurs in the intermediate range between these two states, where the stress nearly causes failures but ultimately does not. While the parameter combinations are reproducible, the occurrence of faults is probabilistic, meaning they do not always propagate to the system’s logical output. Nonetheless, a SUCCESS verdict signifies that the recorded clock trace is directly associated with faulty behavior and helps characterize how the hardware responds in the presence of faults. This behavior is precisely what a fault detector aims to identify, as it represents the signature of failures in maintaining code execution integrity.

A total of 10,488 measurements were conducted, yielding the following results: 92.63% were NORMAL operation, 6.00% resulted in a RESET, and 1.35% produced a SUCCESS output, which occurred within 24-88 *ns* glitch durations and below 0.8V glitch voltages. The parameter ranges that characterize each outcome are presented in Table 1.

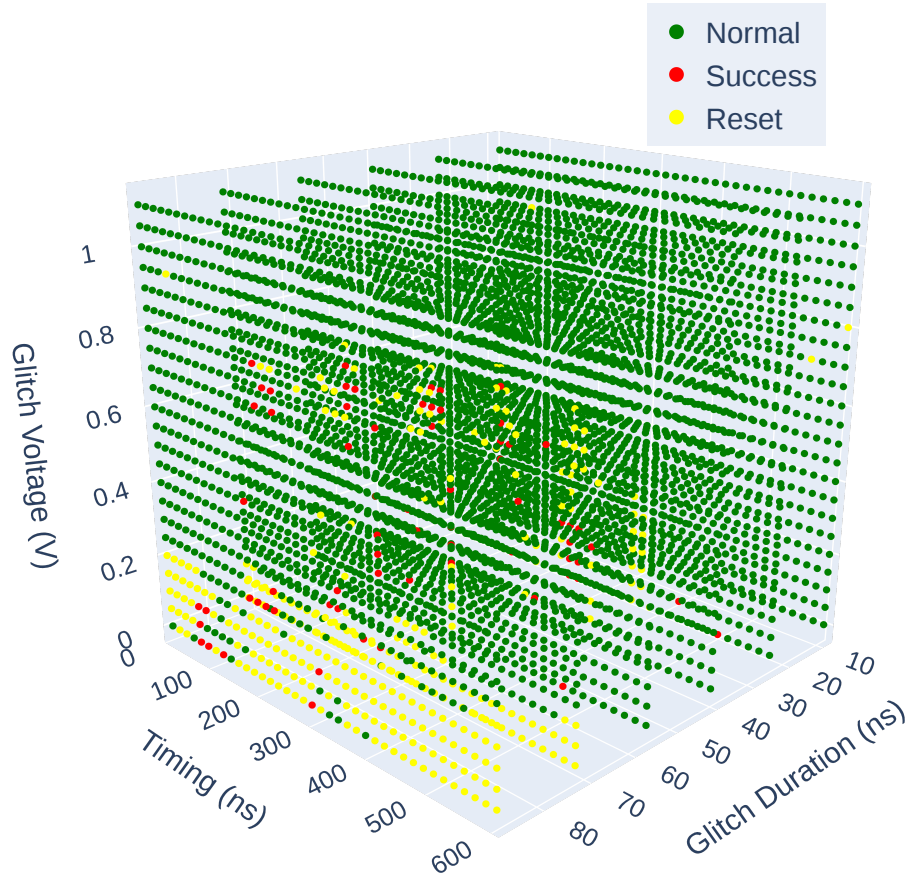


Fig. 3. FI verdicts (color-coded) sorted by glitch parameters (axes).

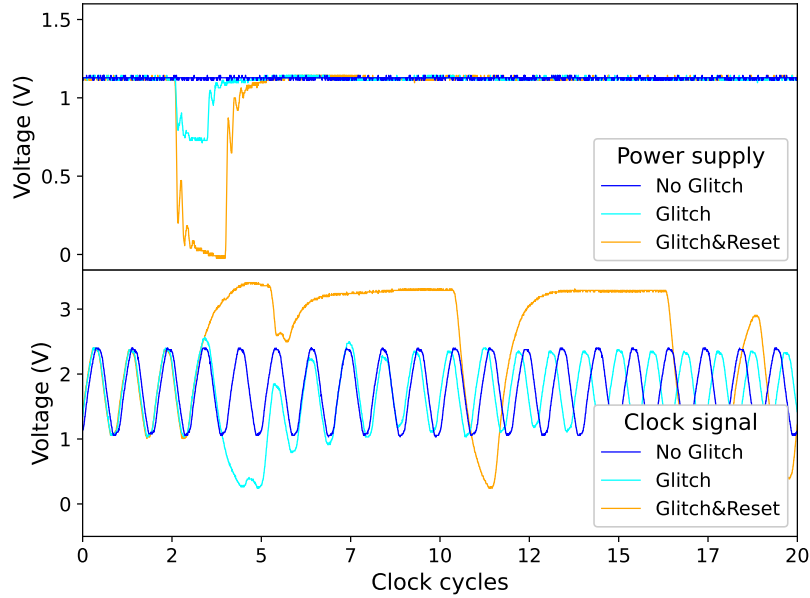
## 5 Glitch detection model

Next, we take the obtained traces and build a glitch detector model able to predict if a potential fault-injecting glitch is being injected on the DUT based on their clock signal. The primary goal of the experiment was to observe the impact of these faults on the clock signal, and it is evident that different glitch power values significantly affect it. Fig. 4 illustrates recorded clock and glitch voltage traces of three cases: regular operation (no glitch), a glitched operation that successfully produced a fault, and a reset-inducing glitched operation.

The fundamental objective of a glitch detector is to identify potential glitch attacks. The neural network is trained with three distinct labels based on glitch characteristics. Traces with no glitches, where the glitch voltage is at the nominal 1.1 V, are considered safe and labeled as *no glitch* (blue). Traces with glitches

**Table 1.** Experimental outcome parameter limits

Parameter range	Success	Normal	Reset
Absolute glitch Vcc max	0.8V	1.1V	0.85V
Absolute glitch Vcc min	0.0V	0.0V	0.0V
Glitch Delay max	520.0ns	600.0ns	600.0ns
Glitch Delay min	40.0ns	8.0ns	8.0ns
Glitch Duration max	88.0 ns	88.0ns	88.0ns
Glitch Duration min	24.0 ns	8.0ns	24.0ns
Glitch power max	96.8pW	96.8pW	96.8pW
Glitch power min	19.6pW	0.0pW	18.0pW

**Fig. 4.** Power supply glitches and their resulting clock signal traces.

that lead to a system restart, characterized by high glitch power (low voltage and long duration), are labeled as *reset* (orange). The remaining traces, which contain glitches of varying intensity but do not trigger a reset, are labeled as *glitch* (cyan), indicating potential fault injection risks.

We adapted the ASCAD Multi-Layer Perceptron (MLP) model for our experiment to meet the specific needs of clock signal fault detection [6]. The MLP’s architecture, originally designed for side-channel analysis, was used to capture anomalies in clock traces related to power supply glitches, leveraging its strength in identifying subtle patterns in temporal data. MLPs are a type of artificial neu-

ral network composed of multiple layers of interconnected nodes. The parameters used for the MLP in this experiment are presented in Table 2.

**Table 2.** MLP model parameters

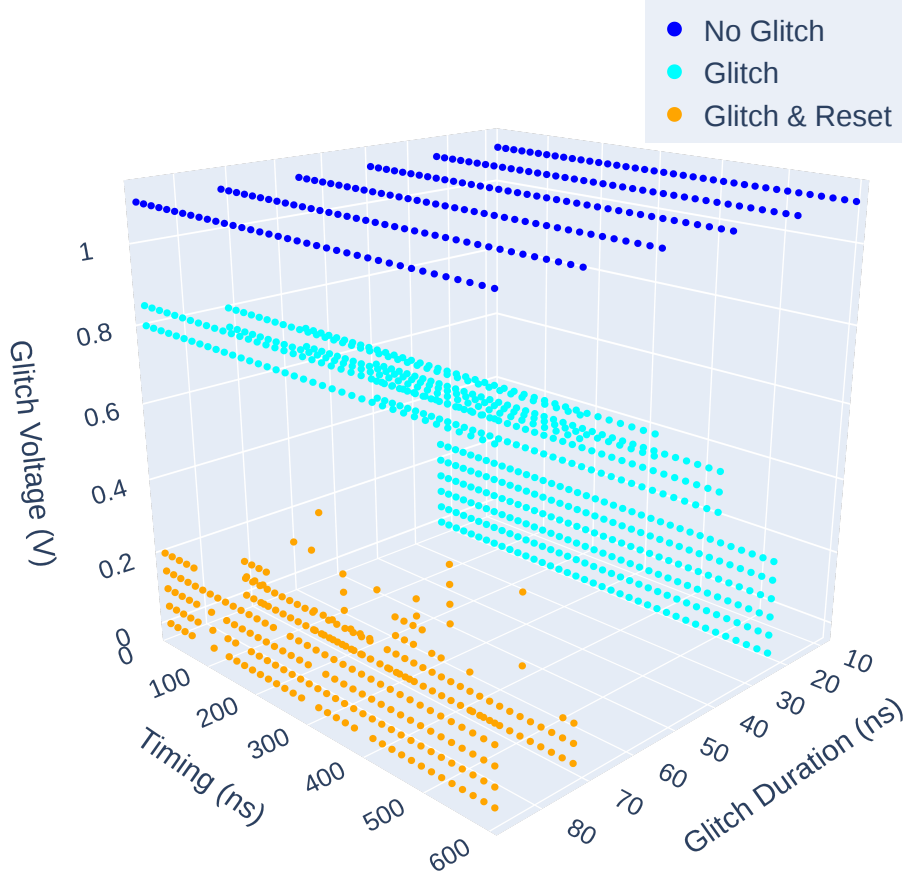
Parameter	Value
Number of layers	6
Number of nodes (Hidden layers)	200
Activation function (Hidden layers)	ReLU
Input dimension	500
Output classes	3
Activation function (Output layer)	Softmax
Optimizer	RMSprop
Learning rate	0.00005
Loss function	Categorical cross-entropy
Metrics	Accuracy

Due to its focus on glitch characterization, the dataset is imbalanced as only 4.3% of traces are glitch-free, and 6.0% show reset behavior. This imbalance can cause the model to favor the majority class, reducing its ability to accurately detect less common cases. To address this issue, we exclude glitched traces with power levels outside the 20 to 27 pW range and reset-inducing glitches with power below 50 pW. This adjustment reduces the proportion of glitched traces to 55.12% while increasing the proportions of reset and glitch-free traces to 22.82% and 22.05%, respectively. Fig. 5 shows the glitch parameters of the selected traces used to train the neural network.

**Trace preprocessing** The model is trained using clock traces as inputs and the corresponding labels as metadata. The clock traces were decimated at a rate of 100 with a 0.5 overlap moving average filter to simulate a lower sample frequency. Measured traces have 50 000 samples recorded at a 6.25 GHz sample freq, while input pre-processed traces have 500 samples at 62.5 MHz (approximately 4 times the DUT clock frequency). The trained model will then learn to recognize the patterns glitch attacks produce on the clock signal in order to detect fault attack attempts.

## 6 Performance evaluation

Results shown in Table 3 are averaged from 10 training runs with different random seeds. This approach reduces variability from data shuffling and model initialization, providing a more reliable measure of performance. The model performs exceptionally well, with an accuracy of 99.20% and low standard error, indicating high consistency. Additionally, the recall and F1 scores are reported for each class (*no glitch*, *glitch*, and *reset*-producing glitches) to further comprehend the model’s performance across different scenarios. Recall, or the true



**Fig. 5.** Dataset class labels (color-coded) sorted by glitch parameters (axes).

**Table 3.** Model performance metrics

Metric	Value	Standard Error
Accuracy	99.20%	0.001907
Recall – <i>No glitch</i>	100.00%	0
Recall – <i>Glitch</i>	99.91%	0.000583
Recall – <i>Reset</i>	96.82%	0.007705
F1 Score – <i>No glitch</i>	99.87%	0.000828
F1 Score – <i>Glitch</i>	99.31%	0.001763
F1 Score – <i>Reset</i>	98.26%	0.004201

positive rate, measures the proportion of actual positive instances correctly identified by the model. The F1 score combines recall with precision, which is the proportion of true positive predictions out of all positive predictions made. By balancing the two metrics, the F1 score provides a comprehensive assessment of the model’s performance, particularly in imbalanced datasets, where it accounts for both false positives and false negatives. The model perfectly identifies cases with no glitches, meaning no false negative prediction of glitches is identified as *no glitch*. The very low standard error *glitch* recall indicates that the model is highly sensitive to detecting glitches, making it well-suited for identifying minor fault conditions. Although slightly lower, it performs well in detecting true *reset*-producing glitches. The higher standard error suggests more variability in this category, meaning *reset* behavior is harder to identify than the others. In summary, the model shows excellent performance, though it may benefit from improvements in detecting the less frequent reset-producing glitches.

## 7 Conclusions

The glitch detector has demonstrated high accuracy and effectively identifies various effects glitches can have on the target, showcasing its potential as a robust fault detection tool. Improvements could involve implementing real-time detection for immediate response, boosting its practical value. Future enhancements might include extending the detector to other fault injection methods.

## References

1. Agoyan, M., Dutertre, J.M., Naccache, D., Robisson, B., Tria, A.: When clocks fail: On critical paths and clock faults. In: Gollmann, D., Lanet, J.L., Iguchi-Cartigny, J. (eds.) Smart Card Research and Advanced Application. LNCS, vol. 6035, pp. 182–193. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
2. Askeland, A., Nikova, S., Nikov, V.: Who watches the watchers: Attacking glitch detection circuits. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(1), 157–179 (Dec 2023). <https://doi.org/10.46586/tches.v2024.i1.157-179>
3. Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., Seifert, J.P.: Fault attacks on rsa with crt: Concrete results and practical countermeasures. In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 260–275. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). [https://doi.org/10.1007/3-540-36400-5\\_20](https://doi.org/10.1007/3-540-36400-5_20)
4. Barenghi, A., Breveglieri, L., Koren, I., Pelosi, G., Regazzoni, F.: Countermeasures against fault attacks on software implemented aes: effectiveness and cost. In: Proceedings of the 5th Workshop on Embedded Systems Security. p. 7. WESS '10, Association for Computing Machinery, New York, NY, USA (10 2010). <https://doi.org/10.1145/1873548.1873555>, <https://doi.org/10.1145/1873548.1873555>
5. Bell, J.: Machine Learning: Hands-on for Developers and Technical Professionals. Wiley (2014), <https://books.google.es/books?id=YP-7nQAACAAJ>

6. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ascad database. *Journal of Cryptographic Engineering* **10**(2), 163–188 (Jun 2020). <https://doi.org/10.1007/s13389-019-00220-8>, <https://doi.org/10.1007/s13389-019-00220-8>
7. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski, B.S. (ed.) *Advances in Cryptology — CRYPTO '97*. pp. 513–525. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)
8. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) *Advances in Cryptology — EUROCRYPT '97*. pp. 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)
9. Bozzato, C., Focardi, R., Palmarini, F.: Shaping the glitch: Optimizing voltage fault injection attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2), 199–224 (Feb 2019). <https://doi.org/10.13154/tches.v2019.i2.199-224>, <https://tches.iacr.org/index.php/TCHES/article/view/7390>
10. Carpi, R.B., Picek, S., Batina, L., Menarini, F., Jakobovic, D., Golub, M.: Glitch it if you can: Parameter search strategies for successful fault injection. In: Francillon, A., Rohatgi, P. (eds.) *Smart Card Research and Advanced Applications*. vol. 8419, pp. 236–252. Springer International Publishing, Cham (Jun 2014). [https://doi.org/10.1007/978-3-319-08302-5\\_16](https://doi.org/10.1007/978-3-319-08302-5_16)
11. Chancel, G., Galliere, J.M., Maurine, P.: A better practice for body biasing injection. In: *2023 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. pp. 48–59. IEEE (Sep 2023). <https://doi.org/10.1109/FDTC60478.2023.00014>
12. Colombier, B., Menu, A., Dutertre, J.M., Moëllic, P.A., Rigaud, J.B., Danger, J.L.: Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. In: *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. pp. 1–10. IEEE (2019). <https://doi.org/10.1109/HST.2019.8741030>
13. Dehbaoui, A., Dutertre, J.M., Robisson, B., Tria, A.: Electromagnetic transient faults injection on a hardware and a software implementations of aes. In: *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*. pp. 7–15. IEEE (Sep 2012). <https://doi.org/10.1109/FDTC.2012.15>
14. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on a.e.s. In: *International Conference on Applied Cryptography and Network Security* (2003), <https://api.semanticscholar.org/CorpusID:1089>
15. Facon, A., Guilley, S., Ngo, X.T., Nguyen, R., Perianin, T., Shrivastwa, R.R.: High precision emfi detector using machine learning and sensor fusion. In: *Cesar-Conference 2019* (2019)
16. Gangolli, A.A., Mahmoud, Q., Azim, A.: A machine learning based approach to detect fault injection attacks in iot software systems. In: *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. pp. 2900–2905. IEEE (Oct 2022). <https://doi.org/10.1109/SMC53654.2022.9945117>
17. Goikoetxea Yanci, A., Pickles, S., Arslan, T.: Detecting voltage glitch attacks on secure devices. In: *2008 Bio-inspired, Learning and Intelligent Systems for Security*. IEEE (Aug 2008). <https://doi.org/10.1109/bliss.2008.26>
18. Khosrowjerdi, H., Meinke, K., Rasmusson, A.: Virtualized-fault injection testing: A machine learning approach. In: *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE (Apr 2018). <https://doi.org/10.1109/ICST.2018.00037>

19. Krcek, M., Fronte, D., Picek, S.: On the importance of initial solutions selection in fault injection. In: 2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). pp. 1–12. IEEE (Sep 2021). <https://doi.org/10.1109/FDTC53659.2021.00011>
20. Krcek, M., Ordas, T., Fronte, D., Picek, S.: The more you know: Improving laser fault injection with prior knowledge. In: 2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). pp. 18–29 (Sep 2022). <https://doi.org/10.1109/FDTC57191.2022.00012>
21. Lu, Y.: Attacking hardware aes with dfa. ArXiv [abs/1902.08693](https://arxiv.org/abs/1902.08693) (2019), <https://api.semanticscholar.org/CorpusID:67856731>
22. Maldini, A., Samwel, N., Picek, S., Batina, L.: Genetic algorithm-based electromagnetic fault injection. In: 2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 35–42 (Sep 2018). <https://doi.org/10.1109/FDTC.2018.00014>
23. Maldini, A., Samwel, N., Picek, S., Batina, L.: Automated Methods in Cryptographic Fault Analysis, chap. Optimizing Electromagnetic Fault Injection with Genetic Algorithms, pp. 281–300. Springer International Publishing, Cham (2019). [https://doi.org/10.1007/978-3-030-11333-9\\_13](https://doi.org/10.1007/978-3-030-11333-9_13)
24. O’Flynn, C.: Fault injection using crowbars on embedded systems. IACR Cryptol. ePrint Arch. **2016**, 810 (2016), <https://api.semanticscholar.org/CorpusID:8502986>
25. Picek, S., Batina, L., Buzing, P., Jakobovic, D.: Fault Injection with a New Flavor: Memetic Algorithms Make a Difference, LNCS, vol. 9064. Springer International Publishing (Apr 2015). <https://doi.org/10.1007/978-3-319-21476-4>
26. Picek, S., Batina, L., Jakobovic, D., Carpi, R.: Evolving genetic algorithms for fault injection attacks. In: 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). pp. 1106–1111. IEEE (May 2014). <https://doi.org/10.1109/MIPRO.2014.6859734>
27. Riscure: Inspector fi, <https://riscure.com/security-tools/inspector-fi/>
28. Rosa, F., Garibotti, R., Ost, L., Reis, R.: Using machine learning techniques to evaluate multicore soft error reliability. Circuits and Systems I: Regular Papers, IEEE Transactions on **PP** (Apr 2019). <https://doi.org/10.1109/TCSI.2019.2906155>
29. Saha, S., Alam, M., Bag, A., Mukhopadhyay, D., Dasgupta, P.: Learn from your faults: Leakage assessment in fault attacks using deep learning. Journal of Cryptology **36** (05 2023). <https://doi.org/10.1007/s00145-023-09462-6>
30. Saš, X.M., Mitev, R., Sadeghi, A.R.: Oops.! i glitched it again! how to Multi-Glitch the Glitching-Protections on ARM TrustZone-M. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 6239–6256. USENIX Association, Anaheim, CA (Aug 2023), <https://www.usenix.org/conference/usenixsecurity23/presentation/sass>
31. Shrivastwa, R.R., Guilley, S., Danger, J.L.: Multi-source fault injection detection using machine learning and sensor fusion. In: Stănică, P., Mesnager, S., Debnath, S., Kumar (eds.) Security and Privacy. vol. 1497, pp. 93–107. Springer (11 2021). [https://doi.org/10.1007/978-3-030-90553-8\\_7](https://doi.org/10.1007/978-3-030-90553-8_7)
32. Skorobogatov, S.: Optical fault masking attacks. In: 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 23–29. IEEE (Aug 2010). <https://doi.org/10.1109/FDTC.2010.18>
33. STMicroelectronics: Stm32 nucleo-64 development board with stm32f410rb mcu, <https://www.st.com/en/evaluation-tools/nucleo-f410rb.html>



34. Tang, A., Sethumadhavan, S., Stolfo, S.: CLKSCREW: Exposing the perils of Security-Oblivious energy management. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 1057–1074. USENIX Association, Vancouver, BC (Aug 2017), <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang>
35. Werner, V., Maingault, L., Potet, M.L.: Fast calibration of fault injection equipment with hyperparameter optimization techniques. In: Grosso, V., Pöppelmann, T. (eds.) Smart Card Research and Advanced Applications. vol. 13173, pp. 121–138. Springer International Publishing (2022). [https://doi.org/10.1007/978-3-030-97348-3\\_7](https://doi.org/10.1007/978-3-030-97348-3_7)
36. Wu, L., Ribera, G., Beringuier-Boher, N., Picek, S.: A fast characterization method for semi-invasive fault injection attacks. In: Topics in Cryptology – CT-RSA 2020. pp. 146–170. Cham (Feb 2020). [https://doi.org/10.1007/978-3-030-40186-3\\_8](https://doi.org/10.1007/978-3-030-40186-3_8)
37. Zhang, J., Xie, X., Peng, G., Liu, L., Yang, H., Guo, R., Cao, J., Yang, J.: A real-time and privacy-preserving facial expression recognition system using an ai-powered microcontroller. *Electronics* **13**(14), 2791 (Jul 2024). <https://doi.org/10.3390/electronics13142791>, <https://www.mdpi.com/2079-9292/13/14/2791>