

Enhancing Power Fuzzing: Synthetic Side-Channel Data Generation, Optimal Sampling, and Noise Mitigation

by

Olha Vakulenko

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2025

© Olha Vakulenko 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Embedded systems increasingly dominate critical applications, driving the need for advanced testing and validation methodologies capable of uncovering hidden or undocumented behaviours. Traditional fuzzing approaches, which rely on observable outputs or system crashes, often fail to reveal the internal operations of embedded devices. Powertrace-based fuzzing provides a non-intrusive alternative by analysing a device’s power consumption during operation. Achieving robust and reliable fuzzing performance requires researchers to overcome significant challenges in signal acquisition, noise mitigation, and classification reliability.

This thesis addresses these challenges by introducing several key improvements to the PowerFuzzer framework. First, it develops SigVarGen, a modular synthetic signal generation framework that produces realistic idle-state and active signals under controlled noise, drift, and timing variations. SigVarGen enables comprehensive algorithm development and stress testing across diverse simulated conditions, bridging the theoretical model design and empirical validation gap. Second, it presents SR&OS, a dynamic calibration algorithm that optimises sampling rate and trigger offset selection. SR&OS leverages adaptive binary search and statistical response detection to capture meaningful system responses despite variable latencies and noise conditions.

The thesis also performs a detailed risk assessment of typical noise sources in side-channel measurements and ranks mitigation strategies based on their effectiveness and practical feasibility. It identifies practical denoising techniques, such as trace averaging, singular spectrum analysis, and independent component analysis, as effective methods for improving signal quality. Furthermore, it evaluates signal quality metrics and validates comparative power and correlation-based indicators as efficient predictors for adaptive acquisition termination.

Together, these developments create a more robust and scalable framework for detecting undocumented behaviours in embedded systems through powertrace analysis. Experimental validation using synthetic datasets and real-world embedded targets demonstrates improvements in calibration accuracy and acquisition efficiency. The findings lay a foundation for future advancements in hardware fuzzing frameworks, mainly targeting embedded environments.

Acknowledgements

I am deeply grateful to my professor, Sebastian Fischmeister, who believed in me, helped me begin a new chapter in my academic journey and guided through it. I am thankful to my labmates, Murray, Cameron, Roberto, Kavish, and, especially, Michael, without whom I wouldn't have been able to neither start nor finish my research. My heartfelt thanks go to my mother, Lily, and my sister, Alyona – my beloved family. I am thankful as well for the unwavering support of my friends Marta, Tonya, Ksusha, Andrii, Halyna, and Sasha, who stood by me through both highs and lows. Finally, I am also thankful to the Natural Sciences and Engineering Research Council of Canada, which supported my research at the University of Waterloo.

Dedications

To the bravery of the Ukrainian army, to whom I owe my life.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedications	v
List of Figures	x
List of Tables	xii
List of Abbreviations	xiii
1 Introduction	1
1.1 Introduction	1
1.2 Thesis Objectives and Contributions	2
2 Powertrace-Based Fuzzing Concept	4
2.1 Fundamentals of Hardware Fuzzing	4
2.2 PowerFuzzer Framework Overview and Methodology	5
2.2.1 Problem Statement	5
2.2.2 Approach and Methodology	7

2.3	PowerFuzzer System Architecture	9
2.4	Challenges in Powertrace-Based Fuzzing	11
3	Time-Series Data Generation Framework	16
3.1	Synthetic Signal Generation Motivation and Background	16
3.2	Related Works	18
3.3	Methodology and Design	24
3.3.1	Signal Generation	25
3.3.2	Noise Generation and Addition	27
3.3.3	Signal Variation and Augmentation	29
3.4	Implementation Details	31
3.4.1	Python Environment and Dependencies	32
3.4.2	Framework Structure	32
3.4.3	Key Functions and Modules	33
3.5	Experimental Setup and Case Studies	39
3.5.1	Baseline Signal Collection	40
3.5.2	Case Study 1: Signal Synthesis for Embedded Platforms	41
3.5.3	Case Study 2: Synthetic Signal Augmentation	52
3.6	Experimental Results	57
3.6.1	Case Study 1: Signal Synthesis for Embedded Platforms	57
3.6.2	Case Study 2: Synthetic Signal Augmentation	59
3.7	Results and Conclusion	61
4	SR&OS: Sample Rate & Offset Search	64
4.1	Introduction	64
4.2	Problem Statement	65
4.3	Methodology	66
4.3.1	Powertrace Acquisition and Preprocessing	67

4.3.2	Dynamic Identification of Response Windows	69
4.3.3	Optimal Parameters Search via Adaptive Binary Search	71
4.3.4	Multi-Offset Evaluation	75
4.3.5	Conclusion	76
4.4	Experimental Results & Evaluation	77
4.4.1	Experimental Setup	77
4.4.2	Performance and Scalability Analysis	79
4.4.3	Comparative Analysis: Adaptive Binary Search vs Grid Search . . .	81
4.4.4	Experimental Results	87
4.5	Conclusion and Key Findings	95
5	Trace Quality Enhancement and Anomaly Detection Methods	99
5.1	Systematisation of Knowledge: The Nature of Noise	100
5.1.1	Thermal and Device Noise	100
5.1.2	Electromagnetic Interference (EMI) and Environmental Noise . . .	101
5.1.3	Power Supply Noise and Ground Bounce	101
5.1.4	Crosstalk and Coupling from Other Signals	102
5.1.5	Clock Jitter and Trace Misalignment	103
5.1.6	Temperature Variations	103
5.1.7	Operating System and Scheduling Noise	104
5.1.8	Algorithmic and Data-Dependent Noise	105
5.1.9	ADC Quantization and Measurement Instrument Noise	106
5.1.10	Summary	107
5.1.11	Risk Assessment	109
5.2	Systematisation of Knowledge: Mitigation Strategies based on Noise Type	110
5.2.1	Noise Mitigation Methodologies	111
5.2.2	Summary	120
5.2.3	Risk Assessment	122

5.3	Systematisation of Knowledge: Signal Quality Metrics	127
5.3.1	Total Power of the Signal	127
5.3.2	Signal-to-Noise Ratio	128
5.3.3	Comparative Power	132
5.3.4	Central Limit Theorem	134
5.3.5	Correlation-Based Trace Prediction	136
5.3.6	Summary	138
5.4	Noise Mitigation and Robust Classification Proposed Strategies	139
5.4.1	Methodology and Approach	140
5.4.2	Unbiased Power Measurement	142
5.4.3	Adaptive Estimation Strategy for Trace Requirements	143
5.4.4	Distribution Preprocessing	144
5.4.5	Downsampled DTW Averaging	144
5.4.6	Singular Spectrum Analysis	149
5.4.7	Shared Noise Removal	151
5.4.8	Trace Length Adjustment	151
5.4.9	Classification	152
6	Conclusion	154
6.1	Challenge-Specific Results	154
6.2	Recommendations for Enhancing PowerFuzzer Performance and Robustness	156
6.3	Limitations and Future Work	158
	References	159
	APPENDICES	171
A	SigVarGen: KDE Plots	172
B	SigVarGen: ED-CP Pair Comparisons	176
C	SR&OS Algorithm Flow Charts	180

List of Figures

2.1	PowerFuzzer Architecture	10
2.2	PowerFuzzer RPI response trace	11
2.3	PowerFuzzer RPI base traces	11
3.1	Real PhiHue signal example	40
3.2	Synthetic PhiHue signal example	40
3.3	Real RPI signal example	41
3.4	Synthetic RPI signal example	41
3.5	Feature-wise Wasserstein distances between synthetic and real Philips Hue, real RPI and real Philips Hue, and synthetic Philips Hue versus real RPI datasets	58
3.6	PCA visualisation comparing feature distributions between synthetic variations (blue) and real RPI measurements (red).	60
4.1	BeagleBone Black: Three representative response traces with the identified response window using optimal parameters	88
4.2	OpenEVSE Charger Board: Three representative response traces with the identified response window using optimal parameters	90
4.3	SIM900 GPRS/GSM Module: Three response traces with the automatically detected full-length response window	91
4.4	SIM900 GPRS/GSM Module: Idle pattern	92
4.5	Raspberry Pi (SSH): Three representative response traces with the identified response window using optimal parameters	94

5.1	Power over averages of the Signal #1	128
5.2	SNR vs measured SNR	130
5.3	SNR vs measured SNR vs CP for stationary signals	132
5.4	SNR vs measured SNR vs CP for non-stationary signals	132
5.5	Dependency between CP, MM and SNR	134
A.1	Distribution of <code>ed_mean</code> for synthetic and real data	173
A.2	Distribution of <code>ed_std</code> for synthetic and real data	173
A.3	Distribution of <code>ed_skewness</code> for synthetic and real data	174
A.4	Distribution of <code>ed_kurtosis</code> for synthetic and real data	174
A.5	Distribution of <code>ed_autocorr_lag1</code> for synthetic and real data	175
B.1	Pairwise scatter plots comparing incremental (Δ) ED-based and CP-based statistical features (mean, standard deviation, kurtosis, and skewness) for synthetic and real embedded-device signals. Synthetic data points are coloured by mean SNR.	177
B.2	Scatter plots comparing mean and standard deviation of ED and CP sequences for synthetic and real embedded-device measurements. The colour scale for synthetic data points reflects the mean SNR values.	178
B.3	Scatter plots depicting relationships between kurtosis and skewness derived from ED and CP sequences. Synthetic signals demonstrate realistic alignment with real embedded-device data distributions.	178
B.4	Scatter plots for spectral (PSD mean) and temporal dependency (autocorrelation lag-1) features derived from ED and CP sequences, comparing synthetic and real embedded-device data. Synthetic data points are color-coded according to their mean SNR values.	179
C.1	Identify response trace	181
C.2	Search optimal params binary search, part 1	182
C.3	Search optimal params binary search, part 2	183
C.4	Multi-offset evaluation	184

List of Tables

3.1	Comparison of signal generation and augmentation libraries across feature categories. Symbols: ✓= supported, O= partially supported, ✗= not supported.	21
3.2	Amplitude and frequency range for various embedded applications.	38
4.1	Command timing scenarios used for evaluating sampling rate and offset search strategies.	84
4.2	Comparison of adaptive binary search and grid search across 10 scenarios. .	86
5.1	Noise/interference types in power analysis, frequency in literature, and impact on trace quality	108
5.2	Quantitative scoring for qualitative frequency and impact levels.	109
5.3	Risk level categorisation based on total score.	110
5.4	Calculated risk scores and levels for different noise and interference sources.	110
5.5	Impact of various noise and interference sources on trace quality and suggested mitigation strategies.	121
5.6	Noted hardware countermeasures against different noise sources.	122
5.7	Final ranking of countermeasures based on number of noise types mitigated, effectiveness, mitigation score, and implementation complexity.	126
5.8	Model comparison for signals with aperiodic noise	131
5.9	Summary of CPA attacks from the literature, including correlation coefficient (cc), noise variance, equivalent switching registers, and used number of traces.	138

List of Abbreviations

AC Alternating Current [44](#), [101](#)

ADC Analog-to-Digital Converter [101](#), [106–108](#), [113](#), [149](#)

AES Advanced Encryption Standard [105](#), [118](#)

AI Artificial Intelligence [130](#)

CAN Controller Area Network [9](#)

CLT Central Limit Theorem [127](#), [134–139](#), [141](#)

CP Comparative Power [3](#), [53–56](#), [59](#), [61](#), [99](#), [127](#), [132](#), [138](#), [139](#), [141](#), [143](#), [144](#), [155](#), [157](#), [176](#)

CPA Correlation Power Analysis [103–105](#), [112](#), [119](#), [125](#), [136](#), [137](#), [140](#), [141](#)

CPU Central Processing Unit [105](#), [118](#)

dB decibel [128](#)

DBA Dynamic Time Warping Barycenter Averaging [144–146](#)

DC Direct Current [44](#), [45](#), [48](#), [115](#)

DFT Discrete Fourier transform [47](#)

DPA Differential Power Analysis [105](#), [112](#), [117](#), [119–121](#), [125](#), [140](#), [141](#), [151](#)

DTW Dynamic Time Warping [116](#), [120](#), [121](#), [139](#), [142](#), [144–146](#), [155](#)

ECU Electronic Control Units [9](#)

ED Euclidean Distance [53–56](#), [59–61](#), [144](#), [176](#)

EMI Electromagnetic Interference [viii](#), [101](#), [108](#), [110](#), [113–115](#), [120](#), [121](#), [125](#), [149](#), [155](#)

FFT Fast Fourier Transform [47](#), [48](#), [114](#)

FPGA Field Programmable Gate Arrays [113](#), [142](#)

GAN Generative Adversarial Network [17](#), [21](#)

GB Gradient Boosting [131](#)

HD Hamming Distance [136](#)

i.i.d. independent and identically distributed [134](#), [135](#)

IC Integrated Circuit [108](#)

ICA Independent Component Analysis [112](#), [119–121](#), [125](#), [126](#), [140](#)

IoT Internet of Things [1](#)

IQR Interquartile Rang [68](#), [69](#), [97](#), [144](#), [155](#)

KDE Kernel Density Estimation [56](#), [57](#), [59](#), [172](#)

LDO Low-Dropout Regulator [116](#)

LPF Low-Pass Filtering [111](#)

LR Linear Regression [131](#)

LSB Least-Significant Bit [106](#)

MAE Mean Absolute Error [133](#), [134](#)

MI Mutual Information [127](#)

ML Machine Learning [113](#), [125](#), [126](#), [130](#), [131](#), [140](#), [143](#)

MLP Multilayer Perceptron [131](#)

MSE Mean Squared Error [131](#), [133](#), [134](#)

OAT one-at-a-time [43](#)

OS Operating System [104](#), [105](#), [107](#), [108](#), [110](#), [117–119](#)

PAT PyArbTools Framework [19](#), [21–24](#)

PCA Principal Component Analysis [55](#), [57](#), [59](#), [62](#), [121](#), [126](#)

PDN Power Distribution Network [101](#)

PRNG Central Processing Unit [119](#)

PSCA Power Side-Channel Attack [100](#)

PSD Power Spectral Density [28](#), [54](#), [61](#), [176](#)

RAPI Remote Application Programming Interface [78](#), [88](#)

REST Representational State Transfer [78](#)

RF Random Forest [131](#)

RMS Root Mean Square [45](#)

RNN Recurrent Neural Network [131](#)

RPI Raspberry Pi [40](#), [51](#), [59](#), [140](#)

SCA Side-Channel Analysis [16](#), [100](#), [108](#), [111](#), [113](#), [127](#), [135](#), [155](#), [157](#)

SNR Signal-to-Noise Ratio [10](#), [12](#), [14](#), [15](#), [36](#), [43](#), [52](#), [53](#), [56](#), [99–101](#), [103](#), [105–108](#), [111](#), [112](#), [114](#), [120](#), [127–134](#), [138](#), [139](#), [144](#), [152](#), [155](#), [157](#), [176](#)

SoC System on a Chip [102](#), [108](#)

SR&OS Sample Rate & Offset Search [3](#), [64–67](#), [69](#), [71](#), [77](#), [86–90](#), [92–98](#), [154–156](#), [180](#)

SSA Singular Spectrum Analysis [112](#), [120](#), [121](#), [125](#), [126](#), [139](#), [140](#), [142](#), [149](#), [150](#), [152](#), [155–157](#)

SUT System under Test [4](#), [67](#), [143](#)

SVD Singular Value Decomposition [112](#), [150](#)

SVR Support Vector Regression [131](#), [138](#)

TSGM Time Series Generative Modeling [21–24](#)

UART Universal Asynchronous Receiver-Transmitter [9](#), [78](#), [79](#)

VAE Variational Autoencoder [17](#), [21](#)

ZCR Zero-Crossing Rate [46](#)

Chapter 1

Introduction

The increasing integration of embedded systems into critical infrastructure has intensified the demand for robust and nonintrusive testing methodologies capable of revealing hidden vulnerabilities and undocumented behaviours. Traditional fuzzing techniques, while effective in general-purpose software testing, are often insufficient for embedded platforms due to hardware constraints and limited observability. To address this, powertrace-based fuzzing has emerged as a promising alternative, leveraging side-channel information to infer internal system behaviour. This thesis focuses on advancing the PowerFuzzer framework by identifying and systematically addressing its key limitations. The central goal of this research is to improve the accuracy, efficiency, and robustness of PowerFuzzer, thereby enabling more reliable side-channel-based testing for a broad class of embedded systems.

1.1 Introduction

Embedded systems are increasingly integral to the modern technological landscape, performing critical roles across sectors including automotive, aerospace, healthcare, and [Internet of Things \(IoT\)](#) infrastructures [92, 112]. Their pervasive deployment and growing complexity demand advanced testing and validation methodologies to ensure reliability, security, and correctness.

Traditional fuzzing approaches, which primarily target software anomalies by observing direct outputs or system crashes [49], often fall short when applied to embedded systems due to their limited observability and tightly coupled hardware-software interactions [34]. Moreover, the rise of sophisticated cyberattacks, such as the SolarWinds breach [8], underscores the critical need to detect undocumented behaviours that may not manifest as

visible failures. Consequently, researchers have sought new methodologies to extract deeper insights about internal system states without invasive techniques.

Powertrace-based fuzzing emerges as a promising solution within this context. By leveraging side-channel analysis, it captures variations in power consumption over time in response to input stimuli, enabling nonintrusive examination of internal device behaviours [104, 43]. This approach builds upon prior work in side-channel-informed coverage analysis and black-box testing of embedded systems [32], extending fuzzing’s reach into domains where conventional techniques often prove inadequate.

An initial version of the *PowerFuzzer* framework, developed as a general-purpose, embedded systems-oriented power-based fuzzing tool, demonstrated the feasibility of side-channel-guided fuzzing [32]. Practical experimentation with this prototype revealed critical challenges, particularly related to precise capture, interpretation, and classification of power traces in noisy and dynamic environments [70].

The present thesis builds upon lessons learned from early *PowerFuzzer* development to address these issues. Specifically, the work targets four significant challenges: precise offset and sampling rate calibration, dynamic response window identification, signal quality enhancement, and robust anomaly classification.

1.2 Thesis Objectives and Contributions

The primary objective of this thesis is to assess the *PowerFuzzer* framework, optimise its performance, and enhance its robustness for effective side-channel fuzzing of embedded systems.

The work focuses on overcoming limitations identified in the early *PowerFuzzer* prototype, improving trace acquisition effectiveness, addressing key noise sources, and strengthening anomaly detection capabilities across diverse operating conditions.

To achieve these objectives, this thesis delivers four primary contributions:

1. **SigVarGen: A Synthetic Signal Generation Framework.** SigVarGen addresses the challenge of insufficient or biased training data by generating synthetic side-channel signals with precisely controlled variations. By incorporating realistic noise patterns, drift, artefacts, and temporal variability, SigVarGen enables more robust algorithm development, benchmarking, and validation against diverse scenarios without overfitting to available limited real-world data.

2. **SR&OS: Adaptive Calibration of Sampling Rate and Offset.** To overcome the problem of misaligned or improperly sampled traces (p_1 and p_2), the [Sample Rate & Offset Search \(SR&OS\)](#) algorithm was developed. It dynamically optimises sampling rates and acquisition offsets through an adaptive binary search strategy enhanced with multi-offset evaluation. [SR&OS](#) isolates accurate response windows, improving the efficiency and accuracy of anomaly detection.
3. **Risk Assessment of Noise Types and Mitigation Strategies.** A systematic matrix-based risk assessment was conducted, quantifying the impact and frequency of diverse noise sources common in embedded side-channel measurements. Correspondingly, noise mitigation strategies were evaluated and ranked based on effectiveness and implementation complexity. The resulting matrix guides the selection of optimal denoising techniques, ensuring targeted and resource-efficient trace quality enhancement.
4. **Practical Evaluation of Signal Quality Metrics for Acquisition Optimisation.** The thesis evaluated existing and newly proposed signal-quality metrics, introducing the [Comparative Power \(CP\)](#) metric alongside correlation-based assessments. These metrics provide practical tools for adaptively predicting the required number of trace acquisitions to achieve the desired signal quality.

Collectively, these contributions yield a strengthened PowerFuzzer workflow, providing robust calibration, precise denoising, and improved anomaly classification capabilities. The advancements presented lay the groundwork for future automated, resilient side-channel fuzzing frameworks tailored for embedded system security testing.

Chapter 2

Powertrace-Based Fuzzing Concept

Powertrace-based fuzzing leverages the analysis of power consumption patterns to detect anomalous or undocumented behaviors in embedded systems. Unlike traditional fuzzing techniques that focus on software outputs or system crashes, this approach monitors the system’s power usage in response to diverse input stimuli, providing deeper insights into the device’s internal states. This chapter introduces the foundational concepts of hardware fuzzing, presents the PowerFuzzer framework and its methodology, details the system architecture, and outlines key challenges in effectively capturing and interpreting power traces for anomaly detection.

2.1 Fundamentals of Hardware Fuzzing

Powertrace-based fuzzing is an emerging method in hardware security testing that leverages power consumption data to derive insights about the internal state of a [System under Test \(SUT\)](#). Unlike traditional fuzzing techniques that rely heavily on software vulnerabilities and direct system outputs, powertrace-based fuzzing monitors variations in power usage as the system processes different inputs. This method falls within the broader category of side-channel fuzzing, which extracts data from physical phenomena like electromagnetic emissions, temperature changes, or power variations to detect anomalous behaviour or system flaws [34]. Such approaches are particularly useful for embedded systems, where conventional output-based debugging mechanisms may be limited or non-existent and where the software is closely intertwined with hardware operation.

Powertrace-based fuzzing shares conceptual similarities with other hardware-based fuzzers,

such as those that utilise electromagnetic side-channel data or acoustic emanations to infer system behaviour [104]. These techniques all serve the critical function of exposing vulnerabilities that are otherwise difficult to detect through standard testing methods. The importance of these side-channel measurements lies in their ability to provide a non-intrusive means of gathering data about a device’s operational state without needing direct interaction with its software or firmware. This capability is indispensable for security testing in environments where the system’s operational transparency is minimal and where traditional testing methods are infeasible.

The effectiveness of powertrace-based and other side-channel fuzzing approaches hinges on the precision and reliability of the measurement setup. Accurate data collection is crucial because the subtlety of side-channel signals often requires high sensitivity and noise-resistant measurement systems. Ensuring accurate and meaningful measurements often involves sophisticated setups with specialized equipment, such as high-resolution oscilloscopes and carefully calibrated sensors. Moreover, the interpretation of side-channel data typically involves advanced signal processing techniques and, increasingly, machine learning algorithms.

2.2 PowerFuzzer Framework Overview and Methodology

This section provides a high-level overview of the developed PowerFuzzer framework. It begins with a formal problem statement 2.2.1 outlining the core challenges in detecting undocumented system behaviours through power analysis. Following this, the Approach and Methodology section 2.2.2 details the hardware and software strategies employed to address these challenges effectively.

2.2.1 Problem Statement

Given a system S and a set of responses R with a set of corresponding powertraces P provoked by a set of inputs D from fuzzing process $\mathcal{F}$, the goal is to assess the power-trace results $R \times P$ for each input and detect anomalies that may indicate undocumented behaviour of the system.

S , the embedded system, is characterised by its response to various inputs $d_i \in D$. We define fuzzing of S as a function $\mathcal{F}$, which maps a set of inputs D to a set of responses

R , observed through powertraces P . A powertrace P for one input d_i is a function $P_i(t)$, where t is a discrete-time variable, and $P_i(t)$ denotes the power consumption of the system S at time t in response to input d_i . The fuzzing process can be represented by Equation 2.1.

$$\mathcal{F} : D \rightarrow R \times P, \quad (2.1)$$

where D is the domain representing the set of all possible inputs, including both legitimate and fuzzed commands and $R \times P$ is the set of all possible responses with a set of corresponding powertraces from the system S .

The fuzzing function $\mathcal{F}$ includes a mechanism to generate an input $d_i \in D$. The input is represented by Equation 2.2.

$$(d_i \in D_{valid} \vee d_i \in D_{fuzz}) \wedge (D_{valid} \subseteq D) \wedge (D_{fuzz} \subseteq D), \quad (2.2)$$

where D_{valid} is set of predefined valid commands and D_{fuzz} is set of mutated commands. d_i is the current generated input from domain D . Each input d_i is applied to the system S through a communication interface.

For each input d_i , the response r_i and the powertrace $P_i(t)$ are linked. The function represented output of the system S is given by Equation 2.3, and the application of the function $\mathcal{O}$ is given by Equation 2.4.

$$\mathcal{O} : D \times S \rightarrow R \times P \quad (2.3)$$

$$\mathcal{O}(d_i, S) = (r_i, P_i(t)) \quad (2.4)$$

where $\mathcal{O}$ represents the combined output function, D is the set of all possible inputs, S is the system, R is the set of all possible responses, and P is the set of all possible powertraces; d_i is a specific input from the set D , r_i is the response of the system S to the input d_i , and $P_i(t)$ is the powertrace generated by the system S in response to the input d_i , as a function of time t .

Powertrace assessment $\mathcal{A}$ defined as $\mathcal{A}(P_i(t)) = a_i$ with $a_i \in \mathbf{R}$ representing the analyzed value of the powertrace $P_i(t)$. The function $\mathcal{A}$ quantitatively evaluates each powertrace $P_i(t)$. Finally, undocumented behaviour should be identified through anomaly detection $\mathcal{D}$ defined by Equations 2.5 and 2.6.

$$\mathcal{D} : \mathbb{R} \rightarrow \{Active_response, Unknown_command\} \quad (2.5)$$

$$\mathcal{D}(a_i) = \begin{cases} Active_response & if a_i \leq \theta \\ Unknown_command & if a_i > \theta \end{cases} \quad (2.6)$$

Where θ is a predefined threshold that distinguishes normal from anomalous power-traces based on their analyzed characteristics a_i .

2.2.2 Approach and Methodology

In alignment with the objectives set forth in Section 2.2.1, power fuzzing aims to identify anomalies that may reveal undocumented behavior in the system S . The PowerFuzzer methodology was developed to detect such anomalies by analyzing the time-dependent power consumption $P_i(t)$ incurred as each input $d_i \in D$ is applied to S . Both legitimate and fuzzed commands ($d_i \in D_{valid} \cup D_{fuzz}$) can trigger distinct power traces, which are then subjected to an anomaly detection function $\mathcal{D}$. This section describes a dual approach outlined by Algorithm 1, covering both hardware and software aspects. Described approach enables precise capture and thorough analysis of respective power traces.

Algorithm 1: Hardware Fuzzing Process Sudo Code

Result: Recorded averaged system responses to inputs D

```

1 initialize iteration_counter = 0;
2 initialize system S;
3 while iteration_counter < n do
4     generate input  $d_i$  from domain  $D$ ;
5     initialize averaged_response  $R_a$ ;
6     for  $j$  in range(number_of_averages) do
7         apply input  $d_i$  to system  $S$ ;
8         observe response  $R_i$  of  $S$  to input  $d_i$ ;
9          $R_a = R_a + R_i$ ;
10    end
11     $R_a = R_a / \text{number\_of\_averages}$ ;
12    process and record  $R_a$ ;
13    iteration_counter = iteration_counter + 1;
14 end
15 check for anomalies in  $S$ ;
```

PowerFuzzer Hardware

From the hardware perspective, the focus lies in capturing power traces $P_i(t)$ under controlled conditions to ensure that transient variations following each command are accurately recorded, where a power trace $P_i(t)$ represents the device’s instantaneous power consumption at discrete times t after receipt of a specific input d_i . To isolate this transient window where the device processes and responds to incoming commands, a dedicated triggering setup must be employed:

- **Trigger Activation.** When the fuzzing process $\mathcal{F}$ transmits an input d_i , the hardware trigger signals the acquisition system to begin recording the power trace.
- **Temporal Window.** Power measurements are gathered during the short duration post-transmission when the system S generates its response r_i .
- **Data Storage.** The resulting power traces are stored in an array or binary format.

The developed approach focuses measurement resources when meaningful variations in $P_i(t)$ are expected. The approach requires alignment to capture intervals with the critical response period. The methodology addresses the need to confine data gathering to those segments most likely to exhibit abnormal power consumption if undisclosed features or hidden states in S are triggered.

Software Dimension

The software layer within the power fuzzing framework orchestrates command generation D , powertrace $P_i(t)$ acquisition, and subsequent analysis $\mathcal{A}$. Specifically, the procedure is as follows:

- **Command Grammar Analysis.** Initially, the accepted command grammar for each target device is surveyed to identify the presence of distinct prefixes or suffixes. This step corresponds to understanding the valid domain $D_{valid} \subseteq D$.
- **Generation of Anomalous Commands.** Building upon the recognized grammar, fuzzed commands ($d_i \in D_{fuzz}$) are generated using a systematic mutation process. An open-source fuzzing platform, BooFuzz, assists in generating diverse inputs that deviate from strictly valid syntax, thereby exploring the broader space $D = D_{valid} \cup D_{fuzz}$.

- **Power Trace Acquisition and Logging.** For each command, legitimate or fuzzed, the corresponding power trace $P_i(t)$ is captured at least 100 times. The repeated acquisition ensures that outliers or measurement noise do not influence the analysis. Command logs are maintained in tandem, providing a clear linkage between the tested input d_i and the resulting power trace $P_i(t)$.
- **Averaging and Statistical Analysis.** The measured traces for each command are averaged to minimize sporadic fluctuations. Subsequently, techniques such as P-value analyses are employed to quantify differences between known and unknown commands. The resulting metrics, denoted $a_i \in \mathbb{R}$, feed into the anomaly detection function $\mathcal{D}(a_i)$. If $a_i < \theta$, the system flags the command’s power trace as potentially indicative of anomalous or undocumented behavior.

2.3 PowerFuzzer System Architecture

The PowerFuzzer setup comprises a target embedded system interfaced via [Universal Asynchronous Receiver-Transmitter \(UART\)](#), a standard communication protocol used in micro-controller based projects, or [Controller Area Network \(CAN\)](#), a message-based protocol used for intercommunication between [Electronic Control Units \(ECU\)](#)’s. The system comprises hardware and software components as outlined by Dunne et al. [32].

The hardware setup encompasses several integrated components crucial for capturing and analyzing the system’s power consumption responses to fuzzing inputs, as Figure 2.1 depicts. The primary elements include a digitizer, a trigger module, an amplifier, a shunt resistor, and the target device itself.

The digitizer serves a dual function in the system: it converts analog power traces into digital form for analysis, and additionally, it houses a trigger channel that synchronizes data acquisition with specific operations or states within the target device based on pre-configured commands. The trigger configures the conditions under which the digitizer initiates data recording. Commands fed into the trigger define the exact moment for starting the data capture, ensuring alignment between the target’s operational state and the recording phase. The trigger passes the command to the target device, typically an embedded system. The SUT receives a sequence of test inputs designed to explore its behavior under varied conditions. The response of the device, particularly changes in power consumption, is relayed back through the shunt resistor and amplifier to the digitizer. Positioned within the power supply line of the target device, the shunt resistor is instrumental in the direct measurement of power consumption. The resistor facilitates an

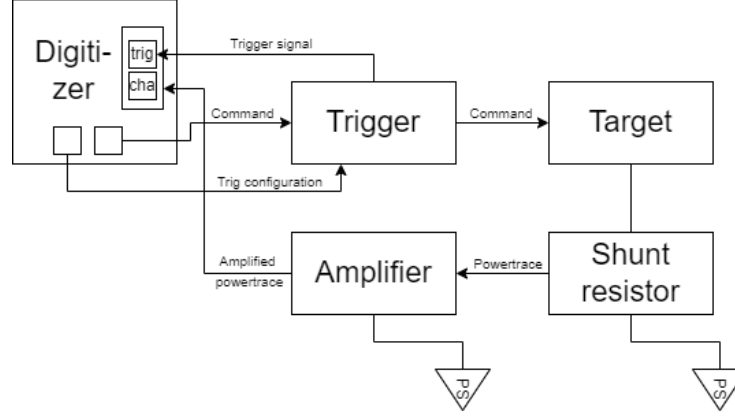


Figure 2.1: PowerFuzzer Architecture

observable voltage drop proportional to the device’s power usage, providing a measurable output that reflects the device’s response to the applied fuzzing inputs. The amplifier later enhances the amplitude of the analog signals detected across the shunt resistor. This amplification is critical to ensure that the subsequent digital conversion by the digitizer retains high fidelity, capturing subtle variations in power consumption that are indicative of different internal operations of the target device.

The outlined hardware setup is directly integrated into the fuzzing process described by Algorithm 1. The fuzzing process is controlled by Powerfuzzer [32] software that manages power trace recording and fuzzed message generation through BooFuzz framework [85]. To mitigate noise from the power traces, the software executes an averaging algorithm that operates simultaneously with data acquisition to accumulate and average power traces, improving signal clarity.

Together, the hardware and software components are able to fulfill the requirement set forth in Section 2.2.1 – to collect, process, and assess power trace data for uncovering anomalies.

However, despite the robust integration of hardware and software components, several challenges persist in effectively capturing and interpreting power traces for anomaly detection. These challenges include determining the specific input that triggered a response, accurately defining the offset and response window, reducing noise to improve the [Signal-to-Noise Ratio \(SNR\)](#), and distinguishing whether a captured trace aligns with known or unknown command distributions. The current version of PowerFuzzer requires manual interventions of analysts to properly work with suboptimal solutions for trace collection, enhancement and analysis. Addressing these issues is critical for enhancing the reliabil-

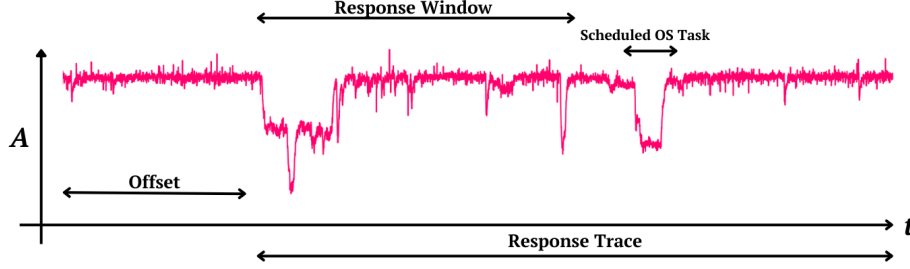


Figure 2.2: PowerFuzzer RPI response trace

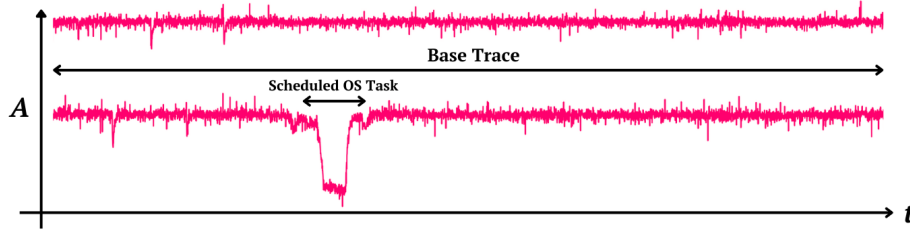


Figure 2.3: PowerFuzzer RPI base traces

ity and precision of the PowerFuzzer framework. The following section describes the core challenges in PowerFuzzer, solutions presented in the following chapters.

2.4 Challenges in Powertrace-Based Fuzzing

Powertrace-based fuzzing seeks to identify undocumented or anomalous behaviours by examining the power consumption traces of a system under test. In this approach, we apply either known commands (D_{valid}) or fuzzed/mutated commands (D_{fuzz}) to the device and capture the resulting power profile over time. Figures 2.2 and 2.3 illustrate two essential types of power measurements involved:

1. **Response Trace (Figure 2.2):** Captured immediately after sending a command. It highlights the system’s power consumption while processing the input and potentially reveals distinctive *response windows*, transient dips, or spikes that may indicate how the device reacts to specific commands.
2. **Base Trace (Figure 2.3):** Represents an “idle” or background measurement without any particular stimulus. These traces frequently exhibit periodic operating sys-

tem (OS) tasks or hardware interrupts, visible as unexpected transients even in nominally idle conditions.

While these traces form the foundation of power-based fuzzing, they also expose five principal problems: (i) ensuring the correct command domain for fuzzing, (ii) calibrating offsets and sampling rates, (iii) adjusting the duration of the response window, (iv) improving SNR amid background activity, and (v) reliably classifying traces as known or unknown. Each challenge corresponds to a key parameter p_0 through p_4 .

(1) p_0 : Defining and Iterating Over the Command Domain.

$$p_0 : d_i \in D, \quad D = D_{\text{valid}} \cup D_{\text{fuzz}}$$

A fundamental challenge lies in defining the command space domain D , which comprises two principal subsets, and selecting specific commands d_i for testing. The first subset of D , D_{valid} , captures the set of all documented (known) commands; identifying these requires collating official device documentation or deducing permissible inputs through observation. The second subset of D , D_{fuzz} , includes mutated or generically generated inputs meant to exhaust device response space in search for atypical or undocumented behaviour. Constructing and iterating over D_{fuzz} is particularly challenging in hardware contexts because traditional coverage-driven metrics are unavailable.

- If *valid commands* are under-sampled, we lack a reliable baseline for comparison.
- If *fuzzed commands* are not sufficiently diverse, we may fail to uncover hidden functionality.
- In Figures 2.2 and 2.3, both the top and bottom traces can reveal how even legitimate commands might provoke background OS tasks; selecting varied commands increases our confidence that observed anomalies arise from the input, rather than random noise.

In this manner, p_0 underscores a primary hurdle of powertrace-based fuzzing: deciding which commands to send, how to structure them into valid and fuzzed categories, and how to navigate an often under-documented hardware interface without the conventional metrics available in software-centric fuzzing frameworks.

(2) p_1 : Offset and Sampling Rate Calibration.

$$p_1 = (t_{\text{offset}}, f_{\text{sample}})$$

This parameter pair ensures that the critical moments of the device’s reaction are *actually captured* within the powertrace. Specifically:

- t_{offset} is the delay (in seconds) between command transmission and the start of power measurement. Incorrect timing can cause partial or complete omission of the response window (see Figure 2.2, top plot, where the main response is visibly shifted).
- f_{sample} is the sampling rate (in samples/second). Under-sampling can obscure transient spikes or dips, while over-sampling may lead to large data overhead and potential synchronization issues.

Communication delays, CPU load, or hardware-specific latencies produce variability in how quickly the device begins processing any given command. A robust calibration of t_{offset} and f_{sample} accommodates these uncertainties, ensuring the captured trace aligns with actual command-induced behavior.

(3) p_2 : Response Window Length Identification.

$$p_2 = T_{\text{response}}$$

The parameter T_{response} defines how long the power measurement continues once the device begins its response. Choosing T_{response} too short risks missing tail-end activity of complex commands, while excessively long durations introduce unrelated background events or OS tasks into the same measurement. In Figure 2.2, we observe variability in how quickly the system completes its response, and in some instances, the second half of the trace shows extraneous tasks creeping into the trace. Accounting for such timing variability becomes critical:

- *Short-duration commands* (e.g., simple status checks) may require only brief observation.
- *Long-duration commands* (e.g., those triggering computationally heavy routines) demand an extended window to capture their full signature.

Without adjusting T_{response} according to each command’s behavioral pattern, one risks confusion between true command effects and random background fluctuations.

(4) p_3 : Trace Quality Improvement (SNR Enhancement).

$$p_3 = \mathcal{A}(P_i(t))$$

In real-world scenarios, the powertrace $P_i(t)$ often contains significant noise:

- **Intrinsic noise:** ADC quantization error, internal sensor distortions.
- **Extrinsic noise:** External electromagnetic interference, high-frequency switching, or concurrent OS processes.

The function $\mathcal{A}$ subsumes various possible processing steps to increase the SNR. These steps might include averaging, smoothing, denoising, background subtraction or other methods. High-quality noise filtering is indispensable since anomalies indicative of undocumented functionality can be subtle and easily masked by background irregularities. Different systems may require fundamentally different trace quality improvement algorithms. For some devices, simple pointwise averaging across multiple traces may suffice, while for others, even sophisticated denoising algorithms could risk excessively smoothing out subtle anomalies, making the careful selection of postprocessing techniques a critical and system-dependent decision.

(5) p_4 : Deciding Normal Versus Anomalous Behavior. A central objective in power fuzzing is to determine whether a newly captured trace corresponds to active system response, or processing an unknown response.

$$p_4 = \mathcal{D}(a_i) = \begin{cases} \text{Active Response,} & \text{if } a_i \leq \theta, \\ \text{Unknown Response,} & \text{otherwise,} \end{cases}$$

where $\mathcal{D}$ is an anomaly detection procedure and a_i denotes features extracted from the preprocessed trace. Various statistical or machine-learning-based methods are employed to operationalise $\mathcal{D}$:

- **Distance/Threshold Methods:** One approach is to quantify the *distance* between the new trace and an “unknown” command distribution (obtained, for example, from random fuzz commands). If the distance is below a threshold θ , the system classifies the trace as an *known* response, signalling that it may belong to the active response set.

- **Statistical Tests (e.g., P-value):** Powertrace features can be compared against a reference distribution using hypothesis testing. If the resulting P-value falls below a chosen significance level, the trace is deemed to deviate substantially from the “unknown” set, suggesting it may not align with the documented behavior.
- **Model-Based Classifiers:** Advanced models, such as autoencoders or neural networks, can be trained on baseline (valid) command data. Deviations in reconstruction error or classification likelihood then indicate anomalous power traces, potentially signalling undocumented commands.

These techniques address the final step of power fuzzing: assigning each measured trace to a *Active Response* or *Unknown Response* class. Robust anomaly detection hinges on a clear separation between documented and fuzzed command signatures, high-quality noise filtration, and well-chosen metrics for comparing observed responses against their expected or learned distributions.

The analysis of Figures 2.2 and 2.3 highlights how each of the five parameters p_0, p_1, p_2, p_3, p_4 directly influences the reliability and precision of powertrace-based fuzzing. In particular, these examples demonstrate how incorrect *offset calibration* and *sample rate calibration* (related to p_1) can obscure the critical response window, how inaccurate *response window identification* (related to p_2) can result in capturing irrelevant background activity, and how unfiltered *background noise* lowers the SNR (related to p_3), reducing the reliability of downstream *anomaly detection* (related to p_4).

This thesis focuses on addressing the challenges associated with p_1 and p_2 – precise offset and sample rate calibration with dynamic response window identification. In addition to these core contributions, the thesis also explores potential techniques to improve p_3 and p_4 . Ultimately, by systematically addressing the problems related to $\{p_1, p_2, p_3, p_4\}$, this thesis aims to enhance the overall effectiveness of powertrace-based fuzzing, strengthening the ability to detect and analyze undocumented functionalities through precise, high-quality power consumption analysis.

Chapter 3

Time-Series Data Generation Framework

Effective development and evaluation of power-based fuzzing rely on access to high-quality datasets. However, hardware constraints, environmental noise, and time demands often limit real-world power trace collection, making large-scale acquisition impractical. Synthetic signal generation offers a scalable alternative by emulating real measurements’ statistical and temporal properties. This chapter presents SigVarGen, a modular time-series data generation framework for synthesising power-like signals, injecting noise, and producing controlled signal variations for [Side-Channel Analysis \(SCA\)](#). It covers the motivation behind synthetic data use, reviews related frameworks, and details SigVarGen’s design, including signal construction, noise modelling, and variation mechanisms. The chapter concludes with implementation details, case studies, and experimental validation of the framework’s effectiveness in supporting fuzzing workflows.

3.1 Synthetic Signal Generation Motivation and Background

Synthetic data generation has emerged as a pivotal tool across numerous domains, including signal processing and machine learning, providing a cost-effective and flexible alternative to real-world data collection [90]. In many applications, obtaining sufficient real data is either prohibitively expensive, logistically challenging, or practically limited. For instance, real-world power trace collection through powerfuzzing is typically a prolonged

and resource-intensive process. Gathering extensive datasets that cover multiple embedded devices, corner cases, and diverse operating conditions can quickly become infeasible due to the time required. Consequently, to ensure robust algorithm development, it becomes essential to rely on realistic synthetic data that comprehensively covers a wide range of scenarios. Under such an approach, algorithms are initially developed and rigorously tested on synthetic datasets, substantially streamlining the process. Subsequent validation in real-world setup then serves primarily as a confirmatory step, effectively reducing empirical testing to a focused validation phase. Synthetic data generation helps bridge the gap between theoretical models and practical implementations by emulating the statistical properties and dynamic behaviour of real signals.

In the field of signal generation, synthetic signals are typically produced through two principal methodologies: simulation of real-world measurement processes or the utilisation of mathematical and algorithmic techniques. Equipment-based simulation emulates the precise measurement conditions associated with specific devices, such as electrocardiogram monitors or industrial sensors, by accounting for various environmental interference, sensor noise, and domain-specific distortions. In contrast, algorithmic approaches involve modelling signal behaviour by applying deterministic functions and probabilistic processes. These methodologies frequently incorporate configurable parameters that allow introducing artificial noise or transformations, thereby simulating a broad spectrum of realistic and hypothetical scenarios [72].

In addition to traditional methods, data-driven generative techniques are emerging as a third category. Techniques like [Generative Adversarial Network \(GAN\)](#)s and [Variational Autoencoder \(VAE\)](#)s suggest learning complex signal dynamics from preexisting datasets. By capturing the statistical dependencies inherent in the data, these methods produce realistic variations that closely mirror observations found in the dataset used for training [90].

The mathematical and algorithmic approaches to synthetic signal generation enable researchers and practitioners to exercise high control over signal parameters for hypothetical situations without hardware, place, and time limitations. Various signal behaviours can be investigated under different conditions using deterministic functions, random processes, and defined transformations. This flexibility allows for targeted experiments that isolate specific variables and enhance research reproducibility. Additionally, fine-tuning parameters in algorithmically generated signals helps create standardised benchmarks for comparing analytical methods, machine learning models, and diagnostic procedures across various fields.

Despite the advancements in synthetic data generation, challenges remain in accurately

modelling noise and artefacts inherent in real-world data. For instance, biomedical signal processing encounters motion artefacts that would require sophisticated artefact removal techniques. In imaging applications, accurately modelling camera sensor noise is crucial for developing effective denoising algorithms [108, 100]. These domain-specific challenges underscore a broader need for synthetic data frameworks capable of simulating realistic noise and artefact characteristics, thereby enhancing the generalisability and robustness of machine learning models deployed in real-world scenarios [93].

The development of SigVarGen is motivated by the need for a comprehensive and systematically structured open-source framework designed to address the problem of complex signal generation through an algorithmic approach and the generation of multiple variants of the same event in a pre-defined environment. Existing frameworks are often fragmented and do not offer the flexibility or scalability necessary to produce diverse event-driven signal variations in such scenarios. Additionally, the absence of integrated modular systems limits the ability to generate extensive and reliable datasets. Furthermore, there is no standardised approach for comparing generated signals with synthetic data to validate outcomes, resulting in challenges related to accuracy and reproducibility. This gap highlights the need for a comprehensive and unified framework designed to generate variations of the same event within a pre-defined environmental context that proves its similarity to real-world data.

3.2 Related Works

As part of our investigation into existing tools, we conducted a systematic search on GitHub using the keywords ‘signal variation generator’, ‘wave variation generator’, ‘trace variation generator’, ‘trace augmentation’, ‘periodic events generator’, ‘periodic signals generator’, ‘time-series generator’, ‘wave augmentation’, ‘signal augmentation’, ‘anomaly generator’. This search, conducted in January 2025, didn’t identify repositories matching similar purposes to our framework. The closest frameworks were identified by ‘time series augmentation’ search (TSGM [77], Tsaug [10]).

Two main comprehensive low-level libraries NumPy and Scipy provide basic functions such as sine wave generation or random noise [51, 117], leaving the user to assemble them into a complete simulation pipeline. These frameworks are flexible but require an understanding of signal-processing practices and a lot of code to build own logic to generate complicated signals similar to real-world ones. Mid-level frameworks offer predefined classes or modules for common signal types, noise models, and transformations. It lets you mix and match components without starting from scratch. The most similar Python frameworks in

the same framework tier we identified were PyArbTools, TimeSynth, Tsaug, Signalz and TSGM.

PyArbTools is a Python-based library designed to streamline the process of waveform creation and remote control of Keysight signal generators [2]. Its primary functionality is enabling users to generate and manipulate complex signal waveforms without relying on proprietary software such as MATLAB or manual SCPI command programming. [PyArbTools Framework \(PAT\)](#) supports a variety of Keysight instruments, including the M8190A/M8195A/M8196A arbitrary waveform generators and N5182B/N5172B vector signal generators, providing users with a seamless interface for configuring device settings, uploading waveforms, and initiating signal playback. The library facilitates the generation of various waveform types—such as continuous wave pulses, amplitude-modulated signals, chirps, multitone signals, and digitally modulated signals—making it particularly valuable in laboratory environments where precision and reproducibility are critical. Moreover, PyArbTools enhances usability by supporting both direct socket communication and integration with PyVISA, offering flexible instrument control across diverse research and testing applications.

PyArbTools distinguishes itself through its specialised integration with Keysight hardware, enabling precise and high-fidelity signal generation in physical environments. Despite its robust functionality in hardware signal generation, PyArbTools presents several limitations that constrain its applicability. Most notably, its reliance on Keysight instruments limits accessibility for users without access to compatible hardware. Additionally, PyArbTools lacks inherent support for advanced signal perturbations, noise injection, and artefact modelling, essential features for simulating real-world conditions in machine learning and algorithm development contexts. PyArbTools is a specialised tool optimised for high-precision signal generation in experimental setups but is less adaptable for comprehensive signal simulation and testing in software-centric workflows.

TimeSynth is an open-source Python library specifically designed to generate synthetic time series data, primarily aimed at facilitating the development and validation of machine learning models [66]. The framework integrates various signal models, including harmonic functions, Gaussian processes, pseudoperiodic signals, and autoregressive models, creating diverse and complex time series patterns. Additionally, TimeSynth incorporates noise modelling capabilities with support for white and red noise. Its modular structure permits the seamless combination of signal generation and noise modelling, further enhanced by built-in support for both regular and irregular time sampling. This flexibility makes TimeSynth particularly valuable for applications requiring controlled simulation of temporal patterns, such as forecasting, anomaly detection, and system modelling.

However, TimeSynth exhibits several limitations that constrain its applicability in more specialised domains. Notably, it cannot produce multiple controlled variations of the same base signal, essential for robustness testing in machine learning and adaptive systems for repetitive events. Furthermore, TimeSynth’s noise modelling is relatively static, offering limited scope for introducing non-stationary noise power.

Tsaug is a Python library designed to augment time series data, providing a diverse range of augmentation techniques to enhance the robustness and generalizability of machine learning models [10]. Its core functionality includes methods such as time warping, cropping, quantisation, drift introduction, reversal, noise addition, and dropout, which allow users to simulate realistic variations in time series datasets. The library supports the construction of complex augmentation pipelines by enabling users to chain multiple augmentation operations with adjustable execution probabilities, offering significant flexibility in data preprocessing workflows. Tsaug accommodates multivariate time series and multi-channel audio data, including speech recognition, sensor data analysis, and financial modelling. Tsaug aims to mitigate overfitting and improve model performance by synthetically expanding limited datasets, particularly in data-constrained environments.

Tsaug primarily focuses on data augmentation rather than the generation of synthetic signals. While Tsaug excels in modifying existing time series through structured transformations and stochastic perturbations, it cannot generate entirely new signals with domain-specific features such as controlled noise injections or dynamic signal disruptions. Tsaug’s augmentation techniques are limited to general-purpose modifications without provisions for introducing domain-specific artefacts or interruptions. In contrast, the developed framework offers a more comprehensive and specialised set of augmentation techniques that extend beyond Tsaug’s general-purpose capabilities. It supports transformations such as time shifting, time warping with data generation, gain variation, amplitude modulation, and intricate forms of baseline drift, allowing nuanced control over signal properties. Tsaug’s lack of support for specialised and domain-tailored augmentations limits its applicability in scenarios requiring realistic environmental variability and signal degradation modelling.

Signalz is an open-source Python library designed to generate synthetic signals and stochastic processes, primarily aimed at facilitating the simulation of diverse time series data for testing and modelling purposes [1]. The library provides a variety of built-in signal generators, including the Lévy walk generator, which simulates heavy-tailed random processes through interpolated Lévy flights, and the Gaussian white noise generator, which produces uncorrelated noise with constant power spectral density. Signalz also incorporates an autoregressive model generator for creating signals based on linear dependencies between current and past values and other stochastic models such as Brownian noise and logistic maps. These capabilities allow Signalz to simulate a broad spectrum of random and

deterministic processes, making it suitable for applications in fields like financial modelling, physics-based simulations, and random walk analysis.

Signalz primarily provides generic stochastic signal generation without offering advanced augmentation techniques or domain-specific customisations. It does not facilitate the generation of controlled signal variations or the simulation of real-world disruptions, thus limiting its applicability in complex signal modelling scenarios.

Time Series Generative Modeling (TSGM) framework is a tool for generating, augmenting, and evaluating synthetic time series data [77]. It implements models such as **GANs**, **VAEs**, and probabilistic structural models to create time series data for various applications. The framework includes augmentation techniques such as time warping, magnitude warping, and additive noise, which modify data for training and evaluation purposes. **TSGM** is built on TensorFlow, enabling integration with machine learning workflows and optimising computation for large datasets. It also provides evaluation metrics that measure generated data’s similarity, predictive performance, and statistical properties. **TSGM** can be used for tasks such as synthetic data generation, anomaly detection, and simulation-based forecasting.

While **TSGM** supports deep generative models and offers augmentation techniques such as time warping and additive noise, it lacks control over signal composition and variation logic. Specifically, TSGM does not support the injection of custom perturbing signals or event-based anomaly simulations such as deterministic interrupts, glitches, or bursts, which are common in embedded and sensor systems.

Table 3.1 summarises and compares the capabilities of analysed signal generation and augmentation frameworks across multiple feature categories, highlighting their strengths, limitations, and relative scope of application.

Table 3.1: Comparison of signal generation and augmentation libraries across feature categories. Symbols: ✓= supported, ○= partially supported, ✗= not supported.

Feature	SigGenVar (Our)	PAT [2]	TimeSynth [66]	Tsaug [10]	Signalz [1]	TSGM [77]
Signal Generation Scope						
Generation of New Signals	✓	✓	✓	✗	✓	✓
Domain-Specific Signals	✓	✓	✗	✗	✓	✓

Continued on next page

Continued from previous page

Feature	SigGenVar (Our)	PAT [2]	TimeSynth [66]	Tsaug [10]	Signalz [1]	TSGM [77]
Sine/Cosine Waveforms	✓	✓	✓	✗	✓	✓
Square, Triangle, Ramp Up, Ramp Down, or DC waveforms	✗	✓	✗	✗	✗	✗
Support for Deep Generative Models	✗	✗	✗	✗	✗	✓
Logic for Custom Perturbations	✓	✗	✗	✗	✗	✗
Stochastic Models	✗	✗	✓	✗	✓	✓
Irregular/Non-Uniform Time Sampling	✗	✗	✓	✗	✗	○
Logic for Periodic Activity	✓	✗	✗	✗	✗	○
Noise Modeling						
Additive White Gaussian Noise	✓	✗	✓	✓	✓	✓
Colored Noise	✓	✗	✓	✗	✓	✗
Non-Stationary Power	✓	✗	✗	✗	✗	✗
FFT-Based Noise Generation	✓	✗	✗	✗	✗	✗
Artifact and Perturbing Signal Injection						
Addition of Interrupt Signals	✓	✗	○	✗	○	○
Controlled Interrupt Scheduling	✓	✗	✗	✗	✗	✗

Continued on next page

Continued from previous page

Feature	SigGenVar (Our)	PAT [2]	TimeSynth [66]	Tsaug [10]	Signalz [1]	TSGM [77]
Randomized Burst and Glitch Simulations	✓	✗	✗	✗	✗	✗
Augmentation and Transformation						
Time Shifting	✓	✗	✗	✓	✗	✗
Time Warping	✓	✗	✗	✓	✗	✓
Amplitude Mod- ulation	✓	✓	✗	✓	✗	✓
Baseline Drift	✓	✗	✗	✓	✗	✗
Slice and shuffle	○	✗	✗	✗	✗	✓
Region-Specific Transformations	✓	✗	✗	✗	✗	✗
Chained/Pipelined Augmentations	✓	✗	✗	✓	✗	✗
Variants Parameter-Space Exploration						
Multi-Variant Signal Genera- tion	✓	✗	✗	✗	✗	✓
Systematic Pa- rameter Sweeping	✓	✗	✗	✗	✗	✗
Controlled Sam- ple Sub-Range Randomization	✓	✗	✗	✗	✗	✗
Extensibility and Customization						
Software-Based Solution	✓	○	✓	✓	✓	✓
Hardware Integration (SCPI/PyVISA)	✗	✓	✗	✗	✗	✗
Modular Frame- work Design	✓	✓	✓	✓	✓	✓
Custom Transfor- mations Support	✓	○	✓	✓	✓	✓

Continued on next page

Continued from previous page

Feature	SigGenVar (Our)	PAT [2]	TimeSynth [66]	Tsaug [10]	Signalz [1]	TSGM [77]
Performance and Scalability						
Batch Processing/Scalability	✓	○	✓	✓	✓	✓
Parallel Processing Friendly	✓	○	✓	✓	✓	✓
High-Precision Signal Configuration	✗	✓	✗	✗	✗	✗
Evaluation Metrics and Benchmark	○	✗	✗	✗	✗	✓
Feature Count (Score)	27	10	11.5	11	11.5	17.5

In summary, SigVarGen addresses limitations found in existing synthetic signal generation frameworks by integrating flexible baseline signal synthesis with controlled noise injection, signal interruptions, and systematic augmentation techniques. Unlike specialised libraries such as PyArbTools, which rely heavily on specific hardware, or general-purpose tools like Tsaug, TimeSynth, or TSGM, which lack precise control over perturbations and realistic, domain-specific signal generation, SigVarGen provides fine-grained control for simulating multiple variations of stable events under defined environmental conditions. By enabling realistic noise conditions, event-driven artefacts, and temporal variations, the framework produces synthetic datasets closely matching the statistical properties of real-world data acquired through the PowerFuzzer setup. SigVarGen thus supports reliable development, testing, and validation of signal processing and machine learning algorithms for side-channel analysis, bridging the gap between theoretical model development and empirical validation.

3.3 Methodology and Design

This section presents the core algorithms underlying the SigVarGen framework, encompassing signal generation, noise modelling and injection, signal variation and augmentation

techniques used to simulate consecutive measurements. Each module is accompanied by relevant pseudocode to illustrate the implementation logic.

3.3.1 Signal Generation

Signals are constructed by crafting a foundational sinusoidal waveform [52] with sporadic interruptions. The foundational waveforms, termed as base signals, are formed by overlaying multiple sinusoidal waves. The Fourier Transform principles justify a dataset consisting of aperiodic waves of sinusoidal forms. Any aperiodic signal $x(t)$ can be extended to a periodic signal $x_P(t)$ by repeating it after a certain period T_0 [12], where $x_P(t)$ can be represented as equation 3.1. This artificial repetition allows for a Fourier series representation of $x_P(t)$, and as T_0 approaches infinity, the series converges to the continuous Fourier Transform of the original aperiodic signal $x(t)$. This is because the spacing between spectral lines becomes infinitesimally small, turning the summation into an integral [12]. Similarly, Fourier’s Theorem elucidates that any periodic signal can be decomposed into pure sinusoidal components [3]. By extension, if an aperiodic signal is artificially made periodic by repetition, the same decomposition principles apply. Therefore, the aperiodic waves in the real world, though naturally not repeating, can theoretically be considered as being composed of an infinite continuum of sinusoidal waves when analyzed through the lens of the Fourier Transform.

$$x_P(t) = \sum_{n=-\infty}^{\infty} C_n e^{j2\pi n f_0 t}, \quad (3.1)$$

where C_n are the coefficients given by equation 3.2.

$$C_n = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x_P(t) e^{-j2\pi n f_0 t} dt \quad (3.2)$$

And as $T_0 \rightarrow \infty$, these equations transform into the Fourier Transform for a non-periodic function $x(t)$, justifying the dataset composed of aperiodic waves from sinusoidal forms.

The `generate_signal` function operationalises the concept of constructing realistic composite waveforms by summing multiple sinusoidal components, each characterized by randomly assigned amplitude, frequency, and phase within predefined intervals [36]. This stochastic generation approach ensures diversity and realism, mirroring complex, naturally occurring signals composed of various harmonic elements. Post-generation normalisation

aligns the resulting waveform within user-specified amplitude limits, ensuring signals remain bounded and suitable for further processing, analysis, or real-world simulation. The randomisation procedure employed draws inspiration from Fourier’s principles, which indicate that virtually any complex or natural signal can be represented or approximated by a suitable combination of sinusoidal components.

The following pseudocode outlines the `generate_signal` function, which forms the foundation of the composite signal synthesis. The loop structure adheres to the methodology discussed by Haslwanter [52] and Esakkirajan et al. [36].

Algorithm 2: Composite Signal Generation (`generate_signal`)

Input: Time vector t , number of sinusoids n , amplitude range $[A_{\min}, A_{\max}]$, frequency range $[f_{\min}, f_{\max}]$

Output: Composite signal $s(t)$, sinusoid parameters list P

```

1 Initialize  $s(t) \leftarrow \mathbf{0}$  (vector of zeros with same length as  $t$ );
2 Initialize  $P \leftarrow []$ ;
3 for  $i \leftarrow 1$  to  $n$  do
4    $a \leftarrow \text{Uniform}(A_{\min}, A_{\max})$ ;
5    $f \leftarrow \text{Uniform}(f_{\min}, f_{\max})$ ;
6    $\phi \leftarrow \text{Uniform}(0, 2\pi)$ ;
7    $s_i(t) \leftarrow a \cdot \sin(2\pi f \cdot t + \phi)$ ;
8    $s(t) \leftarrow s(t) + s_i(t)$ ;
9   Append  $\{\text{amp} : a, \text{freq} : f, \text{phase} : \phi\}$  to  $P$ ;
10 end
11  $s(t) \leftarrow s(t) - \text{mean}(s(t))$ ;
12 Let  $M \leftarrow \max\{|s(t)|\}$ ;
13 if  $M = 0$  then
14   | error: "Generated signal has zero amplitude";
15 end
16  $s(t) \leftarrow s(t)/M$ ;
17 Rescale  $s(t)$  to  $[A_{\min}, A_{\max}]$  using:
18    $s(t) \leftarrow \left(\frac{s(t)+1}{2}\right) \cdot (A_{\max} - A_{\min}) + A_{\min}$ ;
19 return  $s(t)$ ,  $P$ ;
```

Complementing this, the framework integrates additional functions that introduce controlled signal interruptions or perturbations, simulating realistic signal changes in response to events encountered in practical settings. For instance, the methods designed to insert main interrupts or bursts within signals involve strategically placing and blending addi-

tional sinusoidal or semi-periodic waveforms. These interruptions can take various forms: large-scale modifications known as main interrupts, smaller secondary perturbations, or short bursts of periodic digital signals, each customisable in terms of duration, placement, amplitude offset, and complexity.

3.3.2 Noise Generation and Addition

Noise can be described as any unwanted or irrelevant interference superimposed on a signal of interest originating from external sources or stochastic variations inherent to the signal. In its broadest sense, noise reflects unknown activity, while known or understood activity is considered part of the desired signal [21]. In practice, this distinction between noise and signal is fluid, as improvements in understanding a system’s internal or external processes can recast previously unknown sources of interference into meaningful information-bearing signals. From a communications and signal-processing standpoint, noise not only impairs the clarity of transmitted or recorded information but also serves as a carrier of its own information regarding the environment and system state [115]. It may arise from diverse phenomena: electronic (thermal, shot), acoustic (machines, collisions, wind), electromagnetic (radio interference), electrostatic (voltages), or network-based (packet loss, quantisation). The overarching goal in most systems, ranging from telecommunication channels to biomedical measurement, is the effective modelling, mitigation, or removal of these noise components. Moreover, noise can be classified according to its stationarity: stationary noise retains consistent statistical properties over time, whereas non-stationary noise exhibits changing properties, whether abruptly (e.g., sudden equipment malfunctions) or gradually (e.g., evolving environmental factors) [41, 115]. Understanding these distinctions is crucial for developing robust noise-reduction and signal-enhancement strategies.

The approach used in this work follows the frequency synthesis method introduced by Poul Bourke [18], leveraging the observation that many natural phenomena exhibit noise spectra characterised by a $1/f^p$ relationship, where the spectral power decreases inversely proportional to some power p of frequency. This spectral behaviour directly correlates with fractal dimension characteristics observed in natural signals.

The synthesis of coloured noise proceeds in four primary steps:

1. Generation of white noise in the time domain.
2. Transformation of the noise into the frequency domain using the Fourier transform.

3. Scaling of the noise spectrum according to a desired frequency-dependent function ($1/f^p$).
4. Inverse Fourier transformation back to the time domain.

This method allows precise control over the noise spectral characteristics, enabling the generation of broadband noise types – such as white, pink ($1/f$), and brown ($1/f^2$) – which differ in their spectral power distributions [73]. Specifically, white noise has equal spectral power across all frequencies, while pink and brown noises exhibit spectral power distributions decreasing at varying rates with increasing frequency.

In practice, the algorithm implemented as `add_colored_noise` directly embodies this principle. Its steps can be summarized in pseudocode given by Algorithm 3. To achieve a $1/f$ **Power Spectral Density (PSD)** in noise generation, the amplitude spectrum must be scaled by $1/f^2$, since power is proportional to the square of amplitude, ensuring that the desired spectral decay is correctly maintained.

Algorithm 3: Colored Noise Generation with Envelope Modulation

Input: signal, noise_power, noise_color, modulation_envelope, modulation_factor
range (mf), noise power window (npw)

Output: noisy_signal

- 1 Generate white noise $w(t)$ with zero mean and unit variance;
 - 2 Compute frequency spectrum $W(f)$ of $w(t)$ via FFT;
 - 3 **if** *noise_color* is 'pink' **then**
 - 4 | Scale $W(f)$ by $1/\sqrt{f}$;
 - 5 **else if** *noise_color* is 'brown' **then**
 - 6 | Scale $W(f)$ by $1/f$;
 - 7 **else if** *noise_color* is 'white' **then**
 - 8 | Keep $W(f)$ unchanged;
 - 9 Inverse FFT of scaled $W(f)$ to obtain colored noise $n(t)$;
 - 10 Normalize $n(t)$ to zero mean and unit variance;
 - 11 Scale $n(t)$ to desired noise power within the specified range npw ;
 - 12 **if** *modulation_envelope* is defined **then**
 - 13 | Generate modulation envelope $e(t)$ using selected envelope function (linear, sine, random walk, or blockwise);
 - 14 | Apply envelope modulation: $n(t) \leftarrow n(t) \times e(t)$;
 - 15 Apply modulation factor mf randomly chosen within given limits to the original signal amplitude;
 - 16 Combine modulated signal with generated colored noise: $\text{noisy_signal} \leftarrow (\text{signal} \times \text{modulation factor}) + n(t)$;
-

To enhance the realism and complexity of generated noise, we introduced four types of time-varying envelopes: *linear*, *sine*, *random walk*, and *blockwise*. Each envelope modulates noise amplitude differently and contributes non-stationary characteristics commonly observed in natural acoustic signals [41]. For instance, the *sine envelope*, particularly when utilising minor modulation frequencies (e.g., 0.01 Hz), imposes periodic fluctuations on the noise power, introducing cyclostationarity. Such cyclostationary noise is characterised by periodic modulation of statistical properties, such as variance and autocorrelation, and exhibits frequency-dependent correlations [89].

3.3.3 Signal Variation and Augmentation

In many practical measurement systems, repeated observations of the same event exhibit not only stochastic noise variations but also subtle yet significant changes in the underlying

signal itself. These variations may arise due to environmental dynamics, sensor drift, or inherent system instabilities, thereby necessitating a systematic approach to model such variability. The variation module addresses this need by leveraging parameter space exploration and tailored signal transformation functions.

The module achieves its objectives through two primary processes: generating a set of parameter configurations that govern various transformation operations and applying these configurations to produce individual signal variants. The parameter generation process, encapsulated in the `generate_parameter_variations` function, systematically selects sub-ranges of possible parameter values to mimic the inherent uncertainty and variability in repeated measurements. Then, the `generate_variation` function applies a sequence of operations, including time shifts, time warping, gain variation, amplitude modulation, and baseline drift, to the original signal. This layered approach simulates the impact of environmental noise and captures the dynamic behaviour of the signal, thus enabling robust performance evaluation of signal processing algorithms.

The detailed steps involved in parameter variations are described below:

Algorithm 4: Pseudocode for `generate_parameter_variations`

Input: `param_sweeps` (dict), `num_variants` (int), `window_size` (int)

Output: `variations` (list of dict)

```

1 Initialize empty list variations;
2 for param in param_sweeps do
3   Randomly select center index within the length of parameter values array;
4   Choose subrange around center index defined by window_size;
5   Store subrange for each parameter in chosen_subranges;
6 end
7 for variant in num_variants do
8   for parameter in chosen_subranges do
9     Randomly select parameter value within subrange;
10  end
11  Randomly define  $f_{min}$ ,  $f_{max}$ , ensuring  $f_{min} < f_{max}$ ;
12  Randomly select a value for wave_with_score;
13  Append parameters to variations;
14 end
15 Return variations;

```

Subsequently, each variant generated from the above method is augmented through a series of signal transformations using the `generate_variation` function. Transformations include controlled interruptions, time domain modifications, amplitude modulations, and

detailed baseline drifts. Importantly, the function allows flexible control of these augmentations, reflecting realistic disturbances such as sensor fluctuations, environmental interferences, or acquisition anomalies, crucial for comprehensive evaluation of signal processing algorithms.

The pseudocode describing this comprehensive augmentation procedure is presented below:

Algorithm 5: Pseudocode for `generate_variation`

Input: `transformed_wave` (np.array), `variant_params` (dict), `t`, `n_sinusoids`,
`amplitude_range`, `base_frequency_range`, `interrupt_params` (list of dict)

Output: `transformed_wave` (np.array)

```

1 if wave_with_score > threshold then
2   | Substitute signal segments with newly generated waveform segments based on
   | score;
3 end
4 Apply transformations sequentially based on parameters:
5 if time_warp_enabled then
6   | Apply time_warp to transformed_wave;
7 if time_shift_enabled then
8   | Apply time_shift to transformed_wave;
9 Apply gain_variation transformation;
10 Apply global amplitude_modulation;
11 Apply region-specific amplitude_modulation_region transformation using
   parameters f_min, f_max;
12 Apply global baseline_drift;
13 Apply region-specific baseline drift using baseline_drift_region;
14 Return transformed_wave;
```

This variation module not only enriches the synthetic dataset with realistic fluctuations but also provides a robust framework for evaluating signal processing algorithms under conditions that closely mirror real-world measurement variability.

3.4 Implementation Details

This section outlines the practical aspects of the SigVarGen framework’s implementation. It starts by describing the Python environment and required dependencies. Next, it outlines the overall framework structure and presents key modules and functions for signal

generation, noise injection, signal variation, supporting functions and possible configurations.

3.4.1 Python Environment and Dependencies

The project is implemented using Python, with a minimum version requirement of 3.8. The development environment is built upon baseline libraries such as NumPy for efficient numerical computations, SciPy for advanced scientific and signal processing functions, and Matplotlib for visualisation purposes. In addition, testing is facilitated through pytest. The framework incorporates specialised modules tailored for signal processing and noise generation, enabling precise manipulation and augmenting synthetic signals. The project's dependency management is defined in a standardised `pyproject.toml` file, ensuring reproducibility and ease of installation across different systems.

3.4.2 Framework Structure

The project is organised in a modular fashion to promote clarity, maintainability, and scalability. The codebase is structured into distinct directories, each encapsulating a specific functional area. The **SigVarGen** package forms the core of the project and includes submodules for signal generation (**signal**), noise modelling (**noise**), and signal variations (**variations**), alongside utility functions (**utils**) and function parameter examples (**config**). The **signal** submodule comprises functions for generating base waveforms, applying periodic interruptions, and integrating response signals triggered by events. The **noise** submodule contains implementations for various noise types and envelope functions to simulate stationary and non-stationary noise profiles. The **variations** submodule is responsible for signal augmentation techniques such as baseline drift and transformations, allowing the generation of diverse signal variations for more comprehensive dataset creation. The **config** module contains parameter definitions that govern signal generation and augmentation. This includes predefined amplitude and frequency ranges for different device types, noise parameters specifying power variations and spectral properties, and variation parameters for controlled augmentation of signals.

Supporting these are additional directories such as **tests**, which house unit tests ensuring code quality and correctness, and **tutorials**, which provide practical examples and usage guides. The project adheres to established coding standards, including comprehensive docstrings. The approach ensures that each component is testable, and easily extendable.

3.4.3 Key Functions and Modules

This subsection describes five SigVarGen modules and all their functions supporting signal generation, noise injection, variation handling, and configuration management.

Signal Generation

The signal generation submodule is designed to synthesise complex, realistic signals by combining multiple sinusoidal components with targeted interruptions and modifications. The primary components of the framework are described as follows:

- **generate_signal:** This function produces a composite signal by summing a set of sinusoids. For each of the n sinusoids, a random amplitude, frequency, and phase are drawn from the specified ranges. The resulting signal is normalized to have zero mean and scaled to fit within a designated amplitude interval. The function also returns the parameters of each constituent sinusoid, allowing for detailed subsequent analysis.
- **get_non_overlapping_interval:** This routine searches for a valid time interval in the signal that does not overlap with previously allocated regions.
- **place_interrupt:** Serving as a wrapper for interval selection, this function either leverages the non-overlapping interval finder to determine a valid placement for an interrupt based on a specified duration ratio or chooses random placement within time bounds.
- **blend_signal:** A simple linear combination function, which blends two signal segments using a specified blend factor. This enables smooth integration of the interrupt signal and the base waveform.
- **apply_interrupt_modifications:** This function adjusts the interrupt signal to align with device limitations by introducing baseline drifts and offsets. The modifications ensure that the interrupt remains within the physical constraints of the system, either by raising or lowering the baseline.
- **generate_main_interrupt:** Building on **generate_signal**, this routine creates the primary interrupt waveform by selecting amplitude and frequency ranges according to a given domain. The resulting signal serves as the raw interrupt that is later positioned within the base signal.

- **add_main_interrupt:** This function is responsible for inserting the main interrupt into the base signal. It determines the appropriate location using the placement functions and applies the necessary modifications. Additionally, it can incorporate a series of overlapping interrupts within the main interrupt region.
- **add_complexity_to_inter:** This auxiliary routine embeds a smaller interrupt within the main interrupt region. By adjusting the duration and offset, it adds further complexity to the overall signal structure.
- **add_smaller_interrupts:** For added signal variability, this function inserts multiple secondary interrupts into the base signal. Each interrupt is generated, placed, and modified to ensure proper integration without unwanted overlap.
- **add_interrupt_with_params:** This high-level function combines the main and secondary interrupts into one composite modification. It orchestrates the placement, blending, and modulation of the interrupts.
- **add_interrupt_bursts:** This function generates clusters of small interrupt bursts within a defined time window. It is particularly useful for simulating irregular, burst-like disturbances in the signal.
- **generate_semi_periodic_signal:** This routine produces a semi-periodic digital signal by repeating a base binary pattern, with an added random bit-flipping mechanism to introduce system latency. Such signals are used to simulate periodic digital interruptions.
- **add_periodic_interrupts:** This function incorporates periodic digital interruptions into a continuous base signal. It modulates the interrupt signal using two phases—before/after and during the main interruption—to produce distinct patterns of interference.

Collectively, these functions form a cohesive signal generation pipeline for generating idle signals and signals with embedded interrupt signals. The design allows for the controlled synthesis of complex waveforms, which can be tailored to meet the specifications of various experimental and real-world scenarios.

Variation Generation

The variation submodule introduces time-series augmentation functions that allow the simulation of the variability of the system state between different measurements. The

following describes the functions integrated within the SigVarGen framework:

- **generate_parameter_variations:** This function generates diverse sets of parameter configurations to produce multiple signal variants. Given a dictionary of parameters ('param_sweeps'), the function randomly selects subranges around center points within specified windows. For each variant, parameters are randomly drawn from these selected subranges, ensuring systematic yet randomized parameter space exploration.
- **generate_variation:** Creates a transformed version of an input waveform by applying a comprehensive series of augmentations based on parameters provided by 'generate_parameter_variations'. It introduces transformations such as amplitude modulation, gain variations, time shifts, time warping, baseline drift (both global and region-specific), and interrupt signals.
- **apply_time_shift:** Introduces temporal variations by circularly shifting the waveform randomly within a defined maximum shift interval. This simulates temporal offsets commonly observed in real-world signal acquisition scenarios.
- **apply_time_warp:** Applies non-linear time warping to the signal, compressing or expanding temporal segments to simulate natural fluctuations or timing irregularities. Missing segments introduced due to warping are filled with newly generated signal segments, ensuring continuity and realism.
- **apply_gain_variation:** Modifies signal amplitude by multiplying the waveform by a randomly selected gain factor within specified limits. This represents variations in sensor sensitivity or environmental effects impacting the signal amplitude.
- **apply_amplitude_modulation:** Implements global sinusoidal amplitude modulation, introducing periodic amplitude variations that mimic realistic conditions, such as fading or cyclical interference.
- **apply_amplitude_modulation_region:** Applies localized amplitude modulation to a specific region of the waveform defined by fractional indices ('f_min', 'f_max'). This targeted transformation simulates scenarios where interference or amplitude fluctuations occur over specific temporal windows.
- **apply_baseline_drift:** Introduces a global linear drift across the entire signal duration, representing gradual baseline shifts due to device instability or environmental influences.

- **apply_baseline_drift_region:** Adds controlled baseline drift to a specified region of the signal, simulating temporary sensor baseline shifts or localized environmental influences, crucial for realistic artifact simulation.
- **apply_baseline_drift_middle_peak:** Generates a baseline drift with a prominent peak (positive or negative) at the midpoint of the waveform, returning to zero at endpoints. This simulates temporary spikes or dips in baseline commonly occurring in biomedical or environmental data acquisition.
- **apply_baseline_drift_polynomial:** Introduces baseline drift shaped by polynomial functions (e.g., quadratic), simulating complex, gradual, and realistic sensor baseline shifts across the entire signal duration. Useful for scenarios where drift profiles are non-linear and prolonged, common in biomedical monitoring and sensor data.
- **apply_baseline_drift_piecewise:** Implements piecewise-linear drift by dividing the signal into multiple segments, each with independently randomized drift amplitudes. This models complex drift behaviors observed in prolonged signal acquisition scenarios.
- **transform_wave_with_score:** Inserts segments of newly generated waveforms into the existing signal at randomized positions with specified probability.
- **apply_quantization_noise:** Adds quantization noise to the signal by discretizing amplitude values.

Collectively, these functionalities enable SigVarGen to generate multiple variants of the same synthetic signal and capture complex real-world variations. Such signal augmentation and parameter variability capabilities are fundamental for producing robust synthetic datasets.

Noise Generation and Addition

The noise generation and addition submodule is designed to introduce controlled noise into signals, facilitating the simulation of realistic noisy environments. The core functionalities of this submodule are outlined below:

- **generate_noise_power:** This function determines the power of the noise to be added to a given signal by selecting a [SNR](#) value from a specified range. The noise power is computed by first obtaining the standard deviation of the input signal and then applying a scaling factor derived from the selected [SNR](#).

- **add_colored_noise:** This function applies noise with a specific spectral profile (white, pink, brown, blue, violet or custom) to an input signal. The process begins with the generation of a white noise sequence, which is then transformed into the desired noise colour by applying a frequency-domain filter. The noise is normalised and scaled to achieve the intended power level before being combined with the input signal. A modulation factor may be applied to introduce amplitude variations across different measurements. The function returns a combination of the final signal with generated noise and the isolated noise component.

Additionally, noise may be modulated with the envelope function. Specifically, the implemented envelopes function as follows:

- **envelope_linear:** Smoothly increases or decreases the amplitude of the noise over time, introducing gradual trends in noise intensity.
- **envelope_sine:** Modulates noise amplitude periodically, creating cyclic fluctuations with adjustable frequency parameters, making it suitable for simulating repetitive or rhythmic environmental variations.
- **envelope_random_walk:** Implements stochastic amplitude changes modelled as a random walk, simulating unpredictable natural variations and temporal irregularities.
- **envelope_blockwise:** Applies abrupt, step-like variations in noise amplitude, producing intermittent or segmented noise intensities typically observed in dynamically changing environments.

This combination of coloured noise generation and envelope-based amplitude modulation enables customisable, realistic simulation of both stationary and non-stationary noise characteristics.

Configuration

The configuration submodule defines pre-specified parameter ranges for diverse embedded devices and scenarios. Its primary goal is to facilitate a structured generation of realistic signals representative of different device operational characteristics. Parameters for each device include defined amplitude and frequency ranges, selected to reflect typical real-world operating conditions encountered in applications such as communications, robotics,

Device	Amplitude (V)	Frequency (Hz)
Arduino Board	(0, 5)	(0, 12e3)
Drones (control)	(0, 1)	(2.398e9, 2.402e9)
Drones (telemetry_low)	(0, 1)	(432e6, 434e6)
Drones (telemetry_high)	(0, 1)	(2.39e9, 5.9e9)
Cameras	(0, 1)	(24, 120)
Smartphones (LTE Low)	(0, 1)	(699e6, 701e6)
Smartphones (LTE High)	(0, 1)	(2.598e9, 2.602e9)
Smartphones (5G Low)	(0, 1)	(599e6, 601e6)
Smartphones (5G High)	(0, 1)	(38.9e9, 39.1e9)
Wi-Fi Routers (2.4GHz)	(0, 1)	(2.395e9, 2.405e9)
Wi-Fi Routers (5GHz)	(0, 1)	(4.990e9, 5.010e9)
Wi-Fi Routers (6GHz)	(0, 1)	(5.990e9, 6.010e9)
Smart Watches	(0, 0.9e-3)	(2.398e9, 2.402e9)
Home Automation (Zigbee)	(0, 0.9e-3)	(2.398e9, 2.402e9)
Home Automation (Z-Wave EU)	(0, 0.9e-3)	(867e6, 869e6)
Home Automation (Z-Wave US)	(0, 0.9e-3)	(907e6, 909e6)
Automotive Sensors (US)	(0, 0.01)	(314e6, 316e6)
Automotive Sensors (EU)	(0, 0.01)	(432e6, 434e6)

Table 3.2: Amplitude and frequency range for various embedded applications.

consumer electronics, and automotive sensing [94, 37, 57, 96, 121, 28]. Device ranges are summarised in Table 3.2.

To support variability within controlled limits, the framework utilizes pre-defined parameter sweeps for each type of device. Moreover, configurable noise envelopes (linear, sine, random-walk, blockwise), modulation factors, and noise power windows are also provided. These additional configuration options significantly enhance the framework’s ability to simulate realistic scenarios, accounting for practical constraints and environmental conditions.

The configuration parameters support the systematic and reproducible exploration of the operational parameter space, enabling users to generate signals closely aligned with real device behaviours.

Utils

The utils submodule comprises some auxiliary functions helpful for preprocessing, evaluation, and device-parameter handling, thereby supporting the overall signal generation and augmentation process functionality. This submodule enables standardized signal processing steps that enhance reproducibility and comparability across generated signals.

The primary functionalities of the utils submodule include:

- **calculate_SNR**: Computes the power ratio (in dB) between a signal and its noise component, providing an intuitive metric to quantify signal purity and noise impact.
- **calculate_ED**: Determines the Euclidean distance between two given signals, facilitating quantitative comparisons essential in evaluating similarity and variability among signal variants.
- **interpolating**: Resamples signals to a specified target length using linear interpolation. This function is critical in standardizing signal lengths across multiple variants, particularly for comparative analyses and machine learning applications.
- **normalization**: Standardizes signals by subtracting the mean and scaling to unit variance. Normalization ensures comparability and reduces biases due to varying signal amplitudes or offsets.
- **generate_device_parameters**: Splits defined device parameters into two subsets based on configurable criteria such as amplitude splits and frequency-following behaviour. This is particularly useful for defining separate parameter ranges for idle-state behavior versus perturbation responses or for establishing distinct training and testing scenarios.

Collectively, these utility functions provide mechanisms that are helpful in maintaining framework functionality.

3.5 Experimental Setup and Case Studies

This section presents practical case studies and experimental scenarios that demonstrate the capabilities and reliability of the SigVarGen framework. We focus on replicating real-world power traces captured via the PowerFuzzer setup. These experiments evaluate the generated signals' realism. Each scenario reflects a different signal generation challenge or variation type.

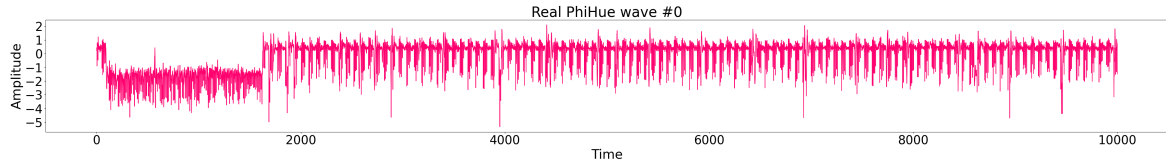


Figure 3.1: Real PhiHue signal example

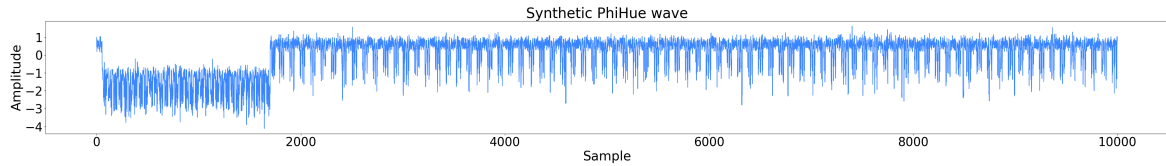


Figure 3.2: Synthetic PhiHue signal example

3.5.1 Baseline Signal Collection

To establish a foundation for evaluating the SigVarGen framework, we collected a comprehensive set of real-world power traces using the PowerFuzzer setup, as detailed in Section 2.3. PowerFuzzer employs a hardware-triggered digitiser to capture power traces from embedded targets in response to a sequence of valid and random test inputs. The PowerFuzzer ensures precise synchronisation between input stimulus and recorded power response with applied voltage amplification.

We used two embedded device platforms as targets for baseline data acquisition: a [Raspberry Pi \(RPI\)](#) device and a Philips Hue bridge. These platforms were selected to represent systems with contrasting runtime behaviours.

For the RPI target, we sent a set of 70 unique commands, comprising 18 valid commands and 52 unknown or potentially malformed instructions. For each command, we recorded 2000 power traces, each of fixed length 100,000 samples. Hardware-level triggering ensured consistent alignment of trace onset with the command execution. Notably, valid commands on the RPI exhibited a characteristic response pattern comprising a double-layered structure: a primary power rise of approximately 9% of the signal length, with a superimposed overlapping response of around 10% within the primary interval. Following this, the signal often contained between 5 and 15 high-frequency burst patterns, lasting approximately 25% of the total trace length. These bursts correspond to rapid fluctuations in power, typically associated with memory or I/O operations. A background system process sporadically introduced short power spikes, resulting in random short-duration deviations across the traces. The average measured amplitude across RPI traces was approximately

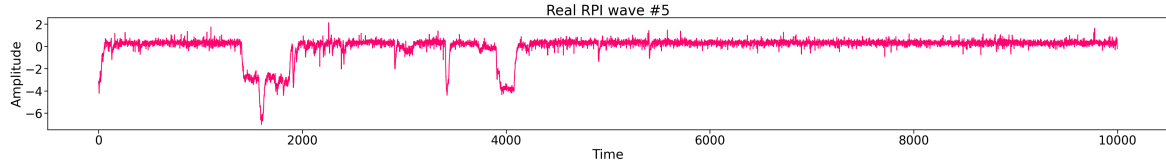


Figure 3.3: Real RPI signal example

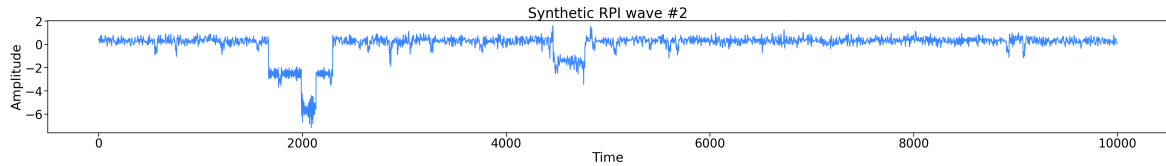


Figure 3.4: Synthetic RPI signal example

−6979 in raw digitiser units.

Similarly, we collected traces corresponding to 80 distinct commands for the Philips Hue device, with 2000 samples per command. The Hue traces exhibit a regular background pattern, with short periodic power increases due to scheduled activities. In contrast to the RPI, the main command execution in the Hue device resulted in a smoother, singular power raise lasting approximately 18% of the trace length, during which periodic background activity was largely suppressed. Occasionally, the target would exhibit short duration and irregular mean power shifts, likely related to asynchronous internal events. The mean amplitude across traces was approximately −3244.

Figures 3.3 and 3.1 illustrate normalised real signals obtained for both devices.

3.5.2 Case Study 1: Signal Synthesis for Embedded Platforms

To evaluate the realism and fidelity of synthetic signal generation, we performed a case study targeting the emulation of power traces typical of embedded platforms, specifically modelled on Philips Hue devices. This case study employed the updated SigVarGen framework v1.1, which was configured with parameters derived through optimisation to reflect the underlying statistical and structural properties observed in Phillips Hue traces.

Synthetic Signal Generation

The synthetic signal generation process was driven by parameter settings controlling key aspects of amplitude variation, burst dynamics, interrupt modelling, and frequency domain structure. Parameter selection was formulated as a data-driven optimisation problem.

We used Optuna with Bayesian optimisation to minimise a distributional discrepancy between real and synthetic features [6]. Specifically, for a parameter vector θ , the objective was the mean 1-Wasserstein distance across d features $J(\theta) = \frac{1}{d} \sum_{j=1}^d W_1(F_j^{\text{real}}, F_j^{\text{synth}}(\theta))$, where F_j^{real} and $F_j^{\text{synth}}(\theta)$ denote the empirical distributions of feature j in the real and generated sets, respectively.

We adopted a two-stage procedure: a broad search over physically plausible bounds from multiple random initialisations to identify promising regions, and a refinement stage that re-centred tighter bounds around the best candidates and reduced the actively optimised subset to the parameters exhibiting the largest influence. The optimal parameters were used to generate the final dataset.

Dataset generation. Specifically, the main amplitude envelope of the base signal for the final dataset was sampled within the range $[1.43, 2.89]$, while interrupt components were injected with amplitudes varying from 0.19 to 2.47. To capture irregular, burst-like fluctuations present in real signals, high-amplitude transient bursts were introduced, with values ranging from -2.08 to 3.2 . Temporal shifts were incorporated using fixed and probabilistic offset constants, ensuring diversity across synthesized traces.

The temporal waveform was generated over a fixed duration of 80 ms at a sampling rate of 1.25 MHz, resulting in signals composed of 100,000 samples. Composite waveforms were constructed from 168 sinusoidal components, distributed across frequency bands ranging from 0 Hz to 183 kHz, to approximate realistic device activity. Additional variation was introduced using interrupt patterns defined by structured binary base patterns with tunable on-off cycles and frequency limits.

The noise was incorporated as additive white noise with a fixed SNR of 25 dB. Each generated signal was stochastically augmented with additional offset regions. The generated waveform was then scaled to match a target variance sampled from the distribution observed in real signals (mean variance $\approx 112,000$, bounded between 58,115 and 167,585), and centred to a target mean drawn from the real dataset’s mean amplitude distribution ($\mu \in [3069.85, 3421.05]$ with $\sigma \approx 63.99$).

100 synthetic traces were generated and subsequently processed using the feature extraction pipeline. The synthetic dataset served as a testbed for evaluating the generative capabilities of SigVarGen. The normalised sample waveform of the generated synthetic trace is depicted in Figure 3.2.

Sensitivity analysis. To understand how individual parameters affect fidelity, we performed a global importance analysis using Optuna’s fANOVA on random trials of the objective, and [one-at-a-time \(OAT\)](#) sweeps for every parameter over expanded local ranges around the baseline. At the baseline, the per-feature discrepancies were dominated by spectral characteristics and impulsiveness (largest Wasserstein gaps at spectral entropy ≈ 2.31 , followed by kurtosis ≈ 1.04 and impulse factor ≈ 1.01), indicating that parameters shaping spectral energy, offsets, and bursts are most consequential.

The fANOVA ranking identified the constant offset inside the interrupt window as most important, followed by [SNR](#), the upper bound of the base amplitude, lower/upper interrupt amplitudes, and the primary base-pattern. Consistently, [OAT](#) sweeps revealed pronounced local sensitivity to interrupt offset and the duration of the interrupt window. Reducing interrupt offset too aggressively degraded the objective (up to +0.40), whereas a modest increase from baseline produced the best local improvement. Extending the interrupt window slightly beyond the baseline end ratio improved the fit, while truncating it early was strongly detrimental. The upper burst amplitude limit exhibited a sharp cliff: a moderate cap improved performance, but allowing overly large bursts worsened it markedly. The pre-offset applied to the periodic interrupts showed a non-monotone effect with a beneficial region near its baseline.

The analysis indicates that matching spectral spread and impulsive behaviour are primarily controlled by the interrupt step height and duration and by constraining burst magnitudes. Secondary adjustments in band placement, window length, and pattern duty cycles further refine the fit. These findings informed the narrowed second-stage search regions and the reduced active set in the final optimisation.

Feature Extraction

We extracted a diverse set of statistical features to enable quantitative comparison between real and synthetic signals that capture each waveform’s structural and spectral characteristics. We selected time-domain and frequency-domain descriptors known to characterise signal dynamics effectively. This comprehensive feature set forms the basis for subsequent evaluation and statistical analysis of signal similarity and variation.

The most important and widely used expected values are the first-order moment, namely the mean value, and the second-order moments, namely the correlation, the covariance, and the power spectrum [115]. Boashash and Ouelha [17] mention mean, variance, skewness, and kurtosis as the base parameters from the time domain and spectral flux, spectral entropy, and spectral flatness from the frequency domain. Jahangir et al. [58] provide an extensive list containing mean, median, peak amplitude, minimum, maximum, standard deviation, variance, harmonic mean, interquartile range, root mean square, peak amplitude, pitch angle, kurtosis, skewness, signal power, coefficient of variation, zero cross rate as features from the time domain, and energy, mean frequency, spectral flux, spectral roll-off, spectral centroid, spectral flatness, spectral entropy and spectral contrast from the frequency domain, without specifying their importance for analysis.

We analysed and adapted features from these key studies, subsequently designing a set of time-domain and frequency-domain features. Time-domain features primarily describe the characteristics of a signal in its original form without transforming it to another domain. These features capture a signal’s inherent patterns, fluctuations, and dynamics as it evolves over time. Frequency-domain features offer insights into a signal’s frequency components, helping to identify the most dominant frequencies and their distribution.

Time-domain Features Below is a conceptual explanation of the time-domain features computed for a 1D signal under analysis. Each feature is widely used in fields like signal processing, vibration analysis, and audio/speech processing to characterise the signal’s statistical or morphological properties in the time domain. Time features were extracted using the NumPy Python library [51]. The following functions use common nominators such as $x[n]$ representing the signal, N denoting the number of samples, μ is the mean, and σ is the standard deviation.

The *mean* value of a signal represents its central tendency [17]. A nonzero mean indicates the presence of a **Direct Current (DC)** component, also known as **DC** bias or **DC** offset. The mean is computed through equation 3.3. While mean was mentioned as one of the most crucial time-domain features in multiple studies, it is worth noting that in audio and vibration signals, a properly centered signal typically has a mean near zero, reflecting an **Alternating Current (AC)** phenomenon. And thus, the mean value often cannot define the average amplitude. Instead, the mean-square value is computed to measure the *signal’s average strength*.

$$\text{mean } \mu = \frac{1}{N} \sum_{n=0}^{N-1} x[n]. \quad (3.3)$$

The *variance* of a signal measures the degree to which its amplitude values deviate from the mean, providing insight into the signal's dispersion or spread [17]. A higher variance indicates more significant variability in the signal, often corresponding to signals with pronounced fluctuations in amplitude. In contrast, a lower variance suggests that the signal's amplitude values are tightly clustered around the mean. Variance is commonly used in signal analysis to quantify the overall energy distribution around the central tendency and is computed using Equation 3.4.

$$\text{variance } \sigma^2 = \frac{1}{N} \sum_{n=0}^{N-1} (x[n] - \mu)^2, \quad (3.4)$$

where μ is the mean of the signal.

The *standard deviation*, derived as the square root of the variance $\text{std_dev } \sigma = \sqrt{\text{variance}}$, serves as another important metric for analysing signal variability. It directly interprets the average deviation of the signal's amplitude from its mean. A large standard deviation signifies substantial amplitude variations, while a small standard deviation indicates consistency in amplitude values around the mean.

The *median* is the value separating the higher half and the lower half of the sorted sample values [58]. Unlike the mean, the median is more robust to outliers. If a signal has large spikes or outliers, its mean can be heavily affected, while the median remains more stable. The median is calculated using equation 3.5, where the position of the median is determined based on whether the number of samples N is odd or even.

$$\text{median} = \begin{cases} x\left(\frac{N+1}{2}\right), & \text{if } N \text{ is odd,} \\ \frac{x\left(\frac{N}{2}\right) + x\left(\frac{N}{2}+1\right)}{2}, & \text{if } N \text{ is even.} \end{cases} \quad (3.5)$$

Peak-to-peak amplitude measures the total amplitude range of the signal [58] and is commonly used in engineering to quickly assess the maximum excursion from the lowest to the highest point of the signal. Peak-to-peak amplitude is calculated through Equation 3.6.

$$\text{peak_to_peak} = \max_n \{x[n]\} - \min_n \{x[n]\}. \quad (3.6)$$

Root Mean Square (RMS) is a standard measure of the signal's power [58]. In electrical engineering, *RMS* of a voltage or current signal equates to its *DC*-equivalent for power calculations. RMS is calculated through Equation 3.7.

$$\text{rms} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} (x[n])^2}. \quad (3.7)$$

Crest factor indicates how extreme the peaks are in relation to the overall power of the signal, and is calculated as $\frac{\text{peak_to_peak}}{\text{rms}}$. A large crest factor implies the presence of strong transients or spikes. A lower crest factor means a more consistent amplitude. *Impulse factor*, similar to the crest factor, assesses the transient nature of the signal by comparing the maximum absolute amplitude (peak) to the average absolute amplitude μ_{abs} . It is computed as $\frac{\max |x[n]|}{\mu_{\text{abs}}}$. A large impulse factor suggests that the signal contains short, high-intensity bursts significantly exceeding the ongoing average level. *Shape factor* compares the RMS level to the average absolute level. It can be interpreted as how "peaky" a signal is relative to its absolute value version. It is calculated by dividing rms by μ_{abs} , where μ_{abs} is a mean of $|x[n]|$.

The [Zero-Crossing Rate \(ZCR\)](#) quantifies the rate at which a signal's amplitude changes sign. A high [ZCR](#) is typically associated with high-frequency or noisy signals, where numerous sign changes occur per unit time. Conversely, a low [ZCR](#) indicates smoother oscillations or low-frequency components, reflecting fewer sign changes. [ZCR](#) is computed using Equation 3.8.

$$\text{ZCR} = \frac{1}{N} \sum_{n=1}^{N-1} \mathcal{I}(\text{sign}(x[n]) \neq \text{sign}(x[n-1])), \quad (3.8)$$

where $\mathcal{I}(\cdot)$ is an indicator function that evaluates to 1 if the condition inside is true and 0 otherwise.

Skewness of a signal quantifies the asymmetry of its amplitude distribution around the mean, providing a measure of how the distribution deviates from symmetry [17]. Positive skewness indicates the presence of a longer or heavier tail on the right side of the distribution, often caused by a few large positive values. Conversely, negative skewness signifies a longer or heavier tail on the left, typically due to a few large negative values. Skewness is computed using Equation 3.9.

$$\text{skewness} = \frac{\frac{1}{N} \sum_{n=0}^{N-1} (x[n] - \mu)^3}{\sigma^3}. \quad (3.9)$$

Kurtosis of a signal measures the "tailedness" or "peakedness" of its amplitude distribution, offering insights into the signal's propensity for extreme values [17]. A positive

excess kurtosis indicates a sharp peak and heavy tails compared to a Gaussian distribution, often caused by impulsive events. On the other hand, a negative excess kurtosis suggests a flatter distribution with lighter tails. Kurtosis is calculated using Equation 3.10.

$$\text{kurtosis} = \frac{\frac{1}{N} \sum_{n=0}^{N-1} (x[n] - \mu)^4}{\sigma^4}, \quad (3.10)$$

The time-domain features provide a comprehensive analysis of a signal's amplitude distribution, variability, shape, and temporal structure, encompassing various categories such as measures of central tendency (e.g., mean, median), measures of dispersion (e.g., variance, standard deviation, peak-to-peak, RMS), rate of sign changes (e.g., zero-crossing rate), form or shape indicators (e.g., crest factor, shape factor, impulse factor), and statistical moments (e.g., skewness, kurtosis). Together, these properties provide complimentary insights into the signal's energy level, dispersion, dynamic range, impulsiveness, asymmetry, and temporal correlation.

Frequency-domain Features The frequency-domain representation of a signal provides critical insights into its spectral characteristics. Frequency features were extracted using the NumPy Python library. Fourier analysis is fundamentally a method for expressing a function as a sum of periodic components and recovering the function from those components. When the function and its Fourier transform are replaced with discretised counterparts, it is called the [Discrete Fourier transform \(DFT\)](#). Applying the [DFT](#) via the [Fast Fourier Transform \(FFT\)](#) generates a complex spectrum that describes how different frequency components contribute to the overall signal. [FFT](#) refers to a way the [DFT](#) can be calculated efficiently by using symmetries in the calculated terms [51, 25]. DFT is defined by Equation 3.11.

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left\{-2\pi i \frac{k n}{N}\right\}, \quad k = 0, 1, \dots, N-1, \quad (3.11)$$

where $x[n]$ is the discrete-time signal in the time domain, $X[k]$ is the [DFT](#) of $x[n]$ in the frequency domain, n is the time-domain index, ranging from 0 to $N-1$, k is the frequency-bin index in the DFT, ranging from 0 to $N-1$, N is the total number of samples in one analysis frame, i is the imaginary unit, where $i^2 = -1$.

Given an input $\mathbf{x}$, representing the signal in the time domain, one of the first steps is to compute its magnitude spectrum. The magnitude spectrum measures the signal's

amplitude at various frequencies and is obtained using the [FFT](#). This process is formally defined by Equation [3.12](#).

$$\mathbf{X} = \text{FFT}(\mathbf{x}), \quad \text{magnitude_spectrum} = |\mathbf{X}|, \quad (3.12)$$

where $\mathbf{X}$ denotes the Fourier transform of the time-domain signal $\mathbf{x}$, and $|\mathbf{X}|$ represents the absolute value (or magnitude) of the Fourier transform. The FFT algorithm efficiently computes the Fourier transform, which decomposes the signal into a sum of sinusoidal components with specific frequencies and amplitudes. The resulting magnitude spectrum $|\mathbf{X}|$ reflects the relative strength (amplitude) of these frequency components.

The Fourier transform operates in the frequency domain, where each component corresponds to a discrete frequency. These frequencies are defined as $\mathbf{f} = [f_0, f_1, \dots, f_{N-1}]$, where $\mathbf{f}$ is a vector containing the discrete frequencies of the Fourier transform, and N is the length of the input signal $\mathbf{x}$. Here, f_0 corresponds to the [DC](#) component, while subsequent values f_k represent progressively higher frequencies. This definition assumes zero-based indexing for simplicity.

The magnitude spectrum is expressed as $\mathbf{M} = \{M_0, M_1, \dots, M_{N-1}\}$, where $M_k = |\mathbf{X}_k|$ is the magnitude of the k -th frequency component. Together, $\mathbf{M}$ (the magnitude spectrum) and $\mathbf{f}$ (the discrete frequency set) form the basis for computing spectral features such as spectral centroid, spectral spread, spectral skewness, spectral kurtosis, spectral flatness, etc.

The spectral centroid represents the “centre of mass” of the magnitude spectrum [\[46, 58\]](#) and is formally given by Equation [3.13](#). The centroid indicates where the centre of the spectral energy is located. In audio processing, for instance, a higher spectral centroid suggests a brighter or higher-pitched timbre. Mathematically analogous to a weighted mean, it is sensitive to how energy in the frequency spectrum is distributed.

$$\text{spectral_centroid } \mu_{\mathbf{f}} = \frac{\sum_{k=0}^{N-1} f_k M_k}{\sum_{k=0}^{N-1} M_k} \quad (3.13)$$

The spectral spread quantifies the dispersion of the spectral energy around the spectral centroid [\[46\]](#) and is defined by Equation [3.14](#). Spectral spread measures how “wide” or “narrow” the main energy distribution is in the frequency spectrum. Signals with energy localised around a single narrow frequency band will exhibit a small spread; signals with energy spread over many frequencies will have a larger spread.

$$\text{spectral_spread} = \sqrt{\frac{\sum_{k=0}^{N-1} (f_k - \mu_f)^2 M_k}{\sum_{k=0}^{N-1} M_k}}. \quad (3.14)$$

Spectral skewness measures the asymmetry of the spectral distribution around its mean (the spectral centroid) [17, 58]. Positive skewness indicates a distribution that has a tail extending towards higher frequencies. Negative skewness indicates a tail extending towards lower frequencies. It highlights whether a signal's spectral energy is skewed toward higher or lower frequency ranges relative to the centroid. Spectral skewness is computed using Equation 3.15.

$$\text{spectral_skewness} = \frac{\sum_{k=0}^{N-1} (f_k - \mu_f)^3 M_k}{\sum_{k=0}^{N-1} (M_k)^3}. \quad (3.15)$$

Spectral kurtosis quantifies the sharpness and concentration of the spectral distribution [118]. A positive kurtosis indicates a distribution with sharp peaks and heavy tails, suggesting energy concentrated in a few strong frequency components or impulsive events. Conversely, a lower kurtosis indicates a flatter, more uniform distribution, implying the energy is spread across a broader range of frequencies. High spectral kurtosis is often associated with signals containing transient or narrowband components. Spectral kurtosis is computed using Equation 3.16.

$$\text{spectral_kurtosis} = \frac{\sum_{k=0}^{N-1} (f_k - \mu_f)^4 M_k}{\sum_{k=0}^{N-1} (M_k)^4}. \quad (3.16)$$

The *spectral entropy* quantifies the randomness or flatness of the energy distribution across the frequency spectrum, offering a measure of how evenly energy is spread across frequency bins [117]. A low spectral entropy value signifies that energy is concentrated in a small number of frequency bins, indicating a tonal or peaked signal. Conversely, a high spectral entropy reflects a more uniform and broadband energy distribution, often associated with noise-like signals. Spectral entropy is computed using Equation 3.17.

$$\text{spectral_entropy} = - \sum_{k=0}^{N-1} p_k \log_2(p_k + \epsilon), \quad (3.17)$$

where $p_k = \frac{M_k}{\sum_{j=0}^{N-1} M_j}$ is the normalized magnitude of the spectrum, $\epsilon \approx 2.220446 \times 10^{-16}$ is a small constant for numerical stability.

The *peak frequency* identifies the frequency bin with the maximum magnitude, offering insight into the dominant frequency component of a signal [120]. Peak frequency is determined using Equation 3.18.

$$\text{peak_frequency} = f_{k^*}, \quad \text{where} \quad k^* = \arg \max_k M_k, \quad (3.18)$$

The *spectral flatness* measures the uniformity of the magnitude spectrum, offering a way to distinguish noise-like signals from tonal or harmonic ones [17, 58]. A spectral flatness close to 1 indicates a uniform energy distribution across the frequency spectrum, as observed in white noise. Conversely, a value near 0 suggests that the signal contains dominant peaks, indicating tonal characteristics. Spectral flatness is computed using Equation 3.19.

$$\text{spectral_flatness} = \frac{\exp\left(\frac{1}{N} \sum_{k=0}^{N-1} \ln(M_k + \delta)\right)}{\frac{1}{N} \sum_{k=0}^{N-1} (M_k + \delta)}, \quad (3.19)$$

where δ is a small constant (e.g., 10^{-12}) to avoid logarithms of zero.

These features collectively capture various characteristics of how the energy in the signal is distributed across frequencies – ranging from its central tendency (centroid) and dispersion (spread) to its shape (skewness, kurtosis), concentration (peak frequency), uniformity (entropy), and harmonic vs. noise-like nature (flatness).

Statistical Testing and Comparison Methodology

To rigorously evaluate the realism of synthetic signals generated using SigVarGen, we employed two complementary statistical evaluation methods: the Mann–Whitney U test and the Wasserstein distance metric. These approaches serve distinct but complementary purposes. The Mann–Whitney U test tells whether the distributions are statistically similar, while the Wasserstein distance shows how different they are in terms of shape and spread.

We utilised the non-parametric Mann–Whitney U test to assess whether the statistical properties of synthetic signals significantly differed from those of real Philips Hue device traces. The Mann–Whitney test, also known as the Wilcoxon rank-sum test, is particularly suited to our analysis because it does not require assumptions of normality, making it robust for feature distributions typical of power traces [69]. Formally, the null hypothesis for the Mann–Whitney U test applied to each feature is defined in Equation 3.20.

$$H_0 : F_{\text{real}}(x) = F_{\text{synth}}(x) \quad (3.20)$$

This hypothesis states that the feature values from the real and synthetic datasets are drawn from the same underlying distribution. The alternative hypothesis, which suggests a significant distributional difference, is defined in Equation 3.21.

$$H_1 : F_{\text{real}}(x) \neq F_{\text{synth}}(x) \quad (3.21)$$

A significance threshold of $\alpha = 0.05$ was adopted for all tests. If the resulting p -value falls below this threshold, the null hypothesis is rejected, indicating a statistically significant deviation between the synthetic and real distributions for that feature.

While the Mann–Whitney U test offers inferential clarity regarding statistical equivalence, it does not quantify the magnitude of distributional discrepancies. To address this limitation, we supplemented our analysis with the Wasserstein distance, which explicitly measures the distance between two probability distributions in terms of the minimal amount of “mass” that must be moved to transform one distribution into the other [82].

We employed Wasserstein distance to conduct three distributional comparisons: synthetic versus real Philips Hue traces, real RPI versus real Philips Hue traces, and synthetic Philips Hue versus real RPI traces. These comparisons enabled a comprehensive assessment of synthetic data realism, contextualised within real-world inter-device variability.

Formally, for each feature $f \in \mathcal{F}$, where $\mathcal{F}$ denotes the set of all extracted features, the Wasserstein distance $W(f; A, B)$ was computed between datasets A and B . We defined a working hypothesis that the synthetic Philips Hue dataset should be distributionally closer to the real Philips Hue dataset than to the real RPI dataset, and that this distance should also be smaller than the distributional distance between the real RPI and real Philips Hue datasets. This condition was evaluated per feature using the Equation 3.22.

$$\forall f \in \mathcal{F}, \quad W(f; S, H) < \min [W(f; R, H), W(f; S, R)] \quad (3.22)$$

where S , H , and R represent the synthetic Philips Hue, real Philips Hue, and real RPI datasets, respectively.

Evaluation under this comparative framework enables us to determine whether the distributional differences between synthetic and real signals are within the bounds of natural inter-device variability. A violation of the working hypothesis defined in Equation 3.22 occurs when the Wasserstein distance between synthetic and real Philips Hue data exceeds

the distance observed between two real devices. Such violations indicate that the synthetic signals deviate from their intended target more than real-world hardware variability would suggest, thereby highlighting specific features where the signal generation process may require further refinement.

Combining the inferential robustness of the Mann–Whitney U test with the descriptive sensitivity of the Wasserstein distance, our evaluation framework provides a comprehensive and interpretable foundation for assessing the realism of synthetic signal generation.

3.5.3 Case Study 2: Synthetic Signal Augmentation

To assess the effectiveness of SigVarGen v0.1 in reproducing realistic variability across repeated measurements, we designed a controlled study simulating multiple observations of a consistent underlying power event. In practical measurement scenarios, even identically configured experiments yield variations arising from physical, electronic, and environmental influences such as sensor noise, temporal sampling discrepancies, gain fluctuations, and non-stationary interference. This case study specifically aimed to validate whether SigVarGen-generated synthetic signals authentically represent such real-world measurement variability.

Synthetic Signal Variations Generation

To explore the effects of domain-specific variation, this study focuses on signal augmentation for two distinct embedded domains: Arduino boards and camera systems. Table 3.2 summarises the expected signal characteristics for these device types. Each base waveform was augmented to produce sixteen distinct variants, resulting in a controlled set of synthetic observations that emulate repeated measurements.

All variants were generated with a fixed signal length of 10,000 samples and modulated through combinations of structured transformations and controlled noise additions. Each synthetic variation was created using the augmentation pipeline outlined in Section 3.3.3, comprising amplitude modulations, time-domain transformations, and local baseline drifts. The variation parameters were selected from narrow sub-ranges within the broader configuration space using algorithm 4 to ensure that while each variant differed measurably, all preserved the semantic structure of the original waveform.

Variants for each domain were generated using tailored noise power configurations, with the Arduino dataset exhibiting a SNR range from -2.92 to 6.05 dB, while camera-generated waveforms spanned a broader SNR spectrum from -2.78 to 15.26 dB. The noise

colour was randomly sampled from the white, pink, or brown options for each base wave. The generated variations leveraged either stationary or non-stationary noise models. Non-stationary noise was modulated via random time-varying amplitude envelopes, effectively simulating power fluctuations and environmental noise bursts.

Feature Extraction

This subsection provides a methodological and conceptual overview of quantitative features extracted from synthetic signal variations. The selected features capture the statistical, temporal, and spectral properties essential for authentic representation and systematic characterisation of realistic measurement variability. Such measures are widely employed in signal processing, biomedical analyses, and condition monitoring applications to analyse variations between sequential observations of nominally identical events.

Before extraction, signals were uniformly resampled to 10,000 samples via linear interpolation, followed by normalisation, subtracting the mean and dividing by the standard deviation. This ensures consistent scale and sample length, eliminating biases due to amplitude or duration discrepancies.

To quantify shape and pattern differences between two sequential signals, we computed the [Euclidean Distance \(ED\)](#). ED measures the global deviation in amplitude and temporal profile between two normalised signals $x[n]$ and $y[n]$, defined mathematically by Equation 3.23.

$$\text{ED} = \sqrt{\sum_{n=0}^{N-1} (x[n] - y[n])^2}, \quad (3.23)$$

where N denotes the total number of samples. ED is well-established in literature for comparing signal profiles, with extensive applications ranging from speech recognition [30] to neural pattern detection [33].

Additionally, we defined a complementary measure termed CP, designed to quantify the relative power or amplitude-based variation between consecutive signals. The CP metric calculates the power ratio between two signals, assessing their comparative strength and offering insight into their relative intensities. Mathematically, CP was defined based on the SNR computation as shown in Equation 3.24.

$$\text{CP (dB)} = 10 \log_{10} \left(\frac{\sum_{n=0}^{N-1} x[n]^2}{\sum_{n=0}^{N-1} (x[n] - y[n])^2} \right). \quad (3.24)$$

CP provides a distinct perspective compared to traditional time-series similarity metrics. While distance-based metrics such as ED emphasise structural similarity, capturing shape, timing, or pattern, CP specifically characterise the intensity differences, often related to amplitude and noise variations. This is particularly valuable in contexts where variations in system behaviour can be discerned purely based on signal strength, potentially making deeper structural analysis redundant. Thus, CP correlates with established distance metrics and supplements them by quantifying amplitude-based variation and revealing underlying characteristics of system noise.

Statistical metrics describing the distributional characteristics of the resulting ED and CP sequences (mean, standard deviation, skewness, and kurtosis) were calculated according to formulas previously defined in Section 3.5.2. These metrics encapsulate the central tendency, dispersion, asymmetry, and peakedness, providing comprehensive statistical descriptors of signal variability.

Temporal dependencies were analyzed using autocorrelation at lag 1, as expressed by Equation 3.25. The autocorrelation at lag 1 quantifies the similarity between adjacent samples in a signal, providing a measure of its temporal dependency or smoothness [115].

$$\text{autocorrelation_lag_1} = \frac{\sum_{n=0}^{N-2} x[n]x[n+1]}{N-1} \quad (3.25)$$

Spectral characteristics were assessed using PSD, computed through Welch’s method, an established spectral estimation technique. Welch’s method divides the original signal into overlapping segments, applies windowing functions to reduce spectral leakage, computes the periodogram of each segment, and averages these periodograms to enhance spectral estimate stability [119]. Mathematically, Welch’s PSD is computed through Equation 3.26).

$$P_{xx}(f) = \frac{1}{K} \sum_{k=1}^K \left| \frac{1}{M} \sum_{n=0}^{M-1} x_k[n]w[n]e^{-j2\pi fn} \right|^2, \quad (3.26)$$

where $x_k[n]$ denotes the k -th segment of the signal, $w[n]$ is the window function, M is the segment length, and K is the number of segments [117]. PSD analysis via Welch’s method has proven effective across various fields, including biomedical signal processing and vibration analysis, due to its robust and stable spectral estimation capability [106].

Furthermore, to analyse rapid temporal dynamics, first-order differences of both ED and CP sequences were computed, followed by calculation of statistical features such as

mean, standard deviation, skewness, and kurtosis. This incremental analysis highlights transient, short-term variations between consecutive measurements, a methodology widely adopted in EEG-based studies and real-time signal characterisation tasks [59, 31].

In summary, the comprehensive suite of extracted features, incorporating ED and CP with temporal dependencies, spectral density characteristics, and incremental statistics, provides robust validation of SigVarGen’s capacity to generate realistic synthetic signal variations. These features enable precise quantification and comparison with empirical variability observed in authentic measurement data.

Testing and Comparison Methodology

A suite of analytical techniques and visualisation approaches was employed to rigorously validate the realism and appropriateness of the synthetic signal variations generated by SigVarGen. These methods facilitated comparisons between synthetic signals and real measurements from Section 3.5.1 expanded with HikCam and GSM module data, providing qualitative insights into the statistical and structural fidelity of the generated data.

The dimensionality and internal structure of the extracted feature sets were initially explored using **Principal Component Analysis (PCA)**, implemented via the *scikit-learn* library [84]. PCA is a widely utilised technique for dimensionality reduction, projecting high-dimensional data onto a lower-dimensional subspace that captures the maximum variance present in the original dataset. Formally, PCA involves finding a set of orthonormal vectors, known as principal components, along which the variance of the projected data is maximised [111]. Given a dataset consisting of N samples of d -dimensional vectors $\{\mathbf{t}_n\}_{n=1}^N$, the principal axes are obtained through an eigendecomposition of the data covariance matrix $\mathbf{S}$ as given by Equation 3.27.

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^N (\mathbf{t}_n - \bar{\mathbf{t}})(\mathbf{t}_n - \bar{\mathbf{t}})^T, \quad (3.27)$$

where $\bar{\mathbf{t}}$ is the mean vector. The principal components correspond to the eigenvectors associated with the largest eigenvalues λ_j , satisfying the eigenvalue Equation 3.28.

$$\mathbf{S}\mathbf{w}_j = \lambda_j\mathbf{w}_j, \quad (3.28)$$

where $\mathbf{w}_j$ represents the j th principal axis. A lower-dimensional representation $\mathbf{x}_n$ of the original data vector $\mathbf{t}_n$ is then computed by projecting the mean-centered data onto the principal axes through Equation 3.29.

$$\mathbf{x}_n = \mathbf{W}^T(\mathbf{t}_n - \bar{\mathbf{t}}), \quad (3.29)$$

with $\mathbf{W}$ being the matrix whose columns are the principal eigenvectors $\mathbf{w}_j$.

PCA effectively reveals inherent groupings or overlaps between real and synthetic signals, thus offering intuitive insights into how well synthetic data replicate the distribution and structure of empirical measurements. The efficacy of PCA in validating synthetic data distributions is extensively documented in various studies, including EEG analysis for validating extracted components [26].

Complementing the PCA analysis, detailed visual comparisons were performed by plotting pairs of extracted statistical features based on [ED](#) and [CP](#). Scatter plots were constructed to visualise the relationships and overlaps between synthetic and real datasets. In these visualisations, synthetic data points were colour-coded according to their mean [SNR](#), allowing immediate recognition of how variations in [SNR](#) influence feature distributions. Real measurements from embedded systems, initially described in Section [3.5.1](#) and complemented by additional datasets from HikCam and GSM module collected through a similar pipeline, were plotted alongside synthetic signals. Distinct markers were assigned to each dataset to enhance visual clarity and facilitate intuitive comparative interpretation. Scatter plot visualisations are critical tools for pattern identification and qualitative assessment of synthetic data realism, as emphasised in research focused on effective data visualisation methodologies [27].

Further statistical realism validation was conducted through distributional analyses, directly comparing feature distributions between synthetic signals and real measurements. Density plots and histograms were generated for each extracted feature, providing visual assessments across synthetic Arduino-board-generated signals, synthetic camera-based signals, and multiple real-device datasets. These analyses employed [Kernel Density Estimation \(KDE\)](#), a non-parametric method for estimating probability density functions from empirical data. Mathematically, [KDE](#) is expressed by Equation [3.30](#).

$$\hat{f}(x) = \frac{1}{Nh} \sum_{i=1}^N K\left(\frac{x - x_i}{h}\right), \quad (3.30)$$

where $\hat{f}(x)$ is the estimated density at point x , N is the number of data points, h is the bandwidth parameter that controls the smoothness of the density estimate, and $K(\cdot)$ denotes the kernel function, typically Gaussian [3.31](#).

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}. \quad (3.31)$$

These KDE-based distribution plots allowed detailed comparisons of central tendency, dispersion, and overall distribution shape, demonstrating whether synthetic signals exhibited statistical properties consistent with real-world measurements across various embedded domains. KDE’s effectiveness in evaluating synthetic data realism has been extensively studied, notably within privacy-preserving synthetic data research, underscoring its robust capacity for distribution similarity assessments [54].

Collectively, this integration of PCA-based dimensionality exploration, scatter-plot visualisation, and KDE-based distributional comparisons provides a robust multidimensional validation framework. Chosen analytical techniques ensure that synthetic signal variations produced by SigVarGen are rigorously validated, closely matching real-world measurement characteristics.

3.6 Experimental Results

This section presents comprehensive experimental results to evaluate the realism, statistical fidelity, and effectiveness of the synthetic signals generated by the SigVarGen framework. Two distinct case studies are reported. First case study assesses SigVarGen’s ability to synthesise realistic signal traces for embedded platforms, and the second examines its capability to replicate realistic variations occurring between multiple measurements of identical events. Each case study employs statistical and visualisation-based validation methods, establishing the robustness and applicability of the proposed synthetic signal generation approach.

3.6.1 Case Study 1: Signal Synthesis for Embedded Platforms

To evaluate the effectiveness and realism of synthetic signal generation framework, we conducted comprehensive statistical analyses comparing the generated synthetic Philips Hue signals with their real-world counterparts and with signals from a distinct embedded platform.

The Mann-Whitney U test assessed whether the distributions of individual signal features from synthetic and real Philips Hue datasets differed significantly. Table ?? presents each extracted feature’s statistics and corresponding p-values.

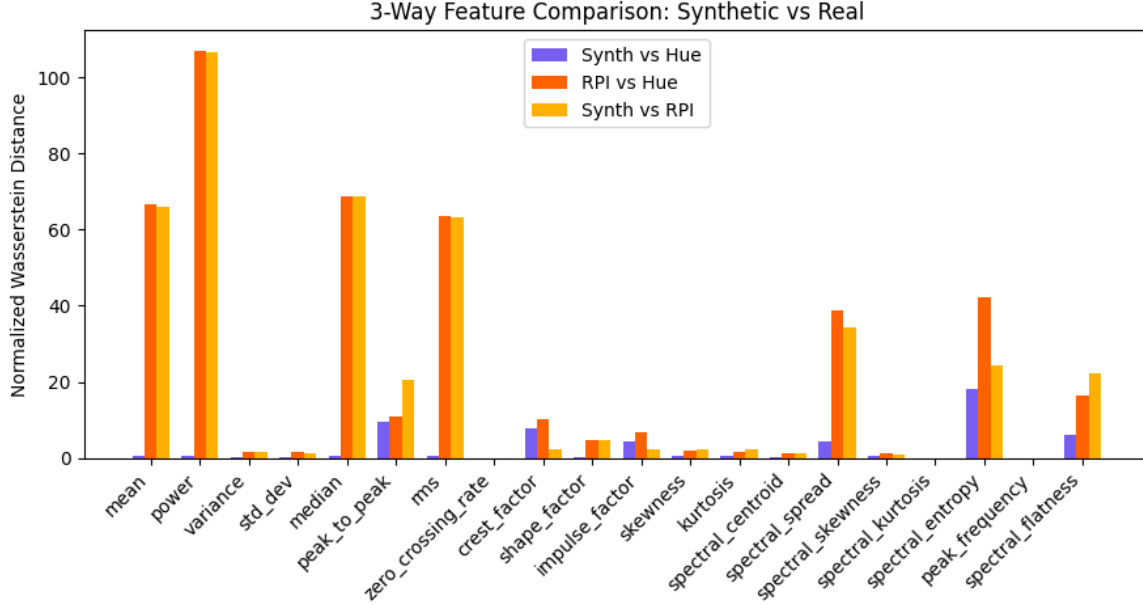


Figure 3.5: Feature-wise Wasserstein distances between synthetic and real Philips Hue, real RPI and real Philips Hue, and synthetic Philips Hue versus real RPI datasets

The results demonstrate that the majority of features (13 out of 20) exhibit no statistically significant difference between synthetic and real signals ($p > 0.05$). However, several critical waveform features showed statistically substantial discrepancies. These findings highlight that while general amplitude-based and spectral features are closely replicated, certain temporal and spectral irregularities require further refinement in synthetic generation.

To complement the hypothesis tests, we computed Wasserstein distances for feature-wise comparisons between synthetic versus real Philips Hue signals, real RPI versus real Philips Hue signals, and synthetic versus real RPI signals. Figure 3.5 visually summarises these distances for each feature.

The Wasserstein distance results demonstrated strong agreement with the Mann–Whitney U test findings. Notably, synthetic signals consistently showed the smallest Wasserstein distances when compared with their real Philips Hue counterparts across all evaluated valid features (17 out of 17, excluding features with insufficient variability). Features reflecting primary statistical properties (mean, median, variance, standard deviation) and fundamental spectral attributes (spectral centroid, spectral skewness) exhibited minimal synthetic-to-real distances, highlighting the effectiveness of the SigVarGen framework in

capturing general statistical behaviour.

Certain waveform shape features, such as crest factor, impulse factor, and peak-to-peak amplitude, alongside spectral complexity indicators like spectral entropy, spectral flatness, and spectral spread, displayed comparatively higher distances. While synthetic data still showed a closer resemblance to the real Philips Hue signals than to the real RPI dataset, the elevated Wasserstein values for these particular features suggest residual gaps in accurately capturing transient peaks and spectral irregularities.

Overall, our results confirm that the developed signal generation framework SigVarGen can produce synthetic datasets that closely emulate key statistical and spectral characteristics of real devices, with minor feature exceptions.

3.6.2 Case Study 2: Synthetic Signal Augmentation

To evaluate the realism and effectiveness of synthetic signal variations generated by SigVarGen, we conducted a comparative analysis against real measurement variations obtained from Raspberry Pi. The [PCA](#), a widely accepted dimensionality reduction method described previously in Section 3.5.3, was applied to visualise and compare the distribution and internal structure of the extracted statistical features.

Figure 3.6 illustrates the two-dimensional [PCA](#) projection, showing distributions of synthetic signal variations in blue and real [RPI](#)-based measurements in red. Visual examination indicates significant overlap between the two groups, suggesting that synthetic signals closely emulate the real-world signal variability measured by the PowerFuzzer framework for the [RPI](#) system. Nonetheless, subtle differences are noticeable: specific real measurements exhibit distinct extremes along the second principal component, reaching values below -60 , which the synthetic distribution does not cover extensively.

Following [PCA](#)-based dimensionality reduction, detailed distributional analyses were conducted to assess the statistical realism of synthetic signal variations. This was achieved by comparing [KDE](#) distributions of multiple statistical and spectral features derived from synthetic and real signals. Figures in Appendix A depict distributions for selected [CP](#) and [ED](#) based features.

Mean values of [ED](#) distributions in Figure A.1) exhibit two distinct peaks for GSM and HikCam datasets, reflecting characteristic signal behaviours likely related to the unique measurement scenarios. Synthetic signals generally capture one prominent peak region, suggesting room for improvement in covering more extensive feature variability observed

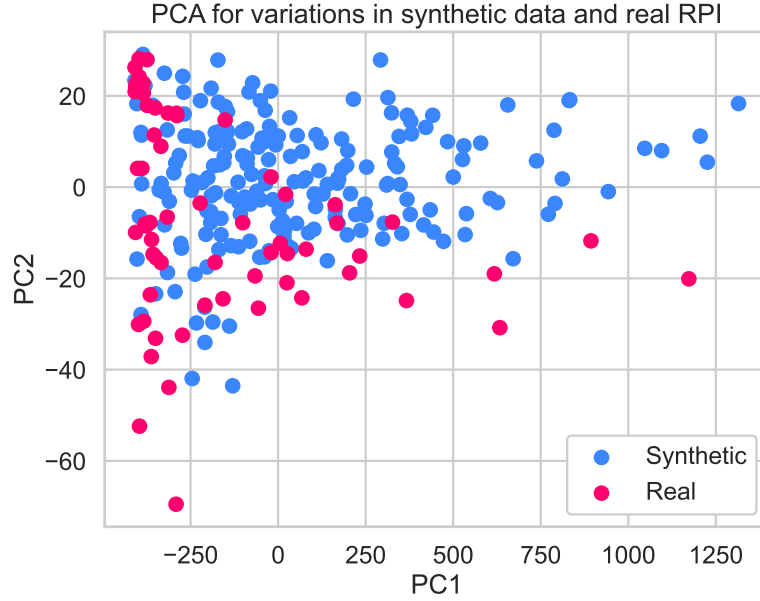


Figure 3.6: PCA visualisation comparing feature distributions between synthetic variations (blue) and real RPI measurements (red).

in practical deployments. Figure A.2 displays the standard deviation distribution. Similarly, GSM and HikCam signals present additional peaks, suggesting nuanced behaviours currently beyond synthetic emulation capability.

Skewness analyses in Figure A.3) indicates a close alignment between synthetic and real data distributions. The synthetic distributions closely mirror real-world measurements' shape, central tendency, and variability, with minor deviations observed in GSM distribution.

Figure A.4 shows the distribution of kurtosis calculated on ED. It can be observed that synthetic signals generated for both Arduino and camera-based configurations closely match the shape, location, and spread observed in real measurements from various devices.

Autocorrelation distributions at lag 1 in Figure A.5 reveal substantial similarity between synthetic and real signals. Synthetic Arduino and camera signals effectively capture the central tendency and variability of autocorrelation observed in real datasets. The GSM dataset shows an extended range, highlighting specific temporal characteristics not fully represented in the synthetic set.

Further validation of SigVarGen-generated synthetic signals was achieved through de-

tailed pairwise analyses of statistical and spectral feature pairs. Scatter plots, presented in Appendix B, illustrate the relationships between various ED and CP derived features for synthetic signals and real measurements.

Analysis of higher-order statistical features, kurtosis and skewness, is presented in Figure B.3. Real and synthetic datasets align closely along clear linear trends, confirming that synthetic signals accurately replicate complex statistical patterns in actual embedded-device measurements. Sparse discrepancies at higher kurtosis values imply occasional extreme deviations in the synthetic dataset, likely related to specific augmentation modelling parameters.

Figure B.2 compares fundamental statistical features calculated from ED and CP sequences, such as mean and standard deviation. These features also display clear linear correlations, highlighting consistency between amplitude-based and shape-based metrics across synthetic and real measurements. The extensive overlap reinforces the effectiveness of synthetic signals in realistically replicating typical measurement variability.

The relationships between incremental feature pairs are displayed in Figure B.1. Synthetic and real datasets demonstrate strongly linear relationships between delta ED and delta CP features across different statistical measures. The synthetic signals comprehensively span the same regions as real measurements, replicating the realistic correlation structure observed in empirical data. Minor outliers or deviations in the delta kurtosis plot suggest instances where synthetic signals exhibit a broader variability range compared to specific real-world conditions.

Finally, Figure B.4 assesses the spectral (PSD mean) and temporal dependency (autocorrelation lag-1) features. Both synthetic and real data exhibit strong linear trends and nearly perfect overlap in autocorrelation values, validating the temporal structure of synthetic variability. The PSD mean scatter plot indicates broader dispersion in synthetic data, particularly at higher PSD values, suggesting synthetic signals may benefit from further spectral refinement.

In conclusion, the results presented in this case study demonstrate that SigVarGen v0.1 effectively reproduces realistic variability observed across repeated measurements of the same underlying power event.

3.7 Results and Conclusion

SigVarGen is a Python-based framework for synthesising realistic one-dimensional signals, including idle-state and active signals representing stable, repeatable events captured

across multiple measurements under consistent environmental conditions. The framework also provides extensive tools for introducing controlled variations and augmentation techniques, including amplitude modulation, baseline drift, time shifting, and warping, as well as noise addition (stationary/non-stationary, white/coloured), effectively simulating real-world measurement variability. Through rigorous experimental evaluation presented in earlier chapters, SigVarGen demonstrated the capability to accurately replicate typical measurement scenarios and realistically emulate variability observed between different signal measurements of the same event. These capabilities position SigVarGen as a robust and versatile synthetic data generation solution suitable for sensor data modelling, embedded systems benchmarking, signal processing validation, and machine learning dataset augmentation.

The developed framework provides a structured, modular signal generation and augmentation approach. Key capabilities include synthesising complex signals by summing multiple sinusoidal components, generating interrupting signals to simulate real-world perturbations, and systematically varying signal parameters using amplitude modulation, baseline drift, time shifting, and warping transformations. Furthermore, SigVarGen’s noise generation module offers extensive control over noise characteristics, supporting white and coloured noise models with stationary and non-stationary noise patterns. This comprehensive feature set positions SigVarGen as a versatile tool for signal processing, embedded systems testing, time-series analysis, and synthetic data generation across various domains, including sensor data modelling and machine learning.

The SigVarGen framework is fully documented and readily accessible. Detailed documentation, including comprehensive tutorials, illustrative examples, and an extensive description of available modules, ensures easy adoption and facilitates community-driven enhancements. The software is publicly available under the MIT License.

Experimental validations through detailed case studies confirmed the framework’s effectiveness. Case Study 1 demonstrated that baseline synthetic signals generated for embedded platforms closely matched empirical measurements obtained via the PowerFuzzer setup. Statistical analyses, including hypothesis tests and distributional similarity measures, revealed high statistical fidelity between synthetic signals and real-world counterparts. In Case Study 2, we evaluated the framework’s capacity to replicate realistic measurement variability observed across repeated observations of the same event. Comparative analyses employing [PCA](#), distribution analysis, and pairwise statistical evaluations collectively verified substantial overlaps between synthetic signal variations and actual measurement distributions.

In conclusion, SigVarGen provides a comprehensive and reliable solution for generating

realistic synthetic signals under consistent environmental conditions. Rather than replicating the exact structural complexity of real-world signals, the framework focuses on efficiently producing diverse, configurable scenarios, including edge cases, with high statistical fidelity and minimal setup, making it particularly well-suited for testing, benchmarking, and algorithm development in data-limited environments.

Chapter 4

SR&OS: Sample Rate & Offset Search

Effective powertrace-based fuzzing requires precisely capturing the system’s response to input stimuli, which hinges on correctly selecting the sampling rate and acquisition offset. Misalignment can obscure key system behaviours or introduce noise that degrades anomaly detection accuracy. This chapter introduces the [SR&OS](#) algorithm, a dynamic optimisation method that calibrates these parameters by identifying and isolating clear response windows in power measurements. The chapter begins with a formal problem definition, followed by a step-by-step explanation of the methodology, including preprocessing, response identification, adaptive binary search, and multi-offset evaluation. It concludes with experimental validation demonstrating improved trace clarity and response coverage compared to grid-based approach originally employed by PowerFuzzer.

4.1 Introduction

Powertrace-based fuzzing aims to detect undocumented behaviours by analysing a system’s power consumption patterns in response to both valid and fuzzed commands. A critical aspect of this approach is ensuring that power measurements accurately capture the system’s transient responses to incoming inputs. However, achieving reliable data collection requires precise offset calibration t_{offset} and sampling rate selection f_{sample} , collectively defined as [SR&OS](#).

In the context of the PowerFuzzer framework, offset calibration ensures that power

measurements begin at the right moment after command transmission, preventing misalignment that could obscure key system responses. Meanwhile, sampling rate selection dictates the frequency at which power values are recorded over time, influencing the measurement accuracy of transient events based on their duration. Chapter 2.4 describes p_1 , offset and sampling rate calibration, and p_2 , response window length identification, as fundamental challenges in powertrace-based fuzzing, highlighting the impact of incorrect configurations on anomaly detection.

This chapter focuses on developing and evaluating the [SR&OS](#) algorithm, designed to optimise dynamically t_{offset} and f_{sample} based on system feedback through the identification of T_{response} .

4.2 Problem Statement

Given a set of powertraces recorded after the execution of a known command, and a set of powertraces recorded during idle periods, along with a set of valid sample rates and an estimated initial offset range, dynamically search for the optimal combination of sample rate f_{sample} and initial offset t_{offset} that maximizes the detection of a clear and complete response window T_{response} by analyzing deviations between response traces and baseline traces, accounting for variations across multiple executions of the same command, to be able to achieve reliable capture of the complete response window T_{response} , excluding irrelevant background noise or unrelated activity with highest score possible. Formally problem is defined by Equation 4.1

$$(p_1, p_2) = \underset{(f_{\text{sample}}, t_{\text{offset}}, T_{\text{response}})}{\arg \max} \mathcal{S}(f_{\text{sample}}, t_{\text{offset}}, T_{\text{response}}) \quad (4.1)$$

where $\mathcal{S}$ is a scoring function 4.2, that reflects the quality and completeness of the detected response trace under the given sampling rate, offset, and window size.

$$\begin{cases} \mathcal{S} : (f_{\text{sample}}, t_{\text{offset}}, T_{\text{response}}) \mapsto s \in [0, 1] \\ \mathcal{S} = w_c \mathcal{S}_c + w_f \mathcal{S}_f + w_l \mathcal{S}_l + w_u \mathcal{S}_u \end{cases} \quad (4.2)$$

The weights are predefined as:

$$w_c = 0.2, \quad w_f = 0.15, \quad w_l = 0.15, \quad w_u = 0.5$$

The function $\mathcal{S}$ combines four criteria:

- Clarity $\mathcal{S}_c$: Favoring single, non-fragmented response windows:

$$\mathcal{S}_c = \begin{cases} 1, & \text{if } |\{c_i\}| \leq 2 \\ 0, & \text{if } |\{c_i\}| > 2 \end{cases}$$

, where

$$c_i = (T_{\text{response}}^{\text{start}}, T_{\text{response}}^{\text{end}})$$

- Fullness $\mathcal{S}_f$: Ensuring the response finishes before touching end of the trace:

$$\mathcal{S}_f = \begin{cases} 1, & T_{\text{response}}^{\text{end}} \leq 0.9L \\ 0, & \text{otherwise} \end{cases}$$

- Length $\mathcal{S}_l$: Rewarding reasonable response durations:

$$\mathcal{S}_l = \begin{cases} 1, & 0.1 \leq \frac{T_{\text{response}}}{L} \leq 0.5 \\ 0.5, & \text{otherwise} \end{cases}$$

- Uniqueness $\mathcal{S}_u$: Measuring statistical separability between response and baseline traces:

$$\mathcal{S}_u = 1 - \frac{1}{N} \sum_{i=1}^N q_i$$

, where q_i denotes the p-value obtained by comparing the distribution of response traces with distribution of idle traces through T-Test.

4.3 Methodology

This chapter describes the methodology employed by the [SR&OS](#) algorithm within the PowerFuzzer framework. The [SR&OS](#) algorithm uses a hybrid strategy integrating binary search with adaptive offset calibration. It leverages several key functions designed to handle time-series analysis, noise reduction, anomaly detection, and optimal parameter identification, as summarised in the following subsections.

4.3.1 Powertrace Acquisition and Preprocessing

Reliable detection of undocumented behaviours through power trace-based fuzzing requires capturing and preprocessing high-quality power measurements. Inaccurate and noisy traces significantly impact anomaly detection accuracy, necessitating careful data collection and preprocessing methods. The [SR&OS](#) algorithm adopts a systematic approach comprising two critical stages: baseline trace acquisition with background processes activity removal and response trace acquisition.

Baseline Trace Acquisition and Noise Reduction

Baseline traces represent the idle power consumption of the [SUT](#) without external stimuli. Capturing clean and stable baseline traces enables accurate differentiation between normal system behaviour and anomalous responses induced by specific commands.

Baseline powertraces are collected for specified sampling rates and offset using the digitiser hardware through ‘collect_base_traces’ function. The procedure involves setting the desired sampling rate and initial measurement offset, capturing multiple repetitions defined p_count parameter to reduce stochastic variability and ensure robustness against transient background fluctuations, and storing the resulting baseline powertraces as NumPy arrays for subsequent preprocessing and analysis.

After the acquisition, baseline traces are cleaned using the ‘clean_background_noise’ function. The function ‘clean_background_noise’ aims to identify and remove sections of background power traces that exhibit anomalous behaviour across idle traces, potentially caused by external disturbances or device-internal background processes. The process begins with a pairwise comparison between all baseline power traces (referred to as ‘base_waves’). Each pair of traces is compared using a ‘moving_window_feature_comparison’ function.

The function ‘moving_window_feature_comparison’ performs a sliding window comparison between two time-series signals, denoted as ‘wave’ and ‘base_wave’. The comparison is performed using fixed-size overlapping windows, each of length ‘window_size’, with consecutive windows offset by ‘step_size’ samples. The Euclidean distance between the windowed segments is computed for each window pair. This results in a sequence of distance values quantifying the local similarity between the two signals over time. These results highlight sections where the signals diverge.

For each pairwise comparison, outliers within the distance sequence are detected using the ‘detect_outliers_iqr’ function. The function ‘detect_outliers_iqr’ identifies outliers within

a given numerical sequence using the [Interquartile Rang \(IQR\)](#) method. This approach computes the first quartile ($Q1$) and the third quartile ($Q3$) of the data, then derives the interquartile range as $IQR = Q3 - Q1$. Outliers are data points that exceed $Q3 + \alpha \cdot IQR$, where α is a configurable sensitivity parameter.

The detected outliers, corresponding to time windows where the signals deviate unusually, are mapped to time intervals within the trace. These intervals are further merged if they overlap. Once all pairwise comparisons are processed, the algorithm identifies consistent outliers with ‘find_consistent_outliers’. These consistent outliers are assumed to correspond to unwanted noise segments caused by background processes. The detected outlier intervals are removed from each power trace. To preserve the overall length of the traces, the removed sections are replaced by extending the remaining parts of the signal. This produces a set of cleaned baseline traces, with background noise segments removed and reconstructed using the retained clean portions of the signals.

Response traces capture the device’s transient power consumption triggered explicitly by known commands. These traces form the core data set for anomaly detection.

The procedure for ‘collect_response_traces’ follows a similar approach to ‘collect_base_traces’, but focuses on capturing powertraces in response to command execution at specified off-sets and sampling rates. The process begins with the initialisation of a digitizer hardware instance, where measurement parameters such as ‘record_length’ (trace length in samples), ‘count’ (number of repeated measurements for averaging), ‘sample_rate’ (sampling frequency), and ‘skip_samples_post_trigger’ (initial offset before measurement start) are configured according to the experimental setup.

Once the digitiser is initialised, a triggering mechanism activates trace recording upon command transmission. The known command is sent to the target device through a interface (specified by analyst, currently were tested serial, CAN and SSH), initiating the system’s response. Following command execution, the system waits for a predefined duration to ensure complete data acquisition. The waiting time is computed through Equation 4.3.

$$\text{wait_time} = \frac{T_{\text{record_length}} + t_{\text{offset_samples}}}{f_{\text{sample}}} + \delta \quad (4.3)$$

where δ , set to 0.01 seconds, introduces a small buffer to accommodate potential variations in response timing.

To enhance statistical reliability, the trace collection process is repeated for the specified number of traces. Each iteration follows the same sequence: command transmission,

response recording, and an appropriate waiting period. This repetition accounts for potential variability in system behavior and ensures the captured data is representative of the device’s typical response.

After completing the acquisition phase, the digitiser returns the recorded powertraces in the form of a NumPy array. Each collected trace is explicitly linked to its corresponding input command. These recorded traces serve as the foundation for subsequent analysis, where they are compared against baseline traces to detect deviations indicative of system response. The collected response traces are subsequently analysed using the [SR&OS](#) methodology to detect and characterise system responses.

4.3.2 Dynamic Identification of Response Windows

Following powertrace acquisition and preprocessing, the [SR&OS](#) algorithm identifies and characterizes intervals within the recorded traces that correspond explicitly to the device’s response to executed commands. Precisely determining these intervals is critical, as they form the basis for accurate anomaly detection and subsequent parameter optimisation.

The identification of response intervals employs a sliding-window similarity analysis implemented through the ‘identify_response_trace’ function. Initially, each collected response trace (representing device behaviour following command execution) undergoes a comparative analysis against the previously cleaned baseline traces. These baseline traces provide a stable reference, representing typical system behaviour devoid of externally induced activities. Pseudo code for the ‘identify_response_trace’ function is attached as Algorithm 6, and its respective flow chart is attached as Figure C.1 in Appendix C.

For each pairwise comparison between a response trace and a baseline trace, a sliding-window approach is utilised via the function ‘moving_window_feature_comparison’. This method systematically calculates local Euclidean distances between segments of the response trace and corresponding segments of the baseline trace. These calculated distances quantify local deviations, effectively highlighting trace regions exhibiting significant divergence from normal behaviour.

Subsequently, statistical methods robust against outliers identify trace intervals with significant deviations. The algorithm employs the [IQR](#) technique, implemented within the function ‘detect_outliers_iqr’. This non-parametric method identifies anomalous windows within the distance measurements by establishing statistical boundaries based on the data distribution. Trace segments exceeding these statistical boundaries are classified as potential anomalous intervals indicative of genuine device responses.

Algorithm 6: Identify response trace

Result: Consistent response intervals across all known traces

Input: *base_waves* – powertraces during idle activity

known_waves – powertraces after sending known command

WINDOW_SIZE, *STEP_SIZE*, *TRACE_LENGTH* – parameters for sliding window comparison

```
1 dist_bases = empty list;
2 for wave in base_waves do
3   for other_wave in base_waves after wave do
4     compare wave and other_wave using sliding window;
5     add all distances to dist_bases;
6   end
7 end
8 total_merged_ranges_list = empty list;
9 for wave in known_waves do
10   merged_ranges_list = empty list;
11   for base_wave in base_waves do
12     compare wave and base_wave using sliding window;
13     combine distances with dist_bases;
14     detect outliers using IQR;
15     convert outlier windows to time ranges;
16     merge overlapping ranges;
17     add to merged_ranges_list;
18   end
19   consistent_ranges = find consistent outliers in merged_ranges_list;
20   add consistent_ranges to total_merged_ranges_list;
21 end
22 final_ranges = find consistent outliers in total_merged_ranges_list;
23 return final_ranges;
```

To further enhance reliability and avoid falsely identifying transient or random deviations as genuine responses, the algorithm incorporates a consistency analysis through the ‘find_consistent_outliers’ function. By aggregating outlier intervals from multiple comparisons and assessing their recurrence frequency, the algorithm isolates only those consistently appearing across independent comparisons. This ensures robust identification of actual device responses, minimising the likelihood of false positives arising from sporadic background events or sensor noise. The function follows steps outlined by pseudocode in Algorithm 7.

Algorithm 7: Find consistent outliers

Result: Consistent outlier intervals

Input: merged_ranges_list – list of outlier ranges from multiple comparisons
threshold – fraction of comparisons a range must appear in

```
1 all_ranges = all ranges from merged_ranges_list;
2 sort all_ranges by start index;
3 combined_ranges = empty list;
4 for range in all_ranges do
5   if combined_ranges is empty or does not overlap with last range then
6     | add range to combined_ranges;
7   else
8     | merge range with last range in combined_ranges;
9   end
10 end
11 for range in combined_ranges do
12   | count how many lists in merged_ranges_list contain overlapping range;
13 end
14 min_occurrences = threshold * number of lists;
15 consistent_ranges = ranges from combined_ranges with count  $\geq$  min_occurrences;
16 return consistent_ranges;
```

The outcome of this analysis is a set of response intervals corresponding to the device's reaction following command execution. These intervals establish well-defined boundaries for the response window, ensuring that power consumption variations attributed to the command are correctly isolated from background noise. Identifying this response window, denoted as T_{response} , is a critical step in the optimisation process, as it directly influences subsequent binary search refinements of sampling rate and offset parameters. By prioritising the most prolonged detected responses, the algorithm ensures that the most complex and extended device reactions are captured first, maximising detection reliability. This approach effectively addresses p_2 , the challenge of response window identification, by dynamically refining the observation window to include the most relevant segments of powertrace activity.

4.3.3 Optimal Parameters Search via Adaptive Binary Search

The [SR&OS](#) algorithm employs a structured binary search approach to optimise (f_{sample}) and (t_{offset}). These parameters determine the accuracy and effectiveness of powertrace-

based fuzzing by ensuring that the recorded power measurements precisely capture the device’s transient response to command execution. An inadequate selection of these parameters can lead to incomplete or misaligned powertraces, reducing the reliability of anomaly detection.

The parameter optimisation process is implemented through the function ‘search_optimal_params_bs’, which systematically refines the sampling rate and offset by iteratively evaluating different configurations. The method operates over a stack-based binary search space, where each state in the search represents a bounded interval of candidate sampling rates and an associated initial offset. This approach ensures efficient convergence toward an optimal configuration while minimising unnecessary evaluations.

The ‘search_optimal_params_bs’ function follows an iterative process, outlined in Algorithm 8, where the sampling rate search space is iteratively narrowed using a binary search approach across a set of candidate sampling rates to identify the optimal combination of sampling rate and initial offset that maximises detection performance for a known command. This search operates over a stack-based search space, where each search state represents a bounded interval of sampling rates and an associated initial offset. The search begins by initialising the full range of valid sampling rates and a predefined initial offset, pushing the entire search space onto a stack. At each iteration, the midpoint sampling rate of the current range is selected as the candidate for evaluation.

To assess the quality of a given sampling rate and offset configuration, the function ‘identify_response_trace’ is invoked. This function extracts the response window T_{response} from the powertraces and computes a detection score that quantifies the clarity and reliability of the response detection. The computed score determines whether the current configuration should be refined further or if an optimal configuration has been reached. If the score exceeds a predefined score threshold, the search terminates early, returning the optimal parameters.

The algorithm dynamically modifies the offset if the detected response appears too early or late within the trace. Depending on the positioning of the detected response intervals, adjustments to t_{offset} ensure that the measurement window aligns correctly with the system’s actual reaction to the command. If the response is split into multiple segments, indicating a double system response, the algorithm shifts the offset to merge the detected response into a single continuous window.

The algorithm accounts for different response detection cases based on the number and position of identified response segments. If a single response is detected, its position within the trace is analysed. If the reaction starts too early or too late relative to the expected T_{response} , the offset is adjusted accordingly. If the response is too short, the sampling

rate search is refined to capture a more extended segment. If the algorithm detects two separate response segments, it computes the time gap between them. If the gap exceeds a predefined fraction of the trace length, the offset is shifted to align the second segment earlier. Otherwise, the two segments are treated as a single merged response, and further search refinement is performed. Otherwise, it is considered that no valid response was detected. In this case, the search space is branched, and higher and lower sampling rates are evaluated to improve trace alignment and resolution.

The search process might terminate according to one of the following reasons. If a candidate configuration yields a detection score higher than the predefined threshold, indicating a high-confidence response detection, the search concludes early, or if the search reaches the maximum allowed number of iterations without identifying an optimal configuration, the best-scoring parameters found so far are returned.

All evaluated parameter combinations are stored in a results list, which is sorted based on descending detection scores. The final output consists of the best parameter set and a ranked list of all evaluated configurations, allowing further analysis.

The structured workflow of the ‘search_optimal_params_bs’ function is illustrated as a flowchart in Figure C.2 Appendix C and the associated flowchart in Figure C.3 Appendix C.

The algorithm systematically converges toward an optimal parameter configuration by incorporating response detection, dynamic offset shifting, and sampling rate refinements.

Algorithm 8: Search optimal sampling rate and offset

Result: Best combination of sampling rate and offset with evaluation scores

Input: *valid_rates*: list of candidate sampling rates; *initial_offset_seconds*: starting offset guess; *trace_length*: length of power traces; *command*: known command sent to device; *window_size*, *step_size*: parameters for windowed comparison; *score_threshold*: minimum acceptable score; *max_iterations*: maximum search depth; *count*: number of traces.

```
1 Initialize stack with full rate range and initial offset;
2 Initialize best_score, all_results;
3 while stack is not empty do
4     pop next search state (rate range, offset, iteration) from stack;
5     if iteration exceeds max_iterations or score exceeds score_threshold then
6         break;
7     end
8     select midpoint sampling rate in current range;
9     compute offset in samples for this rate;
10    evaluate detection score for this rate and offset;
11    store result (score, offset, rate) in all_results;
12    if response consists of exactly two segments then
13        explore higher and lower sampling rates with low priority;
14        calculate time gap between segments;
15        if gap is large then
16            shift offset to align second segment earlier;
17        end
18        treat as a single segment response;
19    end
20    if response consists of one segment then
21        adjust offset if response starts too late;
22        refine sampling rate search if response is too short or too long;
23        explore with high priority;
24    end
25    else
26        explore both higher and lower sampling rates;
27    end
28 end
29 sort all_results by descending score;
30 return best result and all_results;
```

4.3.4 Multi-Offset Evaluation

The precise timing of a system’s response to a given command is often variable due to factors such as execution delays, processing overhead, and communication latencies. Relying on a single initial offset assumption may lead to non-optimal detection of response intervals. To mitigate this, multi-offset evaluation* is employed through the ‘find_optimal_params’ algorithm, which systematically explores a range of offsets alongside sampling rate variations to maximise response identification accuracy.

This method extends the binary search-based optimisation ‘search_optimal_params_bs’ by iterating over multiple candidate offsets. Instead of assuming a fixed offset, the algorithm evaluates a sequence of offsets spaced uniformly based on the total trace length and maximum sampling rate. Each offset is treated as an independent starting point for the optimisation process. The multi-offset evaluation begins by generating a set of candidate offsets, typically defined by Equation 4.4.

$$\text{offset_step} = \frac{\text{trace_length}}{\text{max_sampling_rate}} \quad (4.4)$$

This ensures that offset increments are small enough to capture fine-grained variations in response timing while maintaining computational efficiency. For each offset in the candidate set, the ‘search_optimal_params_bs’ function performs binary search optimisation over the valid sampling rate range.

To improve efficiency, an early termination criterion is applied: if any evaluated configuration achieves a detection score exceeding a predefined threshold, the search terminates early, returning the best result found so far. Otherwise, all results are sorted by descending detection score, allowing the most effective parameter configuration to be selected.

The multi-offset evaluation process is described by Algorithm 9 with the corresponding flow chart attached as Figure C.4 Appendix C. These representations provide a structured view of the iterative process used to refine sampling rate and offset selections, ensuring that powertrace-based response identification is precise and adaptable across different system conditions.

Algorithm 9: Find optimal sampling rate and offset

Result: Best combination of sampling rate and offset with evaluation scores

Input: `valid_rates` – list of candidate sampling rates

`initial_offset_seconds` – starting offset guess

`trace_length` – length of power traces

`command` – known command sent to device

`window_size`, `step_size` – parameters for windowed comparison

`score_threshold` – minimum acceptable score

`max_iterations` – maximum search depth

`count` – number of traces to evaluate

`num_offsets` – number of offsets to try

```
1 if valid_rates is empty then
2   | return no result;
3 end
4 compute step size between offsets based on maximum rate;
5 generate list of offsets to evaluate;
6 all_results = empty list;
7 for offset in offsets to evaluate do
8   | result = binary search for optimal rate using this offset;
9   | add all results from this search to all_results;
10  | if best score in this search exceeds score_threshold then
11    | break;
12  | end
13 end
14 sort all_results by descending score;
15 if all_results is empty then
16   | best_result = no valid result;
17 end
18 else
19   | best_result = first result in sorted all_results;
20 end
21 return best_result and all_results;
```

4.3.5 Conclusion

Outlined methodology directly addresses the challenges posed by p_1 Offset and Sampling Rate Calibration and p_2 Response Window Identification as formulated in the problem

statement. The **SR&OS** algorithm is designed to dynamically search for the optimal sampling rate f_{sample} and offset t_{offset} that maximise the detection of a complete response window T_{response} while minimising background interference. By systematically adjusting the sampling rate and offset, the algorithm ensures that critical response moments are fully captured and adequately aligned, effectively solving p_1 . Additionally, by leveraging response identification and prioritising most prolonged detected responses, the methodology refines T_{response} to exclude extraneous background activity while ensuring full capture of the system’s reaction, thus addressing p_2 .

4.4 Experimental Results & Evaluation

This section evaluates the effectiveness, performance, and scalability of the proposed **SR&OS** algorithm through a series of experiments conducted on four representative embedded platforms. It describes the experimental setup, including hardware configurations, communication protocols, and trace acquisition procedures. Subsequent analyses assess the algorithm’s computational efficiency and memory usage, followed by a comparative evaluation against conventional grid search methods. Finally, detailed experimental results across all test platforms demonstrate the algorithm’s ability to dynamically identify sampling parameters and response windows under varying real-world conditions.

4.4.1 Experimental Setup

To assess the performance and generalizability of the **SR&OS** framework, we conducted experiments on four representative embedded systems: OpenEVSE Charger Development Board, SIM900 GPRS/GSM Module, Raspberry Pi, and BeagleBone Black. These targets were chosen to cover a range of use cases, complexity levels, and communication interfaces commonly encountered in real-world embedded applications. All setups follow the general architecture described in Section 2.3, using a synchronised acquisition system with hardware-based triggering, shunt resistor-based current sensing, and digital powertrace collection.

For each target system, we captured base traces, representing idle or background power consumption, and response traces, representing system power profiles following the application of a specific command. All experiments used raw powertraces without filtering, denoising, or post-processing enhancements, ensuring that any observed anomalies were intrinsic to the system response itself.

Each trace captured consisted of 1,000,000 samples, as defined by a fixed recording length, with sample rates determined from a valid range based on integer downsampling of the base digitiser rate of 2.5 GHz. Valid sample rates s were found through Equation 4.5. Traces were captured with a consistent trigger level.

$$R = \left\{ r \in \mathbb{N} \left| r = \frac{f_{\text{base}}}{n}, n \in \mathbb{Z}^+, f_{\text{base}} \bmod n = 0, r \in [f_{\text{min}}, f_{\text{max}}] \right. \right\} \quad (4.5)$$

Equation 4.5 defines the set R of valid sampling rates used for digitiser configuration. Each valid sampling rate r is obtained by integer division of the digitiser’s base clock frequency $f_{\text{base}} = 2.5 \times 10^9$ by a positive integer downsampling factor $n \in \mathbb{Z}^+$. The condition $f_{\text{base}} \bmod n = 0$ ensures that the result of the division is an integer-valued sample rate. Furthermore, the resulting rate r must lie within the user-defined bounds $[f_{\text{min}}, f_{\text{max}}]$, where $f_{\text{min}} = 10^5$ and $f_{\text{max}} = 10^8$. This construction guarantees hardware-compatible sampling rates that preserve timing resolution while meeting acquisition constraints.

The Raspberry Pi device served as high-level embedded Linux systems with full operating system support and standard user-level access via SSH. This configuration allowed the fuzzing of shell commands and system daemons through a remote shell. The device was accessed via SSH, and commands were transmitted through a pseudo-terminal session. Simultaneously, a hardware trigger was activated by sending a UDP packet to a connected Arduino, which initiated synchronised data capture through the digitiser. Idle traces were collected by issuing trigger signals without command execution, while response traces were collected by executing commands such as ‘ps’.

The OpenEVSE board is an embedded system designed for managing electric vehicle charging. Its deterministic state transitions and command-specific power profiles make it suitable for evaluating the detection of undocumented or unsafe command responses in a critical infrastructure context. It processes a domain-specific set of [Remote Application Programming Interface \(RAPI\)](#) commands for monitoring and control, making it an ideal candidate for protocol-aware fuzzing. The device was connected over a serial [UART](#) interface at 115200 baud and fuzzed with known commands such as ‘GV’. Commands were dispatched to an integrated web server that relayed [RAPI](#) messages internally using HTTP-based [Representational State Transfer \(REST\)](#) calls.

SIM900 GPRS/GSM Module is a representative example of a communication-centric embedded system. It responds to AT commands for SMS, call, and network operations over [UART](#). The SIM900 module exemplifies a low-power, communication-oriented embedded system that processes AT commands over [UART](#). It operates in GSM bands and supports commands for querying device state, signal strength, and network configuration. The

module was interfaced through a serial connection at 19200 baud, and commands such as ‘AT+GSN\r\n’ (request device serial number) were sent directly to the device.

The BeagleBone Black is a Linux-capable microcontroller board with real-time processing capabilities. It operates as a real-time platform that supports both low-level firmware and OS-level operations. It was accessed via [UART](#) at 115200 baud and executed computational tasks to provoke measurable power responses. Command ‘fp_addition 10000\n’ was transmitted to initiate floating-point operations on the device and to initiate a response trace. As with the other targets, base traces were collected without any command transmission, while response traces captured the power signature resulting from command execution.

Each of the experimental setups described above operated using a consistent hardware/software coordination layer. Commands were systematically issued, and corresponding powertraces were recorded synchronised using the methodology from Algorithm 1. No post-acquisition enhancement or filtering was applied, preserving the integrity of raw traces.

4.4.2 Performance and Scalability Analysis

This section provides a detailed evaluation of the algorithm’s performance characteristics, focusing on its computational and memory requirements. Through formal analysis of both time and space complexity, we quantify the scalability of the ‘find_optimal_params’ algorithm with respect to key parameters such as trace length, number of collected traces, sampling rates, and offset candidates. Understanding these factors is essential for assessing the algorithm’s feasibility in real-world settings, particularly in scenarios involving extensive data and constrained computational resources.

Space Complexity

The space complexity of the ‘find_optimal_params’ procedure reflects the cumulative memory requirements across all stages of trace collection, preprocessing, and parameter optimisation. While the algorithm is primarily computationally intensive, efficient memory use is equally critical given the size of high-resolution power traces.

The initial stage involves storing base and response power traces, collected using ‘collect_base_traces’ and ‘collect_response_traces’. These are stored in NumPy arrays of shape (c, L) , leading to a space complexity of $O(c \times T)$, where c is a number of collected traces

and L is the length of powertrace. This data is retained throughout most of the pipeline, representing the algorithm’s baseline memory footprint.

The ‘remove_outlier_segments_and_extend’ function allocates a boolean mask and a cleaned version of the trace, both of length L , resulting in $O(L)$ space usage per trace. The ‘clean_background_noise’ and ‘identify_response_trace’ functions operate over all collected traces, with temporary buffers used for distance computations and outlier aggregation. While these structures can momentarily scale to $O(c \times L)$, they are reused and discarded within function scope, preventing unbounded memory growth.

The ‘moving_window_feature_comparison’ function computes a distance array over sliding windows, introducing additional space usage proportional to the number of windows, approximately $O(L/\text{step_size})$. Other operations such as outlier detection (‘detect_outliers_iqr’) and merging intervals (‘merge_overlapping_indices’, ‘find_consistent_outliers’) operate on compact representations such as lists of index ranges or boolean masks and contribute only a small additional overhead, typically negligible relative to the trace size. The ‘generate_valid_rates’ function produces a relatively small list of candidate rates, which is neglectable as well.

In ‘evaluate_params’, space usage is dominated by the retained traces and the temporary buffers for statistical evaluation. The ‘search_optimal_params_bs’ function maintains a stack, a set of visited rates, and a result accumulator ‘all_results’, each growing up to $O(R)$, where R is the number of valid sampling rates. The outer function ‘find_optimal_params’ iterates over N offsets, resulting in total auxiliary space usage of $O(N \times R)$ for collecting and aggregating results.

The overall space complexity of the procedure is therefore defined by Equation 4.6.

$$O(c \times L) + O(N \times R), \tag{4.6}$$

where the first term accounts for persistent storage of power traces, and the second term reflects the overhead from parameter optimisation. In practical applications, the values of c , L , N , and R are constrained by system limits and experimental design, ensuring that memory usage remains tractable even in large-scale scenarios.

Time Complexity

The time complexity of the ‘find_optimal_params’ function is dominated by two nested iterative structures. First, the function iterates over a set of candidate offsets, whose

number is determined by the parameter ‘num.offsets’ denoted as N . For each offset, it calls the ‘search_optimal_params_bs’ function, which employs a binary search strategy over the list of valid sampling rates of length R . Within this binary search, the while-loop is bounded by a maximum iteration limit I , ‘max.iterations’, so its worst-case execution involves at most I iterations. In each iteration, the function calls ‘evaluate_params’, which performs several computationally intensive operations such as data collection, noise cleaning, and response identification.

More specifically, ‘evaluate_params’ calls functions like ‘collect_base_traces’ and ‘collect_response_traces’, each of which iterates a fixed number of times c , and then proceeds to call ‘clean_background_noise’ and ‘identify_response_trace’. These latter functions use sliding-window comparisons that run in $O\left(\frac{L}{S}\right)$ time per trace, where L is the trace length and S is the step size influencing the number of iterations, as well as nested loops over the number of collected traces. Therefore, the cost of ‘evaluate_params’ can be approximated as $O\left(c^2 \cdot \frac{L}{S}\right)$, where c is the number of traces used for evaluation, L is the trace length and S is the step size.

Combining these factors, the worst-case overall time complexity of ‘find_optimal_params’ is expressed by Equation 4.7.

$$O\left(N \times I \times \left(c^2 \cdot \frac{L}{S}\right)\right) \quad (4.7)$$

Equation 4.7 reflects algorithms’ linear dependency on the number of offsets and the maximum iterations and a quadratic dependency on the trace collection and processing steps. In practice, since I and c are often set to modest constants, and S is chosen relative to L , the complexity is effectively controlled, but the algorithm remains computationally intensive due to the inherent cost of processing high-resolution powertrace data.

4.4.3 Comparative Analysis: Adaptive Binary Search vs Grid Search

A comparative analysis was conducted against the traditional baseline method of grid search to justify the necessity and efficacy of the proposed adaptive binary search. The comparison focuses on parameter optimisation performance using predefined responses, deliberately excluding the complexity introduced by response identification to maintain simplicity and clarity.

Comparison Methodology and Implementation

The comparative assessment utilised a distance-based scoring function formulated to measure how optimally a command aligns within a given recorded trace. This scoring function evaluates three distinct criteria:

- **Early Command Start:** Ideally, the command should commence within the first 10% of the trace. The scoring assigns a higher value the earlier the command begins, decreasing proportionally as the start position moves past the initial 10%.
- **Long Command Duration:** A desirable duration is defined as being at least 50% of the total trace length. This criterion rewards longer commands with higher scores, reaching the maximum when the command duration meets or surpasses half the trace length.
- **Early Command End:** The command’s termination is ideally set to occur before the 90% mark of the trace length. Commands ending earlier than this threshold receive higher scores, with scores diminishing as the ending time approaches or exceeds this mark.

Each criterion is scored independently and combined into a final weighted sum, with the weights currently set at 20% for the early start criterion, 50% for the duration criterion, and 30% for the early end criterion. This balanced weighting system aims to emphasize overall duration and appropriateness of placement within the trace, reflecting realistic operational conditions.

The comparative analysis employs Adaptive Binary Search from Algorithm 8 with several simplifications and adjustments to facilitate clearer performance comparisons with the baseline.

For comparative analysis purposes, we used a simplified scoring function without a response identification function that involves complex extraction and scoring mechanisms for actual hardware-captured powertraces. Whereas the comprehensive, adaptive binary search algorithm handles varied and dynamically identified response segments, including cases of zero, single, or double responses, and adjusts offsets accordingly, the simplified implementation strictly evaluates cases with a single clearly predefined response or no response overlap. The early termination threshold set as 0.65 triggers termination without additional refinement or validation steps. The comparative implementation retains a stack-based iterative approach to explore parameter configurations systematically.

The simplifications were purposefully introduced to isolate the direct computational and methodological advantages of the chosen search methodology versus baseline methodology, allowing more explicit comparative conclusions regarding efficiency, iteration count reduction, and optimal parameter discovery.

We used the grid search implemented through nested linear searches as a baseline method. The grid search implementation exhaustively evaluates all combinations of discrete offsets and sampling rates. To ensure temporal resolution, offsets are iteratively selected based on a fixed step size defined as trace length divided by the max sampling rate, and each is paired with every available sample rate from a predefined list. The evaluation function is invoked for each combination, and raw and synthetic execution times are accumulated using a simplified performance model. Execution is optionally terminated early if a threshold score 0.65 is exceeded.

The comparative analysis employs a well-defined set of scenarios categorised into regular and edge-case groups. Regular scenarios encompass typical operational cases distinguished by varying command durations and positions within the trace – early, mid, and late placements, as well as short and long durations. The edge cases examine the adaptive and baseline methods’ performance in boundary conditions, such as commands beginning immediately before trace initiation, precisely at the trace’s start, exactly fitting the trace length, or slightly exceeding its bounds.

This comprehensive suite of testing scenarios ensures robust evaluation, capturing a broad spectrum of possible operational contexts and highlighting the adaptive approach’s advantage in handling diverse conditions.

Evaluation Scenarios

The comparative assessment incorporates a systematically developed suite of evaluation scenarios, categorised into two principal groups: standard and edge-case scenarios. These scenarios were carefully designed to provide robust testing conditions across diverse realistic operating contexts.

The testing scenarios assume a constant trace length of 1,000,000 samples and examine commands characterised by varying temporal locations and durations relative to the trace length. Sample rates were defined within the range from a minimum rate of 100,000 samples per second to a maximum rate of 100,000,000 samples per second. Table 4.1 presents evaluation scenarios.

These scenarios rigorously evaluate the adaptive binary and linear grid search perfor-

Scenario Name	Start Time (s)	Duration (s)
Early Short Command	0.05	0.02
Mid Short Command	0.50	0.05
Late Short Command	0.95	0.03
Early Long Command	0.10	0.70
Mid Long Command	0.30	0.60
Late Long Command	0.60	0.35
Command Just Before Trace Starts	0.39	0.05
Command Starts at Trace Start	0.40	0.05
Command Just Fits at End	1.29	0.10
Command Exceeds Trace End	1.35	0.10

Table 4.1: Command timing scenarios used for evaluating sampling rate and offset search strategies.

mance across typical operational conditions and challenging boundary conditions, capturing their behaviours under realistic timing constraints.

Comparative Metrics

The comparative performance evaluation considered several explicit numerical metrics to quantify the relative advantages of the adaptive binary search over the linear grid search. Specifically, the following metrics were systematically recorded and analysed for each scenario:

1. **Evaluation Score:** A quantitative value ranging from 0 to 1, computed using the distance-based scoring methodology described in Section 4.4.3. This score reflects how well a given parameter configuration captures the target response interval within the trace.
2. **Iteration Counts:** The total number of parameter evaluations performed before identifying a satisfactory configuration. A configuration is considered satisfactory if it yields a score exceeding the predefined threshold (typically score ≥ 0.65).
3. **Execution Times:**

- **Raw Execution Time (raw_exec_time):** The cumulative time required to collect a single power trace for each evaluated parameter pair. It is computed through Equation 4.8.

$$\text{raw_exec_time}+ = \frac{\text{trace_length} + \text{offset_samples}}{\text{sample_rate}} \quad (4.8)$$

- **Total Execution Time (exec_time):** The cumulative full time required to evaluate a parameter configuration, including hardware overhead, and the acquisition of 10 idle and 10 active traces. It is calculated by Equation 4.9.

$$\text{exec_time}+ = 10 \cdot \frac{\text{trace_length} + \text{offset_samples}}{\text{sample_rate}} + 10 \cdot \frac{\text{trace_length}}{\text{sample_rate}} + 22.1 \quad (4.9)$$

Comparison Results

The adaptive binary search approach demonstrated consistent and effective performance across all ten scenarios. In each case, it successfully identified sampling rate and offset configurations that met or exceeded the predefined detection score threshold of 0.65. Early termination was achieved in every scenario, significantly reducing computation time.

Key performance highlights include:

- **Detection Scores:** Ranged from 0.6617 to 0.7533, with an average score of 0.6944.
- **Iterations to Convergence:** Varied between 3 and 29, with an average of 18.6 iterations.
- **Execution Time:** The average raw execution time was approximately 8.55 seconds, and the average total execution time was 548.93 seconds, including all overheads.

The grid search method exhaustively evaluated all combinations of candidate offsets and sampling rates. While this ensured full search space coverage, it came at a significant computational cost and did not guarantee optimal results.

- **Detection Scores:** Only three scenarios achieved a score above 0.65. The average score across all scenarios was 0.4783, with the lowest at 0.2713.

- **Evaluations:** Most scenarios required the full sweep of 1140 evaluations, with an average of 905 iterations.
- **Execution Time:** The average raw execution time was approximately 1558.49 seconds, and the average total execution time exceeded 49,943.13 seconds.

Despite exhaustive sampling, grid search often failed to discover viable configurations, particularly in cases with partial or marginally aligned responses.

Table 4.2 presents a comparison summary between adaptive binary search and grid search across various scenarios. These results confirm that adaptive binary search effectively adapts to varying command timing and maintains high response alignment while requiring significantly fewer iterations and execution time.

Metric	Adaptive Binary Search	Grid Search
Average Score	0.6944	0.4783
Best Score	0.7533	0.6667
Worst Score	0.6617	0.2713
Average Iterations	18.6	905
Avg. Raw Execution Time (s)	8.55	1558.49
Avg. Total Execution Time (s)	548.93	49943.13
Scenarios Meeting Threshold (Score ≥ 0.65)	10/10	3/10

Table 4.2: Comparison of adaptive binary search and grid search across 10 scenarios.

Conclusion

The presented numerical comparison demonstrates the superior efficiency and effectiveness of the adaptive binary search approach relative to the traditional linear grid search baseline. Across all evaluated scenarios, the adaptive binary search consistently identified optimal or near-optimal parameter combinations significantly faster, reducing the necessary iterations. Correspondingly, substantial reductions in raw and total execution times show the computational savings achievable through adaptive methods. These findings highlight the importance of intelligent search strategies in powertrace-based command analysis and justify adopting adaptive binary search as a core component of the proposed [SR&OS](#) framework.

4.4.4 Experimental Results

This subsection presents the experimental results of the [SR&OS](#) algorithm applied to four distinct embedded systems: the OpenEVSE Charger Board, SIM900 GPRS/GSM Module, Raspberry Pi (via SSH), and BeagleBone Black. These targets represent a diverse range of hardware capabilities, communication interfaces, and system complexities, providing a comprehensive evaluation of the algorithm’s effectiveness across different real-world conditions. For each device, the optimal sampling rate and offset were determined through multi-offset adaptive search, and the response window T_{response} was identified from the collected powertraces.

BeagleBone Black

The evaluation of the BeagleBone Black, a Linux-capable embedded platform with real-time processing capabilities, involved using the command ‘fp.addition 10000\n’ aimed to generate a prolonged and distinguishable power signature. Due to the presence of a full operating system, the device inherently exhibits semi-periodic background activity, often manifesting as abrupt and short power spikes unrelated to direct command execution. These artefacts present a challenge in isolating genuine response windows and may interfere with detection metrics.

The [SR&OS](#) algorithm was executed with a total trace length of 1,000,000 samples, using a sliding window of 500 samples and a step size of 250. The multi-offset evaluation was configured with three offset candidates, though the response onset was found to occur immediately after the trigger, rendering additional offsets unnecessary. The algorithm converged after 18 evaluations, identifying the optimal configuration as a sampling rate of 25 MHz and an offset of 0 samples. The detected response window spanned samples 10,250 to 305,500, corresponding to a clear and sustained increase in power usage. [Figure 4.1](#) illustrates three representative response traces with the identified response window highlighted.

The final detection score using the legacy scoring scheme was 0.881, with individual components broken down as follows: clarity $\mathcal{S}_c = 0.15$, fullness $\mathcal{S}_f = 0.35$, length $\mathcal{S}_l = 0.2$, and uniqueness $\mathcal{S}_u = 0.3$. Some iterations during the optimisation process yielded high detection scores but did not converge toward the correct response window. This behaviour was primarily attributed to a combination of factors: certain traces exhibited abnormally long response segments compared to the majority, and transient background tasks occasionally introduced power spikes that resembled genuine responses. Although these effects were not consistently present across all collected traces, their presence in a

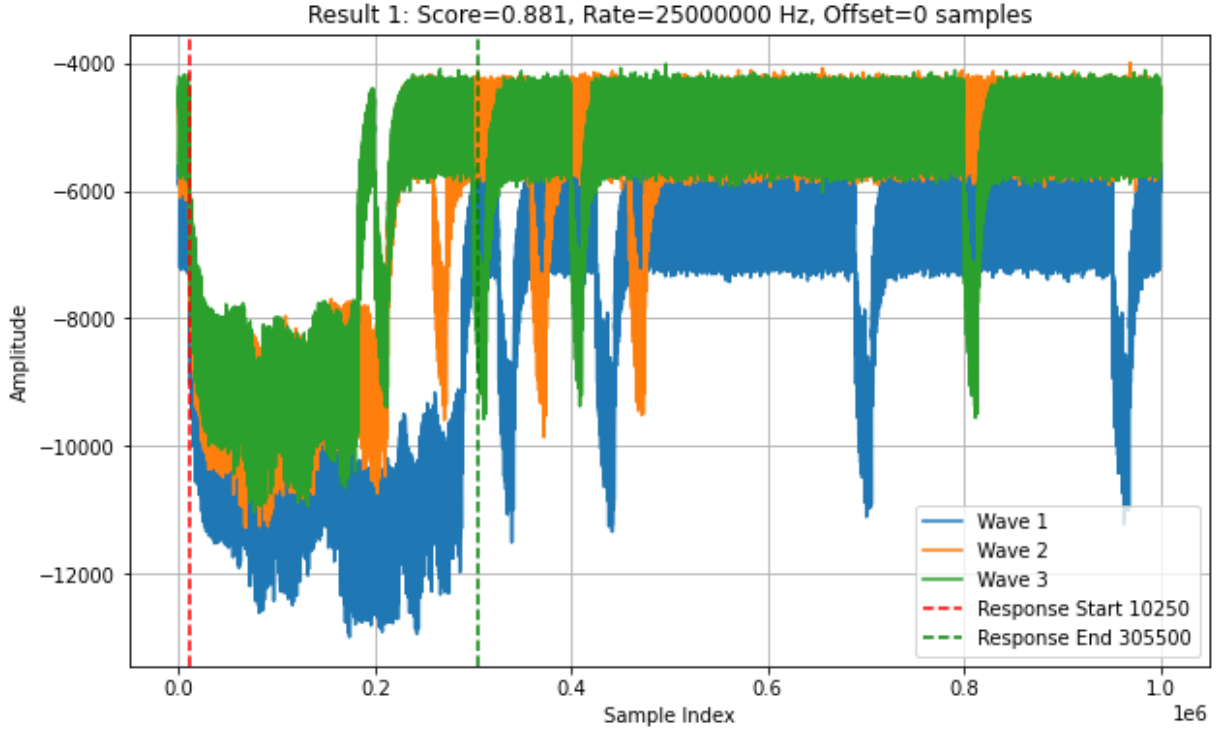


Figure 4.1: BeagleBone Black: Three representative response traces with the identified response window using optimal parameters

subset of measurements was sufficient to inflate the uniqueness component of the detection score, occasionally misleading the algorithm toward suboptimal parameter configurations.

Despite these challenges, the algorithm demonstrated robust performance by consistently identifying the actual response window in most evaluations. The high-resolution sampling rate facilitated accurate capture of the transient response, and the adaptive refinement process effectively avoided misalignment. Overall, the results validate the applicability of the [SR&OS](#) framework to complex, multi-process systems such as the BeagleBone Black and underscore its ability to distinguish relevant system responses in the presence of interfering frequent background activity when it is shorter than analysed response.

OpenEVSE Charger Board

The OpenEVSE Charger Board is a domain-specific embedded system for electric vehicle charging control. It operates deterministically and communicates via predefined [RAPI](#)

commands. In this experiment, the ‘GV’ command, which queries the general status of the charger, was used to trigger a known system response. The target exhibits a characteristic idle behaviour composed of high-frequency white noise, resulting in relatively flat but noisy traces. This idle noise complicates detection, as the power response induced by the command execution is subtle – manifesting as a smooth, gradual increase followed by a symmetrical decay rather than a sharp, discrete spike.

The [SR&OS](#) algorithm was run over a trace length of 1,000,000 samples, with a sliding window size of 500 and a step size of 250. A multi-offset strategy with three initial offsets was employed. The algorithm converged in 9 iterations toward an optimal configuration with a sampling rate of 15.625 MHz and an offset of 156,000 samples. The detected response window spanned from sample 0 to 156,250, encompassing the expected slow power rise and fall in response to the command. The total detection score, computed using the legacy scoring metric, was 0.858. Figure 4.2 shows three selected response traces and the corresponding response window.

While the device’s operational simplicity facilitated relatively consistent trace patterns, pervasive noise reduced the prominence of the power response. It requires precise alignment and adequate sampling to capture the subtle variations. An interesting behavioural distinction was observed between known and unknown command executions. Specifically, unknown commands tend to provoke sharper and earlier power spikes compared to known commands, potentially due to the absence of preprocessing or validation steps. This behaviour could result in misalignment if offsets are explicitly tuned for known command responses. However, as the current evaluation is focused on known command behaviour, the identified parameters are well-suited and reflect stable detection across traces.

Overall, the results demonstrate the effectiveness of the [SR&OS](#) algorithm in handling subtle and noise-obscured responses in deterministic embedded systems. Despite the challenges posed by signal smoothness and background noise, the framework identified the appropriate sampling rate and offset required to capture the whole response window, validating its applicability in noisy targets.

SIM900 GPRS/GSM Module

The SIM900 module, a communication-centric embedded system operating in GSM bands, represents the most complex and challenging target among the evaluated platforms. The module communicates over a UART interface and responds to AT commands, such as ‘AT+GSN’, used in this experiment to issue a query to get a device serial number. Unlike the previously analysed systems, the SIM900 exhibits highly dynamic background activity,



Figure 4.2: OpenEVSE Charger Board: Three representative response traces with the identified response window using optimal parameters

characterised by two distinct types of power spikes: small but frequent fluctuations and more pronounced, less frequent bursts. This dual-mode noise profile introduces substantial variability in the baseline signal, severely complicating response detection.

The [SR&OS](#) algorithm was applied using a trace length of 1,000,000 samples, with a sliding window of 500 and a step size of 250. A broader multi-offset search strategy was employed to account for the long and uncertain delay between command transmission and system response, exploring 20 offset candidates and allowing up to 70 optimisation iterations. The algorithm converged after 70 evaluations, identifying a sampling rate of 9.76 MHz and an offset of 488,000 samples. The final detection score was 0.667. [Figure 4.3](#) visualises three response traces alongside the identified full-trace response window.

However, the detected response window spanned the entire trace length (0 to 1,000,000 samples), indicating an incorrect response window classification and a failure to isolate the true system response effectively. Manual inspection revealed that the actual response

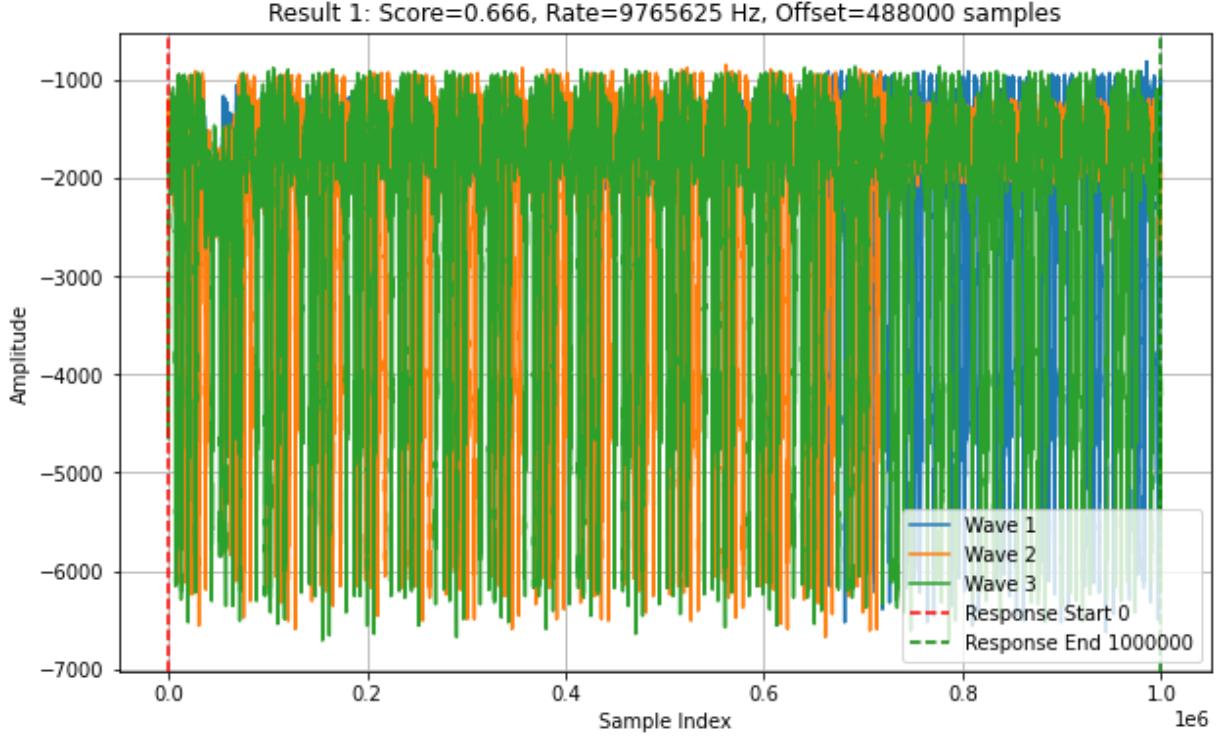


Figure 4.3: SIM900 GPRS/GSM Module: Three response traces with the automatically detected full-length response window

occurred between samples 20,000 and 80,000. The correct response was characterised by a minor downward shift in minimum power levels while the maximum remained unchanged, resulting in a subtle deviation easily obscured by the surrounding noise.

Moreover, throughout the optimisation process, several iterations resulted in false positive response identifications, often driven by GSM background activity being misinterpreted as command responses. These inaccuracies impacted clarity $\mathcal{S}_c$, fullness $\mathcal{S}_f = 0.35$ and length $\mathcal{S}_l = 0.2$ components, leaving only uniqueness $\mathcal{S}_u$ being robust, leading to parameter sets that were locally optimal according to the scoring function but semantically invalid. Figure 4.4 illustrates three traces showing the complexity of the idle pattern contributing to false positive detections.

This experiment underscores the limitations of automated parameter optimisation in contexts where noise dominates signal features and where command responses are weakly pronounced. The high number of false positives and inconsistent trace-level detections

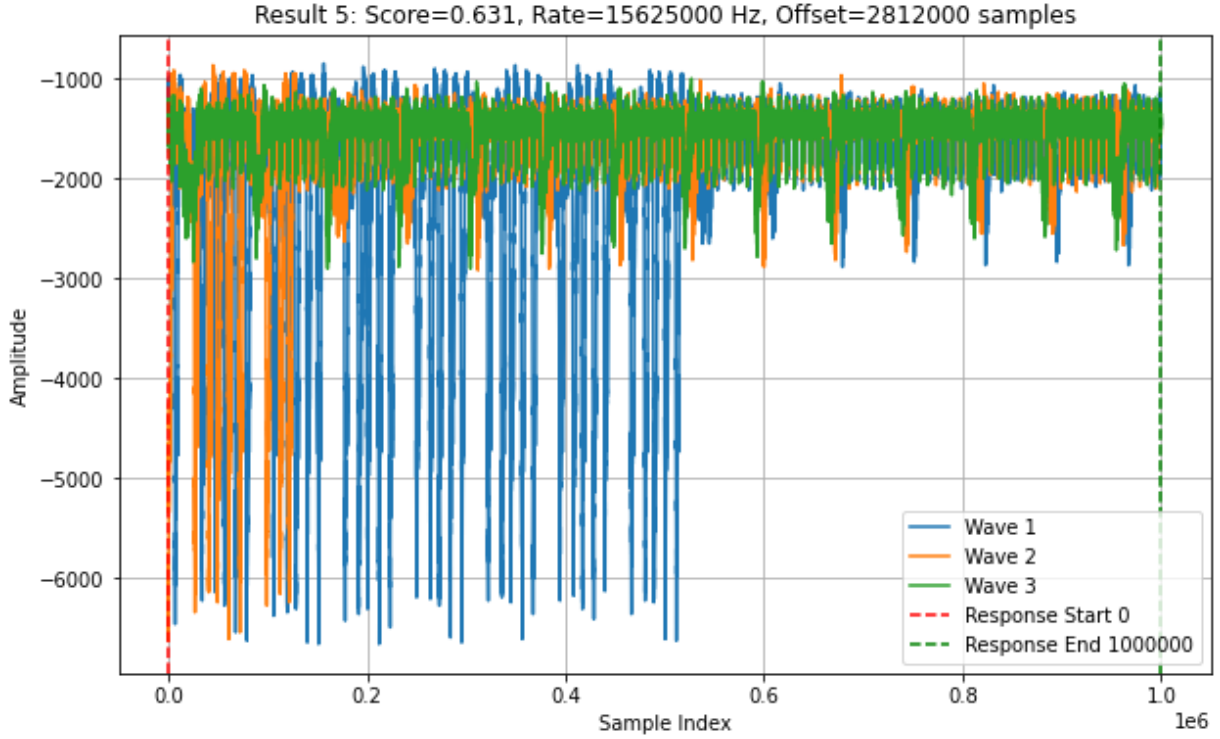


Figure 4.4: SIM900 GPRS/GSM Module: Idle pattern

point to a need for manual inspection to verify algorithm outputs. The `visualize_results` function within the [SR&OS](#) framework proved especially valuable in diagnosing these issues by enabling visual cross-validation of the response window across all evaluated configurations.

In summary, while the [SR&OS](#) algorithm was able to converge and produce a high-scoring parameter set, the identified configuration did not correspond to the true response window. These findings highlight the necessity of integrating analyst-in-the-loop verification mechanisms for noisy or communication-heavy embedded systems such as the SIM900, and suggest that manual verification is required in high-noise environments where legitimate responses are indistinct relative to the background signal.

Raspberry Pi through SSH

The Raspberry Pi, functioning as a general-purpose embedded Linux system, was evaluated in a remote-access configuration, where commands were transmitted via SSH over a pseudo-terminal session. The evaluated command intended to induce a brief but discernible computational load, reflected in the device’s power consumption. Due to the presence of a full operating system and multi-process background services, we expected to see high noise with background tasks.

The [SR&OS](#) algorithm was executed on traces comprising 1,000,000 samples, with a window size of 500 and a step size of 250. The multi-offset search strategy was employed with three offset candidates. After 14 iterations, the algorithm converged on a configuration with a sampling rate of 2.5 MHz and an offset of 955,000 samples. The identified response window ranged from sample 49,500 to 119,500, capturing the characteristic command-induced power spike. The corresponding legacy detection score was 0.753, with the response spike detected despite a substantial offset delay introduced by SSH session latency. [Figure 4.5](#) presents three sample response traces and highlights the identified response interval.

The Raspberry Pi exhibited a uniform high-frequency white noise pattern during idle activity. Despite this, the response to the command manifested as a short, deterministic power spike with a consistent shape across repeated measurements. While the detection outcome aligns with the known system behaviour, it is notable that the chosen sampling rate may not be optimal for capturing the transient nature of the response. The relatively low frequency could limit temporal resolution, potentially smoothing or partially obscuring the sharp edges of the power spike. Visual inspection of the response trace suggests the existence of a small but consistent spike occurring near the beginning of the trace, possibly corresponding to early system activity triggered by the SSH connection itself. This artefact may introduce difficulty in identifying the true response from the SSH activity.

Despite these limitations, the algorithm successfully isolated the correct response interval, demonstrating its robustness under SSH-induced latency conditions. However, this experiment suggests that analyst-in-the-loop strategies or refinement will yield improved resolution and more reliable detection performance in specific contexts. Overall, the results affirm the [SR&OS](#) framework’s adaptability to complex, OS-driven systems such as the Raspberry Pi, though they also underscore the importance of sampling rate selection and human oversight when dealing with complicated setups or targets.



Figure 4.5: Raspberry Pi (SSH): Three representative response traces with the identified response window using optimal parameters

Experimental Results Summary

The experimental results demonstrate the versatility and robustness of the [SR&OS](#) framework across embedded environments. While deterministic systems such as the OpenEVSE Charger yielded clean and stable responses, the BeagleBone Black, Raspberry Pi and SIM900 modules introduced challenges due to operating system latency and background noise. In most cases, the algorithm successfully identified valid configurations and captured the actual response interval. The GSM module highlighted the limitations of automated scoring in noisy environments, necessitating manual verification. Overall, the experiments validate the algorithm’s capacity to adaptively calibrate sampling and offset parameters, achieving meaningful and reproducible response detection even under suboptimal or variable operating conditions.

4.5 Conclusion and Key Findings

This chapter focused on addressing a key requirement in powertrace-based fuzzing: the ability to accurately capture a system’s response to an input command through properly aligned and well-sampled power measurements. In the context of the PowerFuzzer framework, this capability is critical for assessing how a system reacts to both valid and fuzzed commands. Specifically, the chapter focused on solving two tightly coupled challenges – p_1 , the joint calibration of sampling rate f_{sample} and offset t_{offset} , and p_2 , the identification of the true response window T_{response} described in details in Section 2.4. These problems arise due to communication delays, hardware variability, and background system activity, all of which can affect when and how a device responds. Incorrect parameter selection can lead to missing or incomplete response traces, reducing the effectiveness of anomaly detection. The proposed **SR&OS** algorithm was designed to solve this by dynamically searching for the best sampling rate and offset based on observed time-series power trace, allowing consistent and accurate capture of the system’s response with minimal influence from unrelated background activity.

To address the dual challenges of parameter calibration and response window identification, the **SR&OS** algorithm employs a hybrid methodology that combines adaptive binary search with statistical trace analysis. The core approach revolves around iterative refinement of sampling rate f_{sample} and initial offset t_{offset} parameters, driven by feedback extracted from comparisons between response and baseline powertraces. A response detection module is invoked for each candidate configuration, which analyses the trace using sliding-window similarity metrics and statistical outlier detection to isolate intervals of anomalous activity potentially indicative of command-induced responses.

The response window T_{response} is selected based on consistency across traces and duration prioritisation, with preference given to more prolonged and stable deviations from baseline behaviour. Each candidate configuration is evaluated using a composite scoring function $\mathcal{S}$ 4.2, which integrates four heuristics – clarity, fullness, length, and uniqueness. This scoring function enables the algorithm to assess whether a response was detected, and how well it aligns with expectations regarding response completeness and statistical separability from idle traces.

The parameter search process is guided by an adaptive binary search algorithm that efficiently narrows the sampling rate space while dynamically adjusting the offset to align the response interval correctly. The algorithm incorporates a multi-offset evaluation strategy to address uncertainty in response timing, especially in systems with variable latencies. This approach systematically explores several candidate offsets, improving robustness

against delayed or non-deterministic system responses. Early termination is supported when achieving a sufficiently high detection score, improving efficiency without sacrificing accuracy.

By combining robust trace preprocessing, statistical response identification, and adaptive parameter exploration, the [SR&OS](#) methodology offers a generalisable and scalable solution to the calibration problem in powertrace-based fuzzing. It ensures that trace alignment and resolution are optimised dynamically for each system under test, enabling consistent capture of meaningful responses despite variations in execution timing and environmental noise.

The empirical evaluation of the proposed [SR&OS](#) algorithm across four embedded target systems demonstrated its effectiveness, versatility, and computational efficiency. The following key outcomes were observed:

- The algorithm consistently converged toward optimal parameter sets, accurately identifying the true system response windows in the majority of scenarios. For systems such as the BeagleBone Black and OpenEVSE Charger Board, the algorithm quickly identified well-defined responses, achieving legacy detection scores of 0.881 and 0.858 respectively (Figures [4.1](#), [4.2](#)).
- For systems involving OS-induced latency and remote command triggering, such as the Raspberry Pi via SSH, the algorithm demonstrated robustness against substantial timing offsets. Although the identified sampling rate (2.5 MHz) may not have been optimal for capturing sharp response transients, the algorithm effectively handled the introduced delay, successfully detecting the characteristic power spike with a score of 0.753 (Figure [4.5](#)).
- The SIM900 GPRS/GSM module, characterised by complex, high-frequency background noise, posed significant challenges. Despite substantial computational effort (70 iterations and 20 candidate offsets), the automated search led to several false positives and an incorrect response window identification spanning the full trace length. Manual inspection was required to identify the true response interval clearly, illustrating the limitations of automated scoring in extremely noisy environments (Figures [4.3](#), [4.4](#)).
- Performance and scalability analyses (Equations [4.6](#), [4.7](#)) confirmed that, despite being computationally demanding, the algorithm remains tractable and practical. Its space complexity scales linearly with the number of collected traces and offsets, while the quadratic relationship between collected trace count and processing cost primarily

drives the time complexity. Practical experiments affirmed acceptable runtime and resource usage for real-world scenarios.

- Comparative evaluation against traditional grid search methodologies underscored the efficiency and superior performance of the adaptive binary search strategy. The adaptive approach achieved consistently higher scores (average score 0.6944 versus grid search’s 0.4783), required fewer iterations (average 18.6 versus grid search’s 905), and reduced execution times dramatically (Table 4.2).

The experimental evaluation and analytical insights lead to several critical takeaways relevant for powertrace-based fuzzing:

- Adaptive search of sampling rate and offset is essential in powertrace-based fuzzing. Fixed or exhaustive search strategies, such as grid search, are inefficient and less effective when the response timing is uncertain. The proposed adaptive binary search strategy, enhanced with multi-offset evaluation, was able to converge toward optimal configurations across diverse targets dynamically.
- Embedded systems with frequent background noise, such as the SIM900 GSM module, introduce considerable difficulty for automated methods. Consistent noise patterns may influence the scoring function, resulting in false positives. In such cases, analyst-in-the-loop inspection and visual validation remain necessary to ensure the correctness of the identified response.
- Sliding-window comparison with statistical outlier detection proved essential for accurate T_{response} identification. In traces with 1,000,000 samples and low sampling rates, actual responses occupy a small fraction of the signal and are often masked by background activity. The sliding-window approach enables the detection of subtle, fast-changing deviations from the baseline, while the IQR-based outlier detection provides robustness to variability. Alternative methods, such as neural networks or baseline subtraction, require preprocessing and normalisation, which can obscure prolonged or non-abrupt responses. In contrast, the proposed technique preserves raw signal characteristics and facilitates more interpretable and accurate detection.
- The algorithm, though computationally intensive, is scalable and suitable for practical deployments. The controlled iteration limits, early termination criteria, and bounded memory usage ensure that the SR&OS framework remains efficient even with high-resolution trace data and large configuration spaces.

The [SR&OS](#) algorithm is a foundational component within the broader PowerFuzzer framework, providing the essential calibration step required for reliable anomaly detection. By dynamically identifying the optimal sampling rate and offset, [SR&OS](#) ensures that powertraces are precisely aligned and temporally resolved, allowing system responses to be captured without distortion or truncation. Accurate parameter tuning enables the detection pipeline to focus on genuine anomalies rather than artefacts of misalignment or undersampling. In this way, [SR&OS](#) strengthens the entire fuzzing workflow by improving the quality of collected data and expanding the scope of observable system reactions, ultimately supporting more effective and scalable discovery of undocumented or unintended behaviours in embedded systems.

Chapter 5

Trace Quality Enhancement and Anomaly Detection Methods

The quality and interpretability of acquired power traces fundamentally constrain power-based side-channel analysis in embedded systems. The presence of diverse and often overlapping noise sources, ranging from thermal and electromagnetic interference to jitter and algorithmic noise, can obscure subtle behavioural differences and limit the effectiveness of anomaly detection. This chapter addresses two of the core challenges outlined in Section 2.4: p_3 , which concerns improving trace quality through systematic SNR enhancement; and p_4 , which involves distinguishing normal from anomalous power responses robustly and reliably. To solve p_3 , we present a structured framework for identifying, categorising, and mitigating key noise types that degrade trace clarity, supported by a comprehensive evaluation of post-processing strategies and signal quality metrics. For p_4 , we highlight critical considerations in anomaly detection, including the alignment between denoising methods and classification strategies. Additionally, we propose extending the PowerFuzzer framework with an adaptive mechanism for estimating the number of traces required for reliable analysis, combining prior statistical methods with a novel CP-based prediction algorithm. Through a series of systematisations, empirical analyses, and algorithmic innovations, this chapter delivers a coherent set of methods that improve trace reliability and enhance the fidelity of behavioural inference in powertrace-based fuzzing.

5.1 Systematisation of Knowledge: The Nature of Noise

Power Side-Channel Attack (PSCA)s rely on analysing power consumption traces from a target device to extract secret information. The quality of these power traces is critical – low-noise, well-aligned traces make it easier to detect the subtle data-dependent power variations exploited in attacks. However, in practical embedded systems and other contexts like hardware debugging or performance analysis, power traces are contaminated by many sources of interference and noise. Noise is considered a central issue in **SCA** and, more generally, in any signal-processing application [105]. Such factors reduce the **SNR** of the traces, cause trace misalignment, and typically increase the number of measurements or advanced processing needed to succeed in an analysis [15], [62]. This section surveys a comprehensive range of noise and interference sources that degrade power trace quality, emphasising side-channel attack scenarios on embedded devices. Table 5.1 summarises each noise source, how frequently it is cited in studies, and its typical impact on trace quality.

5.1.1 Thermal and Device Noise

All electronic circuits generate intrinsic device noise due to fundamental physical processes. The most common is thermal noise, also named Johnson-Nyquist noise, from resistive elements, which manifests as random voltage/current fluctuations. Flicker noise ($1/f$ noise) is another device noise that dominates at low frequencies in transistors. These noise sources are inherent and cannot be entirely eliminated [15]. In side-channel contexts, thermal noise is explicitly recognised as a noise contributor that blurs power measurements [62]. Because thermal noise is essentially random and independent between traces, its effect can be reduced by averaging many traces at the cost of requiring more samples for a clear signal [15]. Nonetheless, it sets a baseline noise floor in all measurements. Many studies either assume an additive Gaussian noise model to represent the aggregate of thermal and other independent noise [15] or explicitly mention thermal noise as a limiting factor. In summary, thermal/device noise is universally present and typically reduces trace **SNR** by adding a white-noise floor. For example, suppose the signal from a device toggle is on the order of a few microamps. In that case, the thermal noise in the measurement circuit can be of similar magnitude, obscuring fine details unless many traces are averaged.

5.1.2 EMI and Environmental Noise

EMI from external sources is a major contributor to power trace noise. Power measurement setups can unintentionally pick up ambient electromagnetic noise. For example, nearby radio transmissions, switching noise from other equipment, or mains AC hum at 50/60 Hz. Kocher et al. [62] explicitly noted that ‘electromagnetic radiation’ couples into DPA measurements as a noise source. In practical lab setups, engineers often use shielding (Faraday cages, coaxial shunt connections, etc.) to mitigate EMI, underscoring how commonly this issue is acknowledged as most side-channel measurement guides warn about external EMI. If not controlled, EMI raises the noise floor and injects spurious spikes or periodic interference into the power trace, lowering the SNR and potentially introducing false peaks. For example, ambient EMI noise can be on the order of a few millivolts on the measured signal – comparable to or larger than the subtle voltage drop changes caused by cryptographic operations. Socha et al. survey emphasises that environmental noise is among the parameters that directly impact side-channel attack success [101]. In broader contexts like power debugging, EMI from digital logic or communication interfaces can mask the power signature of the component of interest. Environmental EMI is frequently cited in measurement-focused studies, and its impact is to add broadband or periodic interference, often requiring filtering. For instance, it is recommended to use an anti-aliasing filter before the Analog-to-Digital Converter (ADC) to prevent out-of-band environmental noise from folding into the measured band [110].

5.1.3 Power Supply Noise and Ground Bounce

Noise on the power supply rails of the target device is another significant interference source. Embedded systems share their Power Distribution Network (PDN) across components; fluctuations in supply current from one part of the system can introduce noise in the measured power trace of another part. This includes power supply ripple from voltage regulators, ground bounce and decoupling effects when many transistors switch simultaneously. These supply noises manifest as voltage swings unrelated to the operation of interest, degrading trace quality [15], [74]. Measuring power at a less isolated point can significantly increase noise. For instance, if an attacker places a shunt resistor on the main V_{CC} line of a board, the power traces may be contaminated by the activity of other components on the PCB. In contrast, measuring closer to the target chip yields cleaner traces with reduced noise [76, 78]. Supply noise is commonly cited, especially in hardware security and IC design literature. For example, Becker et al. categorise supply noise as part of physical noise in side-channel models [15]. Power supply noise, modelled with standard deviations up to

30% of the nominal rail, introduces voltage fluctuations substantial enough to degrade gate delays and logic levels, masking the variations associated with single operations [11]. The effect is a lower correlation between power traces and the computational activity, meaning attacks require many more traces to average out the unrelated supply fluctuations. In summary, power supply/ground noise appears in a moderate number of studies, particularly those focusing on power integrity or on-chip sensors and typically reduces trace quality by injecting additional fluctuations and jitter, often necessitating special filtering or a more careful choice of measurement point.

5.1.4 Crosstalk and Coupling from Other Signals

Crosstalk refers to interference caused by capacitive or inductive coupling between adjacent signal lines or circuits. In complex boards or chips, a high-frequency or high-current signal in one part can induce currents or noise in another nearby trace. For example, fast toggling of data, address buses or I/O lines can modulate the power line or radiate into the measurement probe [60]. This type of coupling is a known side-channel concern, where leakage from the digital part of the chip couples through the substrate and can even be detected as unintended radio emissions [20, 19]. In the context of power traces, crosstalk means that unrelated circuit activity produces features in the power trace, degrading the clarity of the trace for analysis. This factor is not always distinctly separated from supply noise, but it is explicitly discussed in some studies, especially for mixed-signal systems or FPGAs. For instance, in multi-tenant cloud FPGAs, one tenant’s activity can create measurable noise in another’s power sensor via long-wire crosstalk [48]. Crosstalk is mentioned less frequently in older single-chip attack papers since those often assume a simple microcontroller with few concurrent signals, but it is gaining attention in recent work on complex [System on a Chip \(SoC\)](#)s and boards. Its impact is similar to other interference: it adds spurious power consumption events that are not correlated with the target operation, effectively acting as structured noise. An attacker may observe extra peaks or shifts in power traces that correspond to a peripheral bus transfer rather than the cryptographic algorithm. Noise of this type needs to be statistically removed. In practice, crosstalk can reduce the correlation of a power attack or require the attacker to capture more traces. From a debugging standpoint, crosstalk complicates power-based diagnostics by injecting activity from unrelated signals, potentially leading to misinterpretation unless the engineer isolates the power domain or uses decoupling to suppress the coupling.

5.1.5 Clock Jitter and Trace Misalignment

Clock jitter, the small timing variability of clock edges, and other timing variations are a critical form of noise that affects power trace alignment. In side-channel analysis, repeated traces are often assumed to be time-aligned, so the same operation occurs at the same sample index in each trace. However, real devices exhibit jitter: the clock might not be perfectly periodic due to PLL phase noise, power supply modulation, or intentional clock randomisation. Kocher et al. [62] noted that mismatching of device clocks and sample clocks can cause errors, and uncorrected temporal misalignment of traces introduces a large amount of noise. In other words, even if the power consumption patterns are identical, a slight time shift between traces acts as noise to a differential analysis. Introducing a random clock jitter is a known countermeasure to make attacks harder due to trace misalignment. Almost every side-channel study that deals with real hardware mentions trace alignment issues, either as a challenge to overcome or as a result of countermeasures. The drop in correlation can quantify the effect of the jitter or SNR if not corrected. Even a jitter of a few percent of the clock period can blur the power spikes significantly. For instance, Fledel and Wool [40] showed that with a device implementing jitter up to ± 75 clock cycles, a standard correlation attack fails unless a special sliding-window correlation is used. With smaller natural jitter, the misalignment might be on the order of one sample period or less, but it still contributes severely to desynchronisation, meaning an attack may need dozens of times more traces if no alignment is done. In practical terms, clock jitter and timing variability are very frequently cited, and their impact is to horizontally smear the power trace features, requiring either sophisticated alignment algorithms or a huge trace count to average out the misalignment [62]. In broader performance analysis, timing jitter can similarly complicate aligning power profiles to code execution timelines, especially in multicore systems where scheduling adds jitter on top of clock instability.

5.1.6 Temperature Variations

Temperature variation of the device or environment can alter the power consumption behaviour and introduce low-frequency drift in power traces. As temperature rises, transistor characteristics change, which can change the overall current draw for the same operation. If the temperature is not controlled, traces taken at different times may have systematic offsets or differences unrelated to the data. Historically, many side-channel experiments are done at room temperature with short acquisition times, so temperature was not always explicitly discussed. However, recent research shows it can be significant. Glamocanin et al. found that with the ambient temperature increasing, the success rate of the Corre-

lation Power Analysis (CPA) attack decreases [48], due to temperature-induced changes in the sensor measurements. The authors note that ignoring temperature effects can lead to overly optimistic results and recommend following guidelines to stabilise or account for temperature. Even subtle architectural or measurement variations between devices can lead to observable differences in leakage [14], so conducting side-channel evaluations in a single run or under constant conditions is recommended to prevent false results due to varying device temperatures. Temperature effects are primarily mentioned in specialised papers. Temperature’s impact is often a slow drift or scaling of the power trace. For example, as a chip heats up, its idle power might increase, and its dynamic power might slightly change, leading to a baseline shift or different amplitude for power peaks. This can reduce correlation coefficients in an attack if traces from cooler and hotter conditions are mixed. The CPA success drop reported qualitatively in [48] indicates that more traces or re-calibration were needed at higher temperatures. The temperature can introduce misleading results for engineers conducting power-based debugging or energy profiling. For example, a long-running test may show increasing power consumption caused by device heating rather than changes in the software. Experiments often allow a warm-up period, or temperature chambers are used for consistent conditions to mitigate this.

5.1.7 Operating System and Scheduling Noise

When the target device is running a full Operating System (OS) or multitasking scheduler, the power trace can include many incidental activities unrelated to the intended workload. OS noise refers to interference from OS background processes, interrupts, context switches, and other scheduling events that occur unpredictably from the perspective of the user’s program. In high-performance computing, OS daemons can lead to noise or jitter, which leads to performance variability [88]. In power traces, these OS-induced events manifest as extra spikes or dips whenever the OS schedules a different task or handles an interrupt. For example, if an encryption routine runs on a Linux-based system, a context switch or an interrupt servicing the network stack will cause a burst of CPU activity unrelated to the encryption, effectively adding noise to the trace of the crypto operation. Such OS and scheduling-induced noise is not commonly addressed in classic side-channel literature on microcontrollers because those often run on bare metal or a simple loop without multitasking. However, researchers have noted the need to account for scheduling variability in real-world systems like smartphones or Linux IoT devices. Some recent side-channel experiments on Linux-based devices filter out traces where obvious context switches occur or pin the process to a dedicated core to minimise interference. The frequency of citing OS noise in side-channel papers is relatively low, but it is widely recognised in systems

performance literature. The impact of OS scheduling noise is highly disruptive if present: it can completely ruin trace alignment and add significant power consumption events that dwarf the signal of interest. Unlike thermal noise, which tends to be consistent and statistically manageable, OS-induced noise, such as daemon activity, scheduler interrupts, or kernel-level services, can introduce non-reproducible power fluctuations that severely degrade side-channel analysis. These events may obscure or distort the signal, often requiring traces to be discarded or segmented. To mitigate such effects, engineers can synchronise trace acquisition with the OS to avoid active system processes, employ real-time operating system features to reduce jitter, disable unnecessary background services, bind processes to dedicated Central Processing Unit (CPU) cores, and use high-resolution timers for precise alignment [88]. In distributed systems, global clock synchronisation and pre-acquisition micro benchmarking further help reduce the impact of system-level noise on measurement fidelity. In summary, OS/scheduling noise is highly cited in broader contexts and increasingly acknowledged in side-channel evaluations on complex devices. Its typical effect is to introduce run-to-run variability and occasional large outlier spikes, which can misalign traces and reduce correlation unless mitigated.

5.1.8 Algorithmic and Data-Dependent Noise

Beyond external interference, power traces can also be noisy due to the device’s own concurrent operations that are algorithmically unrelated to the secret of interest. In side-channel analysis, this is often called algorithmic noise or modelling noise. It refers to variations in power consumption stemming from parts of the device or program that are not accounted for in the simple power model used by the attacker [15]. For example, when measuring a cryptographic routine on a microcontroller, the CPU may also be updating a loop counter, performing memory accesses, or toggling status LEDs. These consume power but do not depend on the secret key, thus acting as noise from the attacker’s perspective. Similarly, if a cryptographic algorithm has operations that do not involve the key byte being attacked, those operations contribute power consumption that dilutes the key-dependent signal [44]. Unlike pure random noise, algorithmic noise can be systematic and not reducible by averaging over identical inputs – it might only average out if the unrelated operations vary independently of the secret input. Algorithmic noise is well known but rarely mentioned in general Differential Power Analysis (DPA) and CPA studies. Algorithmic or model noise is a significant factor that requires techniques like profiling or multivariate analysis. The impact is a reduced correlation or SNR because the power trace is a sum of the secret-dependent signal and other activity. For instance, in an Advanced Encryption Standard (AES) trace, the rounds of encryption contribute to key-dependent leakage, but loop

overhead and instruction fetching contribute to a baseline that can be as large or larger. The correlation to the correct key is very low, near zero, without isolating the key-related consumption [44]. In contrast, the correlation jumps significantly if the unrelated code is quiescent. Algorithmic noise is commonly mitigated by designing tests where only the algorithm of interest runs or by using profiling attacks and machine learning that can learn the power pattern of the unrelated operations [15].

5.1.9 ADC Quantization and Measurement Instrument Noise

The final category is the noise introduced by the measurement instrument, including the ADC quantisation error, sampling jitter, and amplifier noise. Whenever a power trace is digitised using an oscilloscope or a dedicated ADC, the measurement has finite resolution and bandwidth. Quantisation noise arises because the ADC converts an analogue voltage into a discrete digital value – any analogue detail smaller than one Least-Significant Bit (LSB) is lost and appears as a quantisation error. For an ideal ADC with resolution N bits, the theoretical quantisation noise yields an SNR of about $(1.763 + 6.02b)dB$ [110]. This limits trace quality hard: if the side-channel signal is very small relative to the total range, a low-resolution ADC will bury it in quantisation error. Kocher et al. [62] identify quantisation errors arising from mismatches between the device’s internal clock and the sampling clock as a source of measurement error. Suppose the ADC sampling is not perfectly synchronised with the device’s clock. In that case, it not only causes timing misalignment but also effectively samples at arbitrary points in the waveform, leading to additional quantisation uncertainty. In side-channel experiments, using a high-quality oscilloscope or synchronous sampling technique is crucial [101]. Aside from quantisation, the measurement device introduces thermal noise of its own, such as ADC aperture jitter or amplifier noise. Measurement noise is acknowledged in essentially every experimental paper, though often implicitly. Its impact adds additional noise floor on top of the device’s noise. When a digital oscilloscope samples a signal, its ADC divides the input into 2^N discrete levels, where N is the number of bits of resolution. For instance, a 2-bit ADC provides only four levels (00, 01, 10, 11), while a 4-bit ADC yields 16 levels, closer approximating the original analogue waveform [80]. Thus, if the voltage difference produced by an operation is smaller than the voltage span of a single quantisation level, that difference may be lost during digitisation. In such cases, if a power difference amounts to only a small fraction of the overall signal range, it may be indistinguishable without oversampling to improve the resolution effectively. Narrow bandwidth filtering can also help by reducing high-frequency ADC noise. From a frequency of citation perspective, ADC and instrument noise are routinely cited in technical notes and a fair number of academic works, especially

those focusing on measurement techniques or the need for low-noise amplifiers. The typical impact is a reduced observable leakage if the instrument is not up to par. Using a 12-bit [ADC](#) versus an 8-bit [ADC](#) could drop the required number of power traces by an order of magnitude because the higher resolution preserved more [SNR](#) of the leakage signal [9]. Instrument noise is one reason why attacks on real devices sometimes fail with a basic setup – the limiting noise may not be the device but the measurement apparatus.

5.1.10 Summary

Each of these noise factors can interplay. For instance, clock jitter and [OS](#) interrupts together could misalign and add extra noise to traces, compounding the difficulty. Table [5.1](#) summarises noise types and their typical impact on trace quality. Achieving high-quality power traces requires careful management of all these interference sources. Previous research notes that minimising noise is key to successful power analysis. By understanding the spectrum of noise and interference factors detailed above, analysts can better design experiments and evaluate the actual leakage in the presence of real-world disturbances.

Noise Type	Citations	Occurrence Frequency	Typical Impact on Trace Quality
Thermal/Device Noise	[14], [15], [62], [110], [19], [48], [55], [78], [44], [105]	Almost universal or baseline assumption in all studies	Adds Gaussian baseline noise floor, lowers SNR and Pearson correlation; requires averaging to overcome.
EMI / Environmental Noise	[62], [74], [20], [19], [55], [14], [44], [105]	Very common in measurement-focused papers and guides	Raises noise floor with external signals such as RF spikes or hum, can obscure small power variations; often necessitates shielding or filtering.
Power Supply Noise	[15], [74], [78]	Common in hardware security and Integrated Circuit (IC) design works	Superimposes supply voltage fluctuations on traces, reducing distinguishability of leakage. More traces or local measurement points are needed.
Crosstalk	[91], [20], [19], [53]	Occasionally cited (recent multi-core or SoC studies)	Injects unrelated power spikes or shifts into traces via inductive coupling; effectively acts as structured noise from other activity, lowering correlation with the target operation.
Clock Jitter / Trace Misalignment	[62], [74], [81], [74], [20], [19], [78], [14], [53], [44]	Very frequently cited (major issue or countermeasure in many papers)	Introduces temporal offsets between traces, blurring power peaks. Requires re-alignment algorithms or increases noise if uncorrected.
Temperature Variation	[15], [19], [48], [55], [14], [14], [53]	Infrequent but well known; noted in recent works and evaluation guidelines	Causes slow drift in trace baseline and amplitude. Higher temps can reduce attack success. Typically, this results in lower SNR and biased power measurements if not stabilised.
OS and Scheduling Noise	[88], [16], [44]	Rare in classic SCA papers and common in systems/HPC literature	Power spikes from OS tasks can misalign traces and add non-repeatable patterns, often rendering some traces unusable or requiring many more traces to average out randomness.
Algorithmic/Parallel Activity Noise	[15], [78], [14], [88], [16], [44], [105]	Rare in classic SCA papers and common in side-channel modelling discussions	Adds constant or input-dependent power consumption from unrelated operations. Does not average out easily if deterministic. Lowers maximum correlation. Often demands profile attacks or isolating the operation of interest.
ADC Quantization / Instrument Noise	[15], [74], [110], [91], [48], [53], [105]	Frequently noted in experimental setups	Limits measurement resolution and adds additional noise. Reduces SNR by a fixed floor. Higher resolution or oversampling improves trace quality significantly.

Table 5.1: Noise/interference types in power analysis, frequency in literature, and impact on trace quality

5.1.11 Risk Assessment

This subsection introduces a quantitative methodology to assess the relative risk of different noise sources based on their frequency and impact on trace quality. The resulting scores help prioritise mitigation strategies by identifying the most critical sources of interference in side-channel measurements.

Quantitative Methodology

We adopt a quantitative risk estimation methodology to assess the potential threat of different noise and interference sources on side-channel trace quality. The *Risk Level* is computed based on two main factors: the frequency of occurrence and the severity of impact on trace quality. Both factors are calculated based on summarised information from Table 5.1. Risk evaluation facilitates the prioritisation of mitigation strategies in real-world signal acquisition setups.

Each qualitative label for *Occurrence Frequency* and *Impact on Trace Quality* is assigned a numerical score, as shown in Table 5.2.

Qualitative Level	Score
Very Low	1
Low	2
Moderate	3
Moderate/High	3.5
High	4
Very High	5

Table 5.2: Quantitative scoring for qualitative frequency and impact levels.

The risk score is computed using the following formula:

$$\text{Risk Score} = \text{Frequency Score} \times \text{Impact Score} \quad (5.1)$$

Based on the final score, the overall *Risk Level* is categorised with labels as given by Table 5.3.

Risk Score Range	Risk Level
1–6	Low
7–12	Moderate
13–20	High
21–25	Critical

Table 5.3: Risk level categorisation based on total score.

Results

The computed risk levels for each noise/interference source using a defined risk assessment methodology are shown in Table 5.4. This allows practitioners to understand each noise type’s criticality and prioritise countermeasures accordingly.

Noise/Interference Type	Frequency Score	Impact Score	Risk Score (F×I)	Risk Level
Clock Jitter & Trace Misalignment	5	5.0	25.0	Critical
EMI	4	4.0	16.0	High
Power Supply Noise	4	4.0	16.0	High
Thermal/Device Noise	5	3.0	15.0	High
OS & Scheduling Noise	3	5.0	15.0	High
ADC Quantization & Instrument Noise	4	3.5	14.0	High
Algorithmic Noise	4	3.5	14.0	High
Temperature Variation	3	3.5	10.5	Moderate
Crosstalk (Signal Coupling)	3	3.5	10.5	Moderate

Table 5.4: Calculated risk scores and levels for different noise and interference sources.

5.2 Systematisation of Knowledge: Mitigation Strategies based on Noise Type

Effective side-channel analysis requires selecting appropriate strategies to mitigate noise impact on trace quality and signal extraction. This section systematically categorises and evaluates noise mitigation methods according to the specific types of noise they address, focusing primarily on post-processing and software-based techniques. Each method is analysed in terms of its effectiveness, applicability, and complexity. Table 5.5 summarises the results of the performed systematisation and supports informed decision-making when designing or adapting signal processing pipelines for embedded system security evaluations.

5.2.1 Noise Mitigation Methodologies

Power side-channel measurements are susceptible to various sources of noise and interference. Below, we organise common algorithms and methodologies for improving power trace quality based on mitigated noise type, focusing on software and post-processing techniques and noting proven hardware countermeasures where relevant. Each method is described in terms of its purpose, effectiveness, popularity in the literature, and practical considerations.

Thermal Noise Thermal noise refers to the intrinsic random fluctuations in the device. This noise is typically high-frequency and uncorrelated with the signal, reducing the clarity of each power trace. Common mitigation techniques include:

- **Peak-to-Peak Trace Averaging:** Repeating the same operation many times and averaging the recorded waveforms can significantly reduce random noise since uncorrelated noise cancels out over multiple samples [44]. Averaging N identical traces improves SNR by roughly $\sqrt{N}$. This is a highly effective technique, proven with time. In a recent study, it was even shown that averaging outperforms other tested methodologies like learned autoencoder [107].
- **Low-Pass Filtering (LPF):** Since thermal noise often manifests at higher frequencies, a digital low-pass filter can remove high-frequency components while preserving the slower power consumption variations. Simple filters like moving average or Butterworth LPF are widely used to smooth traces. The purpose is to average out fast noise spikes and retain the bulk of the side-channel signal, usually concentrated around clock frequency or its harmonics. This is moderately effective if the signal bandwidth is known and much lower than the noise bandwidth. However, improper cutoff can also attenuate the relevant signals. Low-pass filtering is widespread in SCA preprocessing [95]. It's straightforward to apply with low computational cost but offers limited improvement if important signal components lie in the filtered band [39].
- **Adaptive Filters (Wiener/Kalman):** More advanced filtering can be applied with statistical knowledge of signal and noise. A Wiener filter, for instance, optimally balances signal and noise if their spectra are known [23]; Kalman filters can model the power trace as a process and iteratively reduce noise [103]. These filters aim to maximise SNR by adapting to the characteristics of the trace. They have shown success in the literature but are less popular than simple averaging or static filters due to the complexity of modelling the system. Effectiveness can be high when the

model is accurate, but mistuning can lead to filtering out part of the actual leakage. Their computational cost is higher, and they are only as valid as the assumptions made about the signal.

- **Wavelet Denoising:** Wavelet transforms techniques remove noise by decomposing the trace into multi-scale components and thresholding the small coefficients representing noise [29, 97]. The idea is that Gaussian noise yields small-amplitude wavelet coefficients, which can be truncated without losing significant signal [99]. Wavelet-based denoising has been successfully applied to side-channel traces to improve SNR [22]. It can better preserve sharp features like instant power spikes than a simple low-pass filter. While not as ubiquitous as basic filtering, several studies have explored wavelet pre-processing to enhance DPA/CPA attacks. Wavelet denoising often improves attack success with fewer traces [102], though it requires choosing appropriate wavelet families and thresholds. MATLAB and specialised Python libraries provide wavelet denoising functions, making implementation feasible.
- **Singular Spectrum Analysis (SSA):** SSA is a technique that applies a singular value decomposition on time series, treating the trace as a trajectory matrix, to separate signal components from noise [116]. Del Pozo et al. introduced SSA for power trace filtering and demonstrated significant gains in attack efficiency on unprotected and masked implementations [71]. The method reconstructs the trace using only the leading principal components, assumed to correspond to true leakage, and discards components associated with noise. SSA can yield substantial SNR improvements, 5 or more in some cases, [95], and it became fairly popular in academic research. However, it involves hyperparameters (window length, component selection) that are somewhat ad-hoc [67], and if chosen poorly, the benefits diminish. SSA is computationally heavier than a simple filter since it involves an Singular Value Decomposition (SVD), but it is still practical for small trace lengths.
- **Independent Component Analysis (ICA):** ICA is a blind source separation method that can treat each trace as a mixture of independent sources and attempts to separate them [65]. Maghrebi and Prouff [67] report that ICA drastically outperforms simpler averaging and Wiener filtering. ICA's purpose is to statistically isolate the true signal component without knowing its shape, leveraging non-Gaussianity – noise is often Gaussian, whereas leakage has structure. While promising in effectiveness, ICA is relatively complex and not yet a standard practice. It requires multiple traces and works best when the noise and signal are independent. Popularity is growing in the literature as a preprocessing step, but it's less common in simple tools. Computationally, FastICA algorithms [84] can extract components in a reasonable time, but

one must be careful to interpret which component corresponds to the secret leakage.

- **High-Order Statistical Filtering:** Le, Thanh-Ha, et al. [64] introduced a noise reduction using fourth-order cumulants. These aim to exploit the fact that Gaussian noise has zero cumulants beyond the second order, so one can design filters highlighting non-Gaussian, hopefully, signal parts. While interesting, these are niche and require deep signal processing expertise. Similar techniques are not widely used compared to the above methods, and their effectiveness can vary. They might help in scenarios where noise is Gaussian and leakage is not, but they can be less robust if assumptions don't hold.
- **Machine Learning (ML) Denoisers:** As a newer approach, machine learning models (autoencoders, denoising neural nets) have been attempted to learn a mapping from noisy to clean traces. The idea is to use a training set to teach a model the structure of noise and signal. In practice, results have been mixed. For example, an autoencoder was trained to denoise **Field Programmable Gate Arrays (FPGA)** power traces, but its denoised output did not significantly outperform raw traces and was inferior to simple averaging in the attack evaluation [107]. This suggests that while ML can learn some features, traditional methods are still more robust for noise. ML denoisers are not yet proven in the field. They carry the overhead of needing training data and the risk of overfitting. ML models can potentially handle complex noise patterns that are hard to model analytically and can succeed where classical methods struggle, though the result might be uninterpretable [24].

While this section concentrates on software methods, one should not forget hardware strategies to minimise thermal noise at capture. Using low-noise amplifiers and **ADCs**, proper grounding, and shielding of measurement cables, all help reduce the random noise floors. Sampling faster than needed and then digitally downsampling can also average out some thermal noise. Cooling the device or measurement front-end can lower thermal noise, but this is rarely done specifically for **SCA** as the improvement is usually marginal compared to averaging traces. In practice, ensuring a high-quality measurement setup (good probe contact, short wires, low electromagnetic pickup) will set a lower baseline noise, making all the above algorithms even more effective.

Electromagnetic Interference (EMI) **EMI** refers to external or environmental signals coupling into the power measurement. This could be ambient mains hum (50/60 Hz), radio-frequency interference from nearby electronics, or cross-coupling from the device's

clock and I/O lines into the measurement apparatus. EMI can manifest as specific frequency components or bursts overlaying the trace, obscuring the actual power signal. Key mitigation approaches include:

- **Notch and Band-Stop Filtering:** If interference at a known frequency is present, it is useful to apply a notch filter to remove that narrow band. For example, a 50 Hz interference can be filtered with a high-Q notch without affecting the MHz-range crypto leakage. Similarly, band-stop filters can eliminate known unwanted bands. This technique directly targets EMI frequencies, greatly reducing their amplitude in the trace [50]. It's as effective as the interference is narrow. However, narrowband filtering is only a partial fix – broadband or unknown EMI requires broader measures. Nonetheless, digital and analogue filters are common in labs, especially to remove mains noise or other continuous wave interference. One must ensure the side-channel signal doesn't reside in the filtered band.
- **Band-Pass Filtering around Clock Harmonics:** As an alternative to removing specific noise bands, one can focus on the frequency band where the leakages predominantly lie. Power side-channel signals often concentrate around the device's clock frequency or its harmonics [98]. Using a band-pass filter tuned to that range, you keep the relevant signal and reject frequencies far from it. Prior research has shown that most leakage energy in a simple device might be confined to a band, and filtering accordingly improves SNR [95]. This method requires some analysis, such as taking an FFT of sample traces to identify peaks. The effectiveness is scenario-dependent – if the leakage is broadband, a band-pass won't capture it all. Band-pass filtering is widely reported in the literature and is easy to apply once the band is known.

A range of hardware-based strategies exist to mitigate EMI in side-channel measurement setups. One of the most fundamental techniques is electromagnetic shielding, which reduces EMI coupling to the measurement. Attacking setups often employ a Faraday cage or a metal shield around the device and sensor to block external electromagnetic noise. This doesn't remove noise in software but prevents it from appearing in the traces. Differential probes or amplifiers for current sensing can help cancel out common-mode EMI. A differential measurement, such as across a shunt resistor, measures only the difference between two nodes, ideally subtracting noise that is common to both. This provides hardware noise rejection. It is widely used in power analysis setups. Good quality amplifiers have a high common-mode rejection ratio to ignore environmental pickup. For example, measuring the voltage drop across a shunt with a differential amplifier means any EMI that

couples equally into both leads will be rejected. Sometimes, interference comes from specific sources on the board (neighbouring circuits). One can reduce coupling by physically repositioning the measurement probe or using smaller loop areas. For instance, twisting the measurement leads or using coaxial cables ensures any external EMI induces equal and opposite currents that cancel out. Additionally, careful probe placement can emphasise the target signal over others [5].

In summary, for EMI, hardware fixes (shielding, differential probing) are often the first line of defence and are widely adopted because they improve all traces universally. Software filtering is then applied to the remaining interference. It's worth noting that some target devices intentionally inject EMI or noise as a countermeasure.

Power Supply Noise Power supply noise refers to fluctuations and interference from the device's power source and delivery network. This includes voltage droops, ground bounce, or switching regulator ripple that superimpose on the measured power consumption. In traces, this might appear as low-frequency oscillations or spikes unrelated to the cryptographic operations. Techniques and countermeasures include:

- **High-Pass Filtering / Baseline Removal:** Many supply noise, such as DC level shifts or regulator oscillations, is low-frequency [61]. Applying a high-pass filter to each trace can remove the slowly varying component, essentially detrending the trace. This lets the fast, data-dependent power spikes stand out above the baseline. Using a high-pass filter when a device's internal regulator causes large low-frequency noise might mitigate such noise. The purpose is to cut out the baseline wander from the supply and keep only the rapid transients of interest. High-pass filtering effectively eliminates supply drift and regulator oscillation, which greatly improves the clarity of where operations occur. It must be used carefully – too high a cutoff might also remove the bulk of the signal (if the device operations are slow).

Most techniques for power supply noise rely on hardware measurements. For instance, measuring the voltage drop across a small shunt resistor on the supply line rather than measuring an absolute voltage inherently subtracts the supply's DC component and much of its slow variation. This technique not only helps with EMI but also targets supply noise. One source of noise is the target's power regulator, especially switching regulators, which introduce periodic noise. A widely used countermeasure is to power the target with a stable external supply or force the internal regulator into a bypass mode [56]. For instance, some microcontrollers allow feeding a slightly higher voltage so that the internal

[Low-Dropout Regulator \(LDO\)](#) turns off, thereby eliminating its noise contribution [56]. Adding decoupling capacitors near the measurement point can smooth out some supply noise at the expense of smoothing the signal. In an attack, one might remove certain capacitors to increase signal magnitude, but to mitigate noise, having a clean supply with proper filtering can isolate external fluctuations.

In summary, mitigating power supply noise is often about measuring correctly and filtering out low-frequency components. These steps are widely adopted because they directly increase trace quality.

Clock Jitter and Desynchronization Clock jitter causes misalignment of power traces. Essentially, the timing of operations varies between traces, blurring the features when traces are compared or averaged. Even slight jitter or phase drift can severely reduce the effectiveness of straightforward averaging or correlation analysis, making it a significant noise source in side-channel attacks. Techniques to handle jitter and misalignment include:

- **Trace Alignment via Cross-Correlation:** One of the most common post-processing steps is aligning traces to a reference point [68]. A simple approach is to use cross-correlation: take a segment of one trace or an average trace as a reference pattern and slide another trace across it, finding the offset that maximises correlation [67]. The major features line up by shifting each trace accordingly despite the clock jitter. This addresses jitter noise by removing time offsets, turning a fuzzy average into a sharp one.
- **Elastic Alignment ([Dynamic Time Warping \(DTW\)](#)):** When jitter or desynchronisation is not just a simple shift but involves non-linear stretching or compression of the time axis, elastic alignment is a powerful technique [114]. [DTW](#) essentially finds an optimal match between two time-series by allowing one to warp in time to align with the other. In the side channel, this was introduced to deal with random process interrupts or random delays inserted as a countermeasure. Elastic alignment can align traces even if one trace runs slightly faster or slower in parts than another. [DTW](#) alignment is well known and has been applied in several papers. It is less common in public tools due to its complexity and computational cost, but it's increasingly accessible.
- **Peak/Feature Alignment:** A simpler alignment variant is to detect a particular peak or edge in each trace and align based on that index. This could be done by thresholding or edge detection on the signal. It's effective if a clearly identifiable feature

exists in every trace. For instance, if every encryption triggers a notable power spike at the start, aligning all traces so that the spike is at the same sample index removes the jitter. Good for consistent features, but if noise obscures the feature in some traces, those might misalign. It is very fast and easy but not robust to all forms of jitter.

Rather than purely post-process, another strategy is to eliminate jitter at the time of capture. If possible, using the same clock for the device and the oscilloscope (or at least using a phase-locked sampling clock) ensures each trace is sampled at identical points in the cycle. In practice, it does not eliminate jitter to the full extent. The alignment quality can be measured by checking the traces' variance or the correlation spike after alignment. Poor alignment means the power of averaging is lost, whereas good alignment recovers the full leakage signal.

OS and Multi-Process Noise When the target device is a complex system running an operating system or other concurrent processes, the power trace can include unrelated activity. This OS noise might include context switches, interrupts, background tasks, dynamic frequency scaling effects, etc. Unlike simple electronic noise, this interference is structured but unpredictable, often appearing as extra spikes or differing baselines in power traces that are not correlated with the cryptographic algorithm's operation. Mitigating this type of noise is challenging, but several approaches are used:

- **Trace Classification and Outlier Removal:** If some traces contain obvious interruptions, such as out-of-place spikes, an analyst might detect and discard those traces. This could be done manually by dropping any trace that deviates beyond a certain power level at an unexpected time or via clustering. One can focus on a cleaner subset by removing outliers or grouping traces by patterns. Removing blatantly noisy traces can improve analysis, as those traces would otherwise obscure averages. However, if interruptions are frequent, you can't just drop too many traces without losing data.
- A brute-force way to deal with random process noise is to collect a large number of traces so that the cryptographic signal can be statistically extracted. Statistical techniques average out unrelated noise by grouping traces by data values. The idea is that OS noise is random concerning the secret data, so given enough traces, the attacker can still isolate the minute correlations. [DPA](#) was famously shown to work even against complex and noisy systems given sufficient traces [62]. It is effective in theory as the law of large numbers guarantees that deterministic data-dependent

effects will emerge from random background noise if you can gather enough samples. In practice, this might mean tens or hundreds of thousands of traces for an attack that usually needs only a few hundred on a quiet system.

- **Synchronous Triggers and Root-of-Trust Execution:** If the attacker has some control, a common tactic is to minimise OS interference by performing the cryptographic operation in as isolated a context as possible. For example, on a Linux system, one might run the encryption in a real-time process or even in a bootloader where the OS isn't multitasking yet. In embedded systems, sometimes cryptographic routines are run in an interrupt context to make them more deterministic. By aligning the measurement trigger to just the operation of interest, we effectively window out the rest of the multitasking activity.
- **Signal Processing for Periodic Background Noise:** Some OS-related noise might be periodic like an OS scheduler ticks every 1 ms or periodic radio communications in a device. If such patterns are known, they can be filtered with the filters discussed above or subtracted through an average interrupt event waveform. Another scenario is that power management may cause the CPU frequency to change, which skews the trace. If the scaling behaviour is understood, one could segment traces by frequency or normalise them in parts. This is very situational. Removing a specific periodic noise can help if that noise is dominant. If background events are more sporadic, this doesn't apply.
- **Profiling and Machine Learning:** Advanced techniques treat the side-channel as a combination of signals and attempt to separate them via learning. For example, one could train a machine learning model to recognise the signature of the crypto operations amidst the noisy trace. Its effectiveness is potentially very high as ML can surpass human-designed filters when noise is very complex. It's been called a game changer for side-channel analysis on noisy, large devices [35]. However, it often requires an extensive training set and might need labelled data.
- **Increasing Signal:** One indirect method is to amplify the power signature of the operation of interest so that OS noise is relatively smaller. For example, if attacking an AES software implementation on a busy CPU, one could run the AES repeatedly in a tight loop or on multiple cores simultaneously if possible so that the power usage from AES dominates [42]. Essentially, by making the cryptographic operation account for a larger share of total power, the noise from other processes becomes less significant proportionally.

In summary, OS and multi-process noise are some of the toughest challenges because they are complex signals intermixed with the signal of interest. Attackers cope by averaging it out with many traces, isolating it through selective filtering, or learning around it with ML techniques.

Algorithmic Noise Algorithmic noise refers to variations in the power trace caused by parts of the algorithm or device operation that are not directly related to the secret data one is trying to extract. In other words, it's the structured but key-independent portion of the trace. Examples: the power consumption differences due to loop counters, fixed sequence of operations, or concurrent non-crypto computations. Unlike random noise, algorithmic noise is often deterministic given specific inputs or states, but it can mask the valuable leakage. Attackers employ several strategies to reduce algorithmic noise:

- **Differential Analysis (DPA/CPA)**: The fundamental rationale of differential side-channel analysis is to cancel out algorithmic noise by comparing traces for different data inputs. Constant power consumption is eliminated by subtracting or averaging groups of traces. For instance, in classic DPA, traces are partitioned by a guessed key-bit outcome. The subsequent difference in means removes the baseline consumption, which is the same across the two sets, and highlights only the data-dependent differences.
- **Mean or Reference Trace Subtraction**: A simple preprocessing is to compute an average trace over all acquisitions and subtract this average from each trace. The average trace presumably captures the algorithmic behaviour that is common to all. By subtracting it, each trace is left with only the deviations from the average, which often correspond to data-dependent variations. This is effectively what some authors call zero-mean traces.
- **ICA for Algorithmic Separation**: Just as ICA was mentioned for separating generic noise, it can specifically separate independent sources of leakage. Suppose the total power trace is a mixture of the crypto operation leakage and some other concurrent operation leakage. If these are statistically independent, ICA can attempt to extract two separate waveforms. In the context of algorithmic noise, if one part of the power corresponds to an algorithmic deterministic operation and another part is data-dependent, ICA might treat them as independent sources and split them.

Algorithmic noise is often deterministic, given the secret. Examples of algorithmic noise are shuffling with a **Central Processing Unit (PRNG)** seed or inserting dummy rounds

based on a secret pattern. Such noise is incredibly potent because an attacker cannot average it out without knowing the secret pattern because it’s synchronised with the secret, not with their chosen input [35]. When algorithmic noise countermeasures are absent, differential techniques usually handle the natural algorithmic noise well. Such methods as mean subtraction or PCA are often unnecessary if one can do a DPA since DPA inherently removes the static algorithmic footprint. They become more relevant when doing single-trace analysis or template attacks or when the algorithmic noise is so large that it slows down the convergence of DPA. In to subtract or such cases, preprocessing to boost SNR through methods similar to ICA directly addresses the algorithmic noise and can reduce the number of traces needed. In conclusion, algorithmic noise mitigation in the side-channel analysis is about leveraging the structure: using the fact that it is deterministic or common across traces either to subtract it out or to avoid it.

5.2.2 Summary

Effective mitigation of noise and interference in power trace analysis requires careful selection of countermeasures. Table 5.5 summarises recommended mitigation post-processing strategies for various noise types. Thermal and device noise, EMI, power supply disturbances, and algorithmic noise occur frequently, each significantly reducing trace quality. Although varying in occurrence, clock jitter and OS-induced scheduling noise present particularly high risks due to their severe detrimental effects on trace alignment and signal coherence.

Diverse mitigation strategies are available to address these challenges, each targeting specific noise characteristics. Basic statistical techniques, such as trace averaging and DPA, remain highly popular and effective for counteracting random thermal and algorithmic noise by statistically isolating data-dependent signals. Advanced digital filtering techniques, including wavelet denoising, SSA, and ICA, offer significant improvements in SNRs, albeit at increased computational complexity and implementation effort. Clock jitter, a critical source of trace misalignment, requires dedicated alignment methodologies like cross-correlation alignment, elastic alignment using DTW, and synchronous sampling triggered through hardware. Cross-correlation and synchronous sampling are particularly prevalent in practical implementations due to their balance of effectiveness and moderate complexity.

While out of scope for this research, we are noting hardware-based countermeasures used in analysed side-channel related works in Table 5.6.

Overall, selecting optimal mitigation strategies necessitates balancing the trade-offs be-

tween effectiveness, computational complexity, implementation feasibility, and the specific noise characteristics of the measurement scenario. Successful noise mitigation typically combines simpler, broadly applicable techniques with advanced analytical or hardware solutions tailored to particular, high-risk interference sources.

Noise/Interference Type	Occurrence Frequency	Impact	Risk Level	Recommended Mitigation Strategies
Thermal/Device Noise	Very High	Moderate	High	Trace Averaging, Low-Pass Filtering, Adaptive Filtering (Wiener/Kalman), Wavelet Denoising, ICA , SSA
EMI	High	High	High	Shielding, Differential Probing, Band-stop (Notch) Filtering, Band-pass Filtering
Power Supply Noise	High	High	High	Differential Measurement (Shunt), High-pass Filtering, External Stable Power Supply, Decoupling
Clock Jitter & Trace Misalignment	Very High	Very High	Critical	Cross-correlation Alignment, Elastic Alignment (DTW), Peak Alignment, Synchronous Sampling, Triggered Acquisition
OS & Scheduling Noise	Moderate	Very High	High	DPA , Trigger Synchronization, ML Profiling, Outlier Removal
ADC Quantization & Instrument Noise	High	Moderate/High	High	Use high-resolution ADCs, oversampling with averaging, low-noise amplifiers
Algorithmic Noise	High	Moderate/High	High	DPA , Mean Reference Subtraction, PCA , ICA , SSA , Operation Resynchronization

Table 5.5: Impact of various noise and interference sources on trace quality and suggested mitigation strategies.

Mitigation Strategy	Noise Types Effectively Mitigated
Shielding	Electromagnetic Interference (EMI), Crosstalk (Signal Coupling)
Differential Probing	Electromagnetic Interference (EMI), Power Supply Noise, Crosstalk (Signal Coupling), ADC Quantization & Instrument Noise
Differential Measurement (Shunt)	Power Supply Noise, Electromagnetic Interference (EMI)
External Stable Power Supply	Power Supply Noise
Decoupling Capacitors	Power Supply Noise, Electromagnetic Interference (EMI)
Triggered Acquisition	Clock Jitter & Trace Misalignment, OS & Scheduling Noise
Trigger Synchronization	Clock Jitter & Trace Misalignment, OS & Scheduling Noise
Oversampling with Averaging	ADC Quantization & Instrument Noise, Thermal/Device Noise
High-Resolution ADCs	ADC Quantization & Instrument Noise
Low-Noise Amplifiers	ADC Quantization & Instrument Noise, Electromagnetic Interference (EMI), Thermal/Device Noise
Synchronous Sampling (Clock Sharing)	Clock Jitter & Trace Misalignment, ADC Quantization & Instrument Noise
External Temperature Control	Temperature Variation

Table 5.6: Noted hardware countermeasures against different noise sources.

5.2.3 Risk Assessment

This subsection presents a structured evaluation of post-processing mitigation strategies based on their effectiveness, applicability, and implementation complexity. The assessment draws on empirical data and expert insights to support informed countermeasure selection, as Table 5.5 summarises.

Quantitative Methodology

A structured evaluation framework was developed to assess the effectiveness and practicality of various post-processing countermeasures for mitigating noise and interference in

side-channel power traces. The evaluation is grounded in a quantitative risk assessment model incorporating empirical data, documented methodologies, and domain knowledge assessed in Table 5.5.

Each identified noise source was assigned a *risk score*, ranging from 10.5 to 25.0, based on its frequency of occurrence and its impact on side-channel measurement integrity (Table 5.3). These scores serve as weighted indicators of how critical each noise source is to the overall signal degradation.

The effectiveness of each mitigation strategy was quantified using a computed *mitigation score*, defined as the weighted sum of the effectiveness factors assigned to each mitigated noise source. Formally, the mitigation score is given by:

$$\text{Mitigation Score} = \sum (\text{Risk Score of Noise Type} \times \text{Effectiveness Factor}) \quad (5.2)$$

A normalised scale from 0 to 1 was adopted to assign mitigation capability:

- **0.0:** No mitigation effect
- **0.1–0.39:** Limited mitigation (minimal practical benefit)
- **0.4–0.59:** Moderate mitigation (partial reduction, situationally useful)
- **0.6–0.79:** High mitigation (reliable, significant noise suppression)
- **0.8–1.0:** Very high mitigation (eliminates or largely neutralizes noise)

Effectiveness factors represent the degree to which a given method suppresses or neutralises a specific noise type. These were derived from literature benchmarks, experimental observations, and implementation experience. To facilitate interpretation, the average effectiveness of each strategy was computed and mapped to a qualitative level:

- Average ≥ 0.75 : **Very High**
- Average ≥ 0.60 : **High**
- Average ≥ 0.40 : **Moderate**
- Average < 0.40 : **Low**

In addition to technical performance, each strategy was evaluated based on its practical feasibility. The classification is as follows:

- **Low:** Simple to implement with minimal technical or computational overhead.
- **Moderate:** Requires intermediate-level expertise or some parameter tuning.
- **High:** Demands significant technical effort, specialised tooling, or processing time.

To rank the mitigation strategies, a composite *final score* was computed by integrating three key dimensions. The first component, accounting for 30% of the total score, measures the breadth of unique noise sources that a given method is capable of addressing and is referred to as noise coverage. The second component, weighted at 40%, captures the mitigation effectiveness through a normalised mitigation score derived from the weighted sum of effectiveness factors across relevant noise types. The final component, representing the remaining 30%, reflects implementation complexity, where simpler methods receive higher scores due to their lower technical or resource requirements. This multidimensional scoring model enables a balanced evaluation of each countermeasure, considering technical performance and practical applicability. The formula used is:

$$\begin{aligned} \text{Final Score} = & 0.3 \times \text{Normalized Coverage} + 0.4 \times \text{Normalized Mitigation Score} \\ & + 0.3 \times \text{Complexity Score}, \end{aligned}$$

where the complexity score for each method was assigned based on implementation difficulty, with lower complexity receiving higher scores. Specifically, methods classified as having low implementation complexity were assigned a score of 1.0, those with moderate complexity received a score of 0.55, and highly complex methods were given a score of 0.1.

This composite scoring methodology ensures that each mitigation strategy is evaluated on its technical merit, practical usability, and breadth of applicability. The resulting ranking provides a balanced and rigorous prioritisation for side-channel analysis countermeasures.

Results

Table 5.7 presents mitigation strategies' final ranking based on their effectiveness in addressing key noise sources, weighted mitigation score, and implementation complexity. The

composite scoring approach ensures that both technical impact and practical feasibility are considered in the evaluation.

Among all evaluated techniques, *Trace Averaging* emerged as the top-performing strategy with a final score of 0.7590. This is attributed to its broad applicability across four critical noise types and its low implementation complexity, making it both effective and practical. *SSA* also performed exceptionally well, mitigating five distinct noise types with a final score of 0.7514, and offering a balanced trade-off between effectiveness and moderate implementation complexity. *ICA* followed closely, achieving the highest mitigation score overall (62.25) due to its strong performance across six noise sources, including high-risk categories such as *EMI* and algorithmic noise. Despite its high complexity, *ICA* scored third in the final ranking with a score of 0.7300. *ML-Based Profiling* also ranked highly, demonstrating strong mitigation across six noise types. However, its higher implementation complexity reduced its final score to 0.6914. Notably, *DPA/CPA* achieved a competitive position, benefiting from moderate-to-high effectiveness and targeting five distinct noise types with only moderate complexity, resulting in a score of 0.6785.

In the mid-tier rankings, signal processing techniques such as *Low-Pass Filtering*, *Band-Pass Filtering*, and *Reference Trace Subtraction* performed consistently. These methods offer moderate mitigation and are favourable for their simplicity, contributing positively to their overall scores.

Lower-ranked techniques, such as *Elastic Alignment*, *Peak/Feature Alignment*, and *Operation Resynchronization*, typically exhibited high mitigation in narrow scenarios but lacked broad noise coverage or incurred higher complexity. Similarly, specialised methods like *Band-Stop Filtering* were highly effective in specific cases but addressed only a single noise type, thus limiting their contribution to overall robustness.

In conclusion, the final scores highlight a trade-off between mitigation breadth, technical strength, and practicality. Top-ranking methods offer a favourable balance across all criteria, whereas more specialised techniques may still be valuable in targeted applications where specific noise sources dominate.

Countermeasure	Targeted Noise/Interference Types	Noise Num	Risk Score Sum	Effectiveness Level	Mitigation Score	Implementation Complexity	Final Score
Trace Averaging	Thermal/Device Noise, ADC Quantization & Instrument Noise, OS & Scheduling Noise, EMI	4	60.0	High	40.30	Low	0.7590
SSA	Thermal/Device Noise, Algorithmic Noise, OS & Scheduling Noise, EMI, ADC Quantization & Instrument Noise	5	74.0	High	52.35	Moderate	0.7514
ICA	Thermal/Device Noise, Algorithmic Noise, EMI, Crosstalk, Power Supply Noise, ADC Quantization & Instrument Noise	6	85.5	High	62.25	High	0.7300
ML-Based Profiling	OS & Scheduling Noise, Algorithmic Noise, Thermal/Device Noise, EMI, ADC Quantization & Instrument Noise, Crosstalk	6	84.5	High	56.25	High	0.6914
DPA / CPA	Algorithmic Noise, OS & Scheduling Noise, Thermal/Device Noise, ADC Quantization & Instrument Noise, EMI	5	44.0	Moderate	41.00	Moderate	0.6785
Low-Pass Filtering	Thermal/Device Noise, ADC Quantization & Instrument Noise, EMI	3	45.0	Moderate	25.30	Low	0.6126
Reference Trace Subtraction	Algorithmic Noise, OS & Scheduling Noise, Temperature Variation	3	39.5	Moderate	22.55	Low	0.5949
Band-Pass Filtering	EMI, Power Supply Noise, OS & Scheduling Noise	3	47.0	Moderate	22.00	Low	0.5914
High-Pass Filtering	Power Supply Noise, Temperature Variation	2	26.5	High	19.10	Low	0.5227
Adaptive Filtering	Thermal/Device Noise, ADC Quantization & Instrument Noise, EMI	3	45.0	High	32.10	Moderate	0.5213
Wavelet Denoising	Thermal/Device Noise, ADC Quantization & Instrument Noise, EMI	3	45.0	High	29.05	Moderate	0.5017
Band-Stop Filtering	EMI	1	16.0	Very High	14.40	Low	0.4425
Cross-Correlation Alignment	Clock Jitter & Trace Misalignment, OS & Scheduling Noise	2	40.0	High	27.50	Moderate	0.4417
PCA	Algorithmic Noise, OS & Scheduling Noise	2	29.0	High	17.30	Moderate	0.3762
Trace Classification	OS & Scheduling Noise, EMI	2	31.0	Moderate	13.90	Moderate	0.3543
Elastic Alignment	Clock Jitter & Trace Misalignment, OS & Scheduling Noise	2	40.0	Very High	31.50	High	0.3324
Peak/Feature Alignment	Clock Jitter & Trace Misalignment	1	25.0	High	15.00	Moderate	0.3114
Operation Resynchronization	Algorithmic Noise	1	14.0	Very High	12.60	Moderate	0.2960

Table 5.7: Final ranking of countermeasures based on number of noise types mitigated, effectiveness, mitigation score, and implementation complexity.

5.3 Systematisation of Knowledge: Signal Quality Metrics

Having enough samples per command is crucial to reliably analysing and interpreting power side-channel signals and performing successful classification or differentiation. Thus, we need to identify an appropriate metric m that encapsulates the clarity and informativeness of the observed traces. Such a metric should ideally capture both the instantaneous quality of the signal and its evolution over time, thereby facilitating adaptive decision-making regarding the required number of samples during trace collection.

Papagiannopoulos et al. [83] outline a range of established metrics used in SCA, including SNR, correlation, Mutual Information (MI), and the t -statistic, each offering unique perspectives on signal quality and leakage detectability. In this section, we extend this range by considering additional metrics such as total signal power and Central Limit Theorem (CLT), evaluating their relevance and effectiveness for signal clarity estimation in the context of SCA quality assessment. Additionally, we propose to incorporate CP metric as a new metric for quality estimation.

5.3.1 Total Power of the Signal

In signal processing, the total power is crucial for calculating SNR and reducing noise. Signals comprise a consistent desired component and a variable noise component. Averaging across multiple instances reduces noise variability, decreasing the total power through averaged signals. The power of a discrete signal can be quantified with Equation 5.3.

$$P = \frac{1}{N} \sum_{n=0}^{N-1} |s[n]|^2, \quad (5.3)$$

where $s[n]$ represents the n -th sample of the signal, and N is the total number of samples in the signal. For signals with deterministic components and zero-mean noise, total power (P) is the sum of signal (P_s) and noise power (P_n). After averaging M samples, given noise reduction by [13], the power ($P_{M-\text{avg}}$) adjusts as shown in Equation 5.4.

$$P_{M-\text{avg}} = P_s + \frac{P_n}{M}, \quad (5.4)$$

The Formula 5.4 provides a mechanism to compute the power of a signal after M averages, isolating its inherent strength from the noise. Non-stationary signals can exhibit

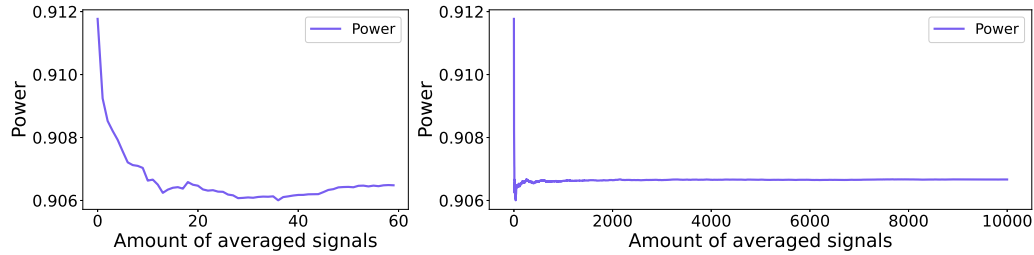


Figure 5.1: Power over averages of the Signal #1

time-varying statistical properties, causing their power to evolve over time. When averaging such signals, the total power does not remain constant or decrease predictably as it would with stationary signals; instead, it may fluctuate as the intrinsic characteristics of the signal and noise vary. The equation 5.3 remains a foundational tool for power calculation, yet its application to non-stationary signals must account for the changing nature of both signal and noise components.

The power values for both stationary and non-stationary signals demonstrate a trend of rapidly diminishing returns from averaging. The change in power seems to be more pronounced during the initial averages and significantly levels off after the first ten averages. This behaviour is demonstrated for signal #1 in Figure 5.1 and is stable over different signal samples. The initial reduction does not necessarily correlate with optimal SNR, indicating that total power alone might not be a reliable metric for determining signal clarity in stationary and non-stationary contexts. The rapid change in total power and the dynamic nature of some signals necessitate a more nuanced approach to signal clarity estimation.

5.3.2 Signal-to-Noise Ratio

The SNR quantifies the extent of noise in a signal, comparing the signal's power levels or amplitudes to noise, where power relates to the amplitude squared. A higher SNR means the signal is dominant, and the noise is less noticeable. On a logarithmic decibel (dB) scale, a factor of 10 applies to power-based calculations and 20 to amplitude-based ones. For our research, we measured the SNR ratio in dB using Equation 5.5, where powers are calculated as mean squares of a signal and noise, respectively.

$$10 * \log_{10}(P_s/P_n) \quad (5.5)$$

SNR improvement is often represented as $10 \log_{10}(N)$, where N denotes the number of averages. The rationale behind this equation is grounded in the principle that while

the signal power remains unchanged with averaging, the noise power reduces, effectively improving the SNR. Specifically, averaging N independent noise samples reduces the noise power by a factor of N . However, its use without preliminary analysis of the application domain is risky, as it does not take into account natural changes of the noise through time.

Real SNR estimation is not straightforward. We tried a few approaches to calculate SNR without additional environmental information to address this issue for the adaptive quality estimation algorithm.

Statistical method

The statistical method is one of the existing methods in the industry, mainly employed for the image processing domain. For this method, P_s and P_n from equation 5.5 are updated with a new equivalency. Signal power is represented as a mean μ of the data, and the noise is represented by the standard deviation σ of the data [79]. The mean/std method applies to continuous data measured on a ratio scale characterized by a true zero point and meaningful ratio comparisons [79]. The discretized power trace data with a true zero representing the absence of power aligns with ratio scale properties, making it suitable for this analysis. Notably, this method is suited only for additive white Gaussian noise.

The experiments show this method's *ineffectiveness* for stationary embedded discretised signals. The expected SNR improvement trend over new averages was not observed when the statistical method was used to calculate the results. The resulting trend follows the opposite trend of the total power, where SNR noticeably improves only during initial averages and then stabilizes.

Cryptoanalysis method

Another method in the domain is a calculation of SNR with real measurements [83]. It starts by collecting a dataset of side-channel traces, each trace comprising time samples that reflect potential leakage from a cryptographic algorithm's intermediate values. The leakage at a specific time sample is modeled as $L = L_d + L_n = g(V) + L_n$, where L_d represents the deterministic part linked to the intermediate value V , and L_n represents the non-deterministic noise, modeled as $L_n \sim N(0, \sigma^2)$.

To compute SNR, the traces are first grouped by the intermediate variable V , resulting in sets L_v for each value v . For each L_v , the mean leakage $\hat{\mu}_v$ is calculated as the average of its elements. The overall mean leakage $\hat{\mu}$ across all groups is then derived by averaging

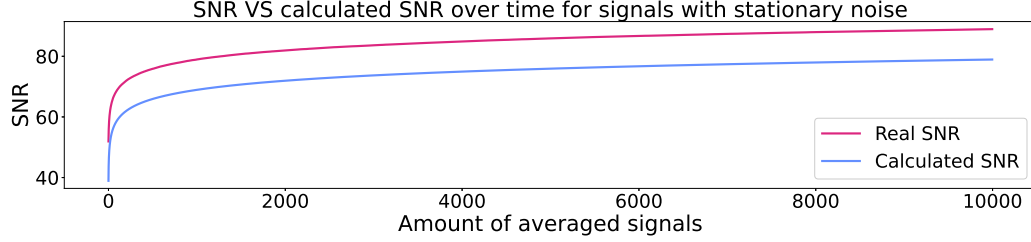


Figure 5.2: SNR vs measured SNR

these means. Similarly, the variance $\hat{\sigma}_v^2$ for each group is computed as the mean squared deviation from $\hat{\mu}_v$, and the overall noise variance $\hat{\sigma}^2$ is the mean of these variances.

The variance of L_d , $Var(L_d)$, representing the spread of the deterministic leakage across all values of V , is calculated next. **SNR** is finally determined as the ratio of $Var(L_d)$ to $\hat{\sigma}^2$, providing a measure of signal strength relative to noise.

This approach was tailored for estimating the **SNR** directly from real measurement data. Still, it suits only a limited application and produces an uninformative correlation to the actual **SNR**.

Subtracting Signal Power from Total Power

Let P_s be defined as $2P_{avg} - P_{initial}$ and P_n as $P_{initial} - P_s$, according to Equation 5.4. We can substitute these values into Equation 5.5 to estimate the SNR using Equation $N = 10^{(\frac{\Delta SNR}{10})}$.

It is important to note that the estimated power and SNR calculations may diverge substantially from actual SNR values in practical scenarios. These estimates may accurately reflect the overall trend for stationary signals but with real-value discrepancies. In the case of non-stationary signals, these estimates may fail to indicate improvement trends accurately. An example of calculated SNR over averages with actual SNR over averages for stationary noise is shown in Figure 5.2. Relying solely on extracted SNR can be unreliable when evaluating a signal's clarity, especially when the signal's nature is unknown.

Fourier Transformation and Artificial Intelligence

Artificial Intelligence (AI) and **ML** models are becoming increasingly popular, positioning them as versatile tools for various tasks. We successfully implemented six algorithms

with the use of Scikit-learn [84], and TensorFlow [4] for SNR prediction using only noisy signals. Table 5.8 showed a detailed comparison of achieved accuracies, errors, and recorded computation time to use the model successfully.

Model	R2	MSE	Train.time	Predic.time	Extract.time
Lin. Reg.	0.91	2.17	0.009	0.002	7.33
Rand. Forest	0.96	1.00	0.30	0.011	0.78
Grad. Boost.	0.97	0.73	0.75	0.004	0.78
SVR	0.97	0.72	3.10	0.44	0.78
RNN	0.87	3.20	2850	5.80	0.00
MLP	0.97	0.73	68.7	0.18	7.33

Table 5.8: Model comparison for signals with aperiodic noise

In the process of predicting the SNR for a collection of noisy signals, we extracted key features from both the time and frequency domains using methodology from Section 3.5.2.

While all algorithms that use extracted features predict SNR with varying degrees of success, only the Recurrent Neural Network (RNN) were proficient at making predictions without manually extracted features. However, despite its success, the RNN was inefficient. Compared to Linear Regression (LR), the RNN was only marginally superior in terms of cumulative prediction and extraction time, largely due to LR’s intensive preprocessing needs. While both LR and RNN showed comparable accuracy, LR is considerably easier to train, avoiding the need for intricate architectural and hyperparameter fine-tuning. All other methods surpassed these two in all terms, with only the Multilayer Perceptron (MLP) taking longer in certain time metrics.

The most efficacious algorithms were Random Forest (RF), Gradient Boosting (GB), and Support Vector Regression (SVR). All demonstrated comparable accuracy and Mean Squared Error (MSE) metrics. Notably, the SVR required more input features for periodic signals than RF and GB, slightly reducing its versatility. GB showed marginally superior metrics but needed longer training and hyperparameter tuning times.

Overall, it is possible to achieve high accuracy by extracting only two features from the frequency domain – spectral entropy and spectral spread, where spectral spread relies on spectral centroid, using ensemble ML methods. However, the required computational time is not efficient even by significantly reducing the feature domain and choosing optimal ML models. SNR itself does not produce an easily predictable trend, and would require multiple computations. Using ML methods, the computation cost of the fuzzing algorithm might unpredictably increase.

5.3.3 Comparative Power

We suggest an alternative metric for an assessment called **CP**, which calculates the ratio between consecutive averaged signals and provides a robust measure for evaluating signal improvement. Additionally, we estimated its relationship to **SNR**. The ratio is measured by adapting Equation 5.5, where we substitute P_s with the power of the current averaged signal $A(\langle ts_1, \dots, ts_n \rangle)$ and P_n with the power of the previous averaged signal $A(\langle ts_1, \dots, ts_{n-1} \rangle)$. The powers are calculated through Equation 5.3.

This approach is particularly effective for non-stationary signals. In contrast to measured **SNR** 5.3.2, **CP** estimates the nature of the noise and introduces a measure for estimating noise stationarity. The greater the variation in values across different measurements, the higher the likelihood of non-stationarity in the noise. Figures 5.3 and 5.4 show the correlation between **SNR** defined by Equation 5.5, measured **SNR** and **CP** for stationary and non-stationary signals, respectively.

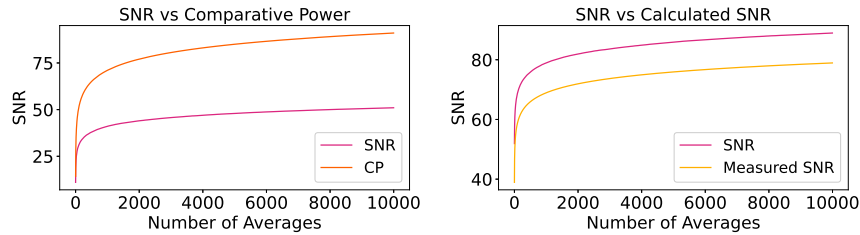


Figure 5.3: SNR vs measured SNR vs CP for stationary signals

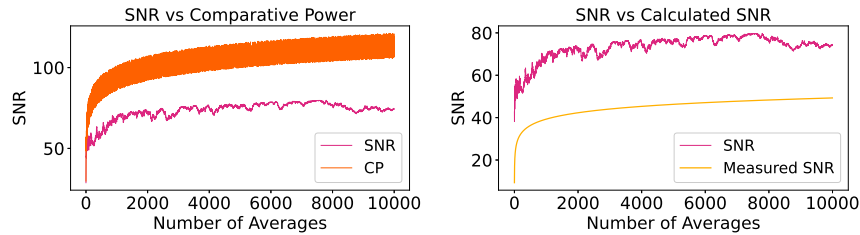


Figure 5.4: SNR vs measured SNR vs CP for non-stationary signals

We can predict **SNR** through linear regression with equation 5.6, where ‘comparative power’ **CP** and ‘min-max’ **MM** are independent variables. The **MM** metric quantifies variability within **CP** measurements by computing the difference between its maximum and minimum values in specific subsets. To calculate **MM**, outliers are removed, and the

average is taken across subsets. Figure 5.5 shows a correlation between SNR, MM, and CP. For particular cases, MM metric reduces MSE. Although we found the MM metric mostly redundant for our final dataset, it could be used for calculating the drop rate.

$$SNR = \alpha + \beta_1 \cdot CP + \beta_2 \cdot MM \quad (5.6)$$

The intercept parameter, denoted as α , exhibits a logarithmic relationship with the index x of the averaged array, which corresponds to the total number of signals averaged. The relationship is given by Equation 5.7.

Beta β coefficients indicate a polynomial dependence on the indexes. Given the complexity inherent in the fit, using a coefficients table or formulating an equation for the exact application domain could be beneficial. Simplifying the scope by narrowing down the possible ranges for variables such as amplitude, frequency, stationarity, and the count of foundational sinusoids will improve equations. This refinement, in turn, facilitates an acceleration in the computation process. Equation examples for the polynomial dependency of β_1 and β_2 on their respective variables are presented in Formula 5.8 and Formula 5.9, where x refers to the number of averages.

$$\alpha = -2.21 - 4.16 \cdot \log(x) \quad (5.7)$$

$$\begin{aligned} \beta_1 = & 0.995 + 5.99 \cdot 10^{-6} \cdot x - 8.00 \cdot 10^{-10} \cdot x^2 \\ & - 5.93 \cdot 10^{-15} \cdot x^3 + 3.87 \cdot 10^{-18} \cdot x^4 \end{aligned} \quad (5.8)$$

$$\begin{aligned} \beta_2 = & -0.192 + 1.37 \cdot 10^{-4} \cdot x - 4.39 \cdot 10^{-8} \cdot x^2 \\ & + 5.87 \cdot 10^{-12} \cdot x^3 - 2.64 \cdot 10^{-16} \cdot x^4 \end{aligned} \quad (5.9)$$

The estimated accuracies of the equations, as indicated by the metrics R^2 , MSE, and Mean Absolute Error (MAE), reveal a high degree of predictive accuracy and consistency across various models. With R^2 values predominantly exceeding 0.97, the models demonstrate a strong ability to predict the SNR from the inputs CP, derived for averaged signals at specific indexes, and MM, consistent across all indexes. The MSE and MAE values, which measure the average magnitude of the prediction errors, show slight variations but remain relatively low, indicating precise predictions with minimal deviation from the actual

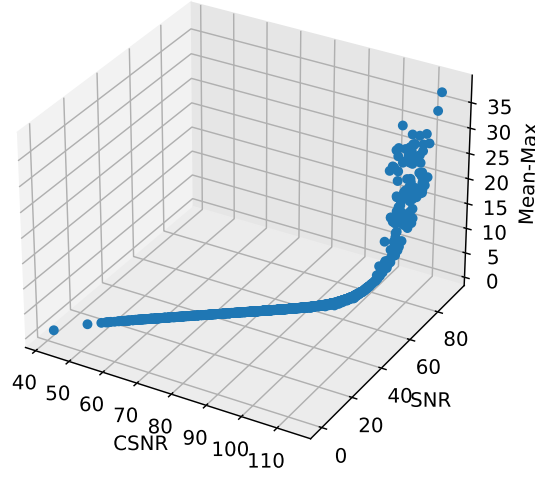


Figure 5.5: Dependency between CP, MM and SNR

values. The maximum [MSE](#) recorded was 4.27, while the [MAE](#) remained below 1. These results suggest that the equations are robust in their capability to capture the underlying patterns in the data, making them suitable for applications requiring high precision in [SNR](#) estimation.

5.3.4 Central Limit Theorem

The [CLT](#) is a fundamental result in probability theory that establishes the conditions under which the sum (or average) of a large number of [independent and identically distributed \(i.i.d.\)](#) random variables converges in distribution to a normal distribution [\[63\]](#). This theorem provides the theoretical foundation for modelling aggregate noise as Gaussian in many empirical and experimental domains, including side-channel analysis [\[105\]](#).

Let $X_1, X_2, \dots, X_n$ denote a sequence of [i.i.d.](#) random variables drawn from a population with mean μ and finite variance σ^2 . The sample mean is defined as

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i. \quad (5.10)$$

The standardized form of the sample mean is given by

$$Z_n = \frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}}. \quad (5.11)$$

The [CLT](#) states that as $n \rightarrow \infty$, the distribution of Z_n converges to the standard normal distribution:

$$Z_n \xrightarrow{d} \mathcal{N}(0, 1). \quad (5.12)$$

Equivalently, the distribution of the sample mean $\bar{X}_n$ converges in distribution to a normal distribution with mean μ and variance σ^2/n :

$$\bar{X}_n \xrightarrow{d} \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right). \quad (5.13)$$

In side-channel analysis, particularly in power or electromagnetic measurements, each recorded trace T_i is often modelled as a superposition of a deterministic component S_i (representing signal) and a stochastic noise component N_i :

$$T_i = S_i + N_i. \quad (5.14)$$

Assuming that the noise terms N_i are [i.i.d.](#) with finite variance, the [CLT](#) implies that the sample average of repeated measurements will approximate a Gaussian distribution, even if the individual noise distributions are non-Gaussian. Specifically, for a fixed signal value S , the distribution of the mean trace over n measurements converges as:

$$\frac{1}{n} \sum_{i=1}^n T_i \xrightarrow{d} \mathcal{N}\left(S, \frac{\sigma^2}{n}\right). \quad (5.15)$$

This result justifies the common practice of modelling measurement noise in side-channel traces as Gaussian when a sufficiently large number of samples are available. Moreover, it underpins the statistical power of many parametric techniques used in side-channel attacks, such as correlation power analysis and template matching, which often assume normally distributed noise for theoretical and practical tractability.

However, while [SCA](#) methodologies commonly operate under this Gaussian assumption driven by the [CLT](#), practical experiments indicate that reaching a fully Gaussian approximation often requires an impractically large number of samples.

The required sample size n to approximate the noise component by a normal distribution within a specified margin of error in a practical experiment was calculated as:

$$n = \left(\frac{z_{\alpha/2} \sigma}{E} \right)^2, \quad (5.16)$$

where $z_{\alpha/2}$ is the critical value associated with the desired confidence level, σ is the estimated standard deviation of the noise, represents the acceptable margin of error.

Applying the standard CLT-based approach to a GSM module indicated a sample size needed exceeding two million traces. In practice, however, stable results sufficient for power fuzzing analysis were achieved with as few as 100 samples. This substantial discrepancy suggests that strict adherence to Gaussian assumptions may lead to significant overestimation in sample size recommendations, highlighting that a fully Gaussian noise distribution may not always be necessary for effective side-channel analysis.

5.3.5 Correlation-Based Trace Prediction

Predicting noise levels in side-channel measurements is inherently challenging due to numerous influencing factors such as hardware variability, measurement setups, environmental conditions, and power supply characteristics. Consequently, empirical side-channel analyses often simulate varying noise levels to evaluate attacks' robustness under realistic conditions [15].

In practice, simulated power traces are initially generated from an idealised power consumption model. For example, the simple 1-bit Hamming Distance (HD) model assigns a binary value based on whether two consecutive signals differ (1) or remain unchanged (0), resulting in an ideal, noise-free power trace. Realistic conditions are simulated by superimposing additive Gaussian noise with mean μ and standard deviation σ , denoted as $\mathcal{N}(\mu, \sigma^2)$:

$$T_{\text{noisy}} = T_{\text{ideal}} + \mathcal{N}(0, \sigma^2). \quad (5.17)$$

Since the Pearson correlation coefficient, commonly used in CPA, is invariant to shifts by a constant mean, noise is conventionally assumed to have zero mean ($\mu = 0$) [15].

To connect this theoretical noise representation to practical hardware, noise variance σ^2 can be expressed equivalently as the activity level of randomly switching registers within a device. If each register independently switches with a probability of 0.5 and contributes one arbitrary unit of power consumption, the resulting power distribution approximates a Gaussian distribution with parameters:

$$\mu = \frac{n}{2}, \quad \sigma = \sqrt{\frac{n}{4}} \quad \Rightarrow \quad \sigma^2 = \frac{n}{4} \quad \Rightarrow \quad n = 4\sigma^2. \quad (5.18)$$

Thus, the noise standard deviation σ offers an intuitive interpretation as the equivalent number of randomly switching registers.

The effectiveness of correlation-based attacks depends significantly on the number of collected traces (N), the inherent noise variance (σ_N^2), and the variance of the employed hypothetical power model ($\text{var}(H)$). The Pearson correlation coefficient p between the hypothetical model H and the noisy measurement X can be modeled as [15]:

$$p = \frac{1}{\sqrt{1 + \frac{\sigma_N^2}{\text{var}(H) \cdot N}}}. \quad (5.19)$$

Rearranging this expression to determine the required number of traces N yields:

$$N = \frac{\sigma_N^2}{\text{var}(H)} \left(\frac{1}{p^2} - 1 \right), \quad (5.20)$$

where σ_N^2 is the noise variance, $\text{var}(H)$ denotes the variance of the power model, p represents the desired or observed correlation coefficient. The variance of hypothetical power models $\text{var}(H)$ strongly depends on both the bit width and the type of leakage model employed (Hamming Weight vs. Hamming Distance).

Different embedded systems exhibit significantly different leakage characteristics and complexity [15]:

- **Simple microcontrollers (e.g., PIC, AVR):** Typically feature narrow datapaths (8 or 16 bits) with relatively predictable and lower switching activities. Here, straightforward HW or HD models often accurately represent leakage, leading to comparatively high correlation coefficients.
- **Complex OS-level systems (e.g., CPUs, FPGAs):** These have wider datapaths (32/64 bits), higher parallelism, and substantial noise originating from simultaneous system processes. Such complexity significantly reduces correlation coefficients, increasing effective noise. More detailed or single-bit transition models may be necessary to accurately characterize leakage in these contexts.

Table 5.9 provides a summary of representative CPA attacks from the literature, illustrating typical correlation coefficients, associated noise variances, and their equivalent switching register activity [15].

We applied the correlation-based trace prediction method to a GSM module. The resulting estimation indicated a requirement of over 30,000 traces to achieve a target correlation coefficient, which is significantly lower than the CLT-based estimate, but still overestimates the number of traces needed in practice.

Table 5.9: Summary of CPA attacks from the literature, including correlation coefficient (cc), noise variance, equivalent switching registers, and used number of traces.

Target Design	Power Model	cc	Noise Variance	Switching Registers	Used Traces
PIC16F886	8-bit HD	0.75	5.1	12.5	100
Yubikey 2 (EM)	8-bit HW	0.3	42	161	300
Virtex-4 Bitstream	1-bit HD	0.08	39	155	60,000
Virtex-5 Bitstream	1-bit HD	0.04	156	623	90,000
SHA-1 EEPROM	8-bit HW	0.1	400	1600	2,500
Kintex-7 FPGA	8-bit HW	0.1	400	1600	6,000
Yubikey 2 (Power)	8-bit HW	0.06	1110	4430	6,400
Stratix II Bitstream	8-bit HD	0.05	1600	6400	400,000
Mifare DesFire (EM)	4-bit HD	0.01	10,000	40,000	250,000

5.3.6 Summary

Accurately estimating the quality of power side-channel signals is essential for predicting the required number of traces needed to distinguish signal characteristics from noise reliably. The preceding analysis evaluated several signal quality metrics, including total power, [SNR](#), [CP](#), and statistical approaches based on the [CLT](#) and correlation-based trace predictions.

Total signal power provides a straightforward measure that reflects noise reduction through averaging. However, this metric exhibits diminishing returns rapidly after initial averages and does not reliably correlate with optimal [SNR](#), especially in non-stationary environments. Thus, total power alone is insufficient as a standalone signal quality indicator.

The [SNR](#) is conceptually robust but challenging to estimate accurately in practice. Various approaches, such as statistical mean, standard deviation methods, cryptanalysis-based groupings, and subtractive power techniques, show differing degrees of effectiveness. The statistical method was ineffective for embedded discrete signals, while the cryptanalysis method provided a theoretically sound but practically limited application. The subtractive approach, though practically appealing, often deviates substantially from true [SNR](#) values.

Machine-learning methods have demonstrated high predictive accuracy for [SNR](#) estimation, particularly ensemble methods such as Random Forest, Gradient Boosting, and [SVR](#), achieving R^2 values exceeding 0.96. Despite their accuracy, these methods require substantial computational overhead for feature extraction, training, and prediction, thus limiting their practical utility in scenarios with real-time constraints.

Proposed [CP](#) metric, which measures the ratio of powers between consecutively aver-

aged signals, provides a robust and intuitive indication of signal improvement over successive averages. Combined with a variability metric, linear regression models using CP offer accurate and reliable estimates of SNR ($R^2 > 0.97$). The CP metric is particularly effective for handling non-stationary signals due to its inherent adaptability.

Classical statistical methods like the CLT and correlation-based trace prediction offer closed-form estimates of the required number of averages. However, the CLT-based formula significantly overestimates practical sample sizes – for example, predicting over two million traces for GSM, whereas only around 100 are practically sufficient. Correlation-based methods improve this estimate substantially (around 2,500 traces for GSM), yet remain exaggerated.

In summary, while traditional metrics such as total power and direct SNR estimation provide foundational insights, they either exhibit rapid saturation or practical limitations in accurately quantifying signal quality, especially in non-stationary contexts. Machine learning approaches offer remarkable predictive accuracy, yet their computational complexity limits applicability in real-time or resource-constrained environments. Conversely, the CP metric combined with variability-based linear regression achieves both high accuracy and practical efficiency, effectively addressing the shortcomings of classical methods. CLT- and correlation-based trace prediction techniques are well-established techniques in the domain. However, when transferred to the hardware fuzzing domain, they tend to substantially overestimate the number of traces. Still, we believe that with appropriate adjustments, they hold significant potential for providing efficient and theoretically grounded estimations. Thus, integrating CP-based methodologies alongside refined statistical approaches provides an optimal balance between accuracy and computational practicality, enhancing the robustness of trace-number predictions in power side-channel analyses.

5.4 Noise Mitigation and Robust Classification Proposed Strategies

This section presents a structured set of complementary noise-mitigation and classification techniques tailored for powertrace-based fuzzing. Building on prior evaluations of noise mitigation techniques, it outlines the most promising noise removal strategies, and proposes an integrated approach that combines bias removal, adaptive sample-size estimation, outlier filtering, DTW-based averaging or SSA, shared-noise subtraction, and targeted classification methods to robustly address diverse noise sources and improve anomaly detection accuracy.

5.4.1 Methodology and Approach

Based on an evaluation of noise mitigation strategies presented in Table 5.7, five methods emerged as particularly effective in addressing a broad spectrum of noise types commonly encountered in side-channel analysis: Trace Averaging, SSA, ICA, ML, and DPA/CPA. Each technique has distinct advantages and limitations regarding applicability, computational complexity, and effectiveness across different noise scenarios.

Trace averaging stands out due to its simplicity, ease of implementation, and robustness. Its performance is generally resilient even against non-Gaussian noise distributions, provided a sufficient number of samples is collected. Given these advantages, trace averaging is an effective baseline technique suitable for most side-channel contexts, especially when computational resources or expertise for parameter tuning are limited.

SSA offers a more sophisticated, data-driven approach, decomposing traces into interpretable signal and noise components. Its implementation is straightforward, requiring only basic parameter tuning, primarily selecting the appropriate number of eigenvalue groups, typically isolating the first component group as signal-related. SSA demonstrates strong capability in noise suppression, although computational overhead is significantly higher compared to basic trace averaging. Nonetheless, the computational demands remain manageable in practice and can justify the computation overhead given reduced acquisition overhead compared to trace averaging.

ICA theoretically offers powerful capabilities for separating independent source signals from mixed observations, thus potentially improving noise reduction and clarity in the processed signals. However, the practical efficacy of ICA is critically dependent on the accurate determination of parameters, such as the number of independent components, which significantly impacts the outcome. Additionally, identifying the correct component corresponding to the signal of interest can be challenging, as the order of components provided by ICA is not consistent. Moreover, ICA requires considerable computational resources and processing time, limiting its feasibility on constrained hardware such as the Raspberry Pi. Experimental results on RPI indicated ICA performed notably worse than simple trace averaging across several optimisation attempts. Finally, ICA struggles with non-Gaussian noise distributions, further limiting its universal applicability.

ML approaches provide flexibility and adaptive learning capabilities for noise mitigation but present significant practical challenges. Previous studies have demonstrated ML's inferior performance compared to traditional trace averaging methods, mainly when generalised datasets are unavailable [107]. ML methods typically require extensive and target-specific training datasets, preprocessing, and training, severely impacting scalability.

DPA and CPA techniques essentially combine grouping of traces based on input hypotheses, averaging, and subsequent differential analysis. While powerful in cryptographic side-channel analysis, these methods are impractical for PowerFuzzer applications due to the extensive input space, necessitating an unmanageable number of hypotheses or key guesses. Furthermore, the beneficial aspects of DPA/CPA (trace grouping and averaging) are effectively encompassed by existing methods within the evaluation framework, reducing their additional utility in this context.

Despite their strengths, none of the evaluated strategies address jitter noise or effectively remove algorithmic and operating system-induced interferences. Therefore, additional algorithmic noise suppression strategies and either elastic alignment or distribution-preserving methods must be incorporated.

Once the noise mitigation strategy is chosen, estimating the number of samples required to achieve successful classification is essential. Classical approaches based on the CLT and correlation-based trace prediction provide well-established, theoretically sound methods for sample size estimation. However, both techniques tend to overestimate the number of traces needed in practical scenarios significantly. With appropriate adjustments and empirical calibration, these methods hold substantial potential for efficient and accurate trace prediction. Recent evaluations highlight the CP metric as an effective predictor of signal improvement during averaging. Monitoring the progression of CP values across successive averages makes it possible to detect the point of diminishing returns where additional traces no longer meaningfully enhance signal quality.

Therefore, to achieve robust classification and analysis, we propose a structured combination of several strategies:

- **Bias Removal:** Techniques such as randomised trace acquisition sequences and temperature regulation mitigate systematic measurement biases at acquisition time, thus increasing PowerFuzzer robustness.
- **Adaptive Sample Distribution Size Estimation:** The system can efficiently balance robustness and resource usage by continuously assessing signal quality indicators and adjusting acquisition in real time.
- **Distributional Preprocessing (Outlier Removal):** Identifying and removing outliers enhances the reliability of statistical representations and prevents distortion of averaging or decomposition methods by extreme, non-representative measurements.
- **Advanced Noise Removal Techniques:**

- **DTW-based Trace Averaging:** DTW-based averaging aligns and averages traces adaptively, effectively reducing jitter-induced misalignments, resulting in more representative signal averaging.
- **SSA:** SSA improves powertraces by decomposing signal and noise explicitly, preserving essential distributional information while significantly reducing structured noise.
- **Shared Noise Identification and Subtraction:** This approach explicitly targets deterministic, common noise artefacts, such as algorithmic noise or OS-induced interference (if aligned by OS response), by identifying and subtracting shared signal components common to multiple traces.
- **Cropping Irrelevant Trace Segments:** Removing segments of traces unrelated to the target command reduces unnecessary variability and computational complexity.
- **Classification Techniques:**
 - **T-Test Classification:** Employed for scenarios involving clearly defined, representative averages, this statistical test provides effective classification capability in cases where trace averaging methods have stabilised representative traces.
 - **Distributional Analysis:** When jitter remains problematic or when using methods such as SSA that preserve distributional information, distribution-based statistical tests such as multivariate T-Test provide superior classification performance by explicitly accounting for inherent trace variability.

Integrating these complementary strategies enables robust, adaptive, and accurate noise mitigation and classification tailored explicitly to PowerFuzzer contexts.

5.4.2 Unbiased Power Measurement

Reliable and accurate power measurements are crucial for effective power-based side-channel fuzzing. However, systematic biases significantly affect the quality of acquired traces, potentially misleading side-channel analyses and subsequent fuzzing outcomes [48].

Glamočanin et al. [48] demonstrated that FPGA-based sensors are sensitive to temperature fluctuations, affecting both the DC offset and variance of power traces. Their results show that even modest temperature variations can substantially degrade the effectiveness

of correlation-based attacks, requiring significantly more traces to reliably recover cryptographic keys. Moreover, ML-based side-channel models, when trained on biased datasets, tend to exploit these temperature-induced artefacts rather than actual leakage patterns.

Papagiannopoulos et al. [83] state that randomised acquisition helps mitigate measurement biases and ensures better statistical independence between samples. Sequential acquisition of multiple traces for a single command risks introducing correlation and systematic biases that can either mask or exaggerate leakage.

To mitigate bias in measurements, the analyst should integrate the following points:

- Power traces corresponding to different operations should be acquired in an interleaved and randomised order to ensure uniform environmental influence across all classes.
- The internal state of the SUT should be fully reinitialised between successive measurements to eliminate inter-trace dependencies and ensure statistical independence of acquired traces.

5.4.3 Adaptive Estimation Strategy for Trace Requirements

Effective power-based side-channel analysis critically depends on acquiring an optimal number of measurement traces – sufficient to reliably distinguish signal from noise, yet without unnecessary overhead. To address this challenge, we propose to integrate an adaptive estimation strategy that dynamically adjusts trace acquisition in real-time based on evolving estimates of signal quality.

First, adaptive acquisition systems should be implemented to initially collect a fixed number of traces n and perform an early estimation of the total required trace count based on the observed signal quality. We suggest $n = 100$ as the smallest number we used in successful PowerFuzzer experiments and seen in literature (Table 5.9). As acquisition progresses, the system should periodically re-evaluate signal conditions to verify that the acquisition remains on track toward the predicted goal. The frequency of these re-evaluations can gradually decrease over time, following either a linear or logarithmic schedule, as confidence in the prediction increases.

The CP metric is central to this adaptive approach, which provides a practical method for assessing noise suppression efficacy throughout the averaging process. By examining the ratio of signal powers between successive averaged measurements, the system can intuitively

estimate the rate at which noise reduces and quickly identify the point where additional measurements yield minimal or no significant improvements.

Alternatively, correlation-based trace prediction methods, traditionally prone to over-estimation, can be refined through domain-specific calibration tailored explicitly to the PowerFuzzer environment. Correlation coefficients should be empirically established via controlled experiments, comparing predicted trace counts with actual, observed numbers of required traces under representative operational conditions.

This proposed strategy integrates adaptive acquisition systems informed by the CP metric and calibrated correlation-based predictions to ensure optimal trace count estimation, enhancing power-based side-channel classification’s efficiency and practical applicability through acquisition optimisation.

5.4.4 Distribution Preprocessing

To enhance the quality and consistency of side-channel measurements, we propose incorporating an outlier detection and removal step before further response assessment. This step mitigates the impact of anomalous traces arising from transient disturbances, hardware glitches, or acquisition artefacts.

Identifying and discarding anomalous traces before distribution averaging can reduce the influence of rare but high-magnitude deviations that would otherwise distort statistical metrics such as mean and variance.

The proposed method involves computing the mean trace across the command responses distribution and subsequently identifying outlier traces based on their ED from this mean. Specifically, the distribution of ED distances is analysed using the IQR rule $[Q_1 - \alpha \cdot \text{IQR}, Q_3 + \alpha \cdot \text{IQR}]$, where Q_1 and Q_3 are the first and third quartiles respectively and α is predefined threshold. The equation could be simplified to account only for Q_3, as we are interested in only outliers that lie far from the mean trace.

This procedure ensures that only statistically representative traces are retained for subsequent analysis, thereby improving the SNR and preserving response-relevant peaks from smoothening.

5.4.5 Downsampled DTW Averaging

To mitigate the effects of clock jitter and response stretching in power traces, we propose employing trace averaging based on the DTW distance, commonly referred to as Dynamic

[Time Warping Barycenter Averaging \(DBA\)](#). Table 5.5 notes that elastic alignment techniques such as [DTW](#) offer the highest mitigation score for trace misalignment. However, their practical applicability is hindered by the quadratic time complexity of [DTW](#), which significantly increases implementation complexity and reduces the final score in realistic measurement pipelines.

[DBA](#), while powerful, becomes computationally infeasible in the context of PowerFuzzer, where trace lengths frequently exceed 10,000 samples and can reach up to 500,000 samples per trace. To address this scalability challenge, we propose a novel method: **Shrunk-DBA**, an efficient approximation of elastic averaging tailored for extensive, noisy time-series data.

Our approach introduces a preprocessing and constraint pipeline that enables [DBA](#) to scale without sacrificing alignment quality. The procedure includes the following steps:

- **Smoothing:** Each trace is first smoothed using a Gaussian filter to suppress high-frequency noise and reduce sensitivity to irrelevant signal fluctuations.
- **Downsampling:** The smoothed traces are then downsampled to a manageable resolution (10-60 samples) to reduce computational load while preserving global alignment structure.
- **Constraint Estimation:** Based on an estimate of the typical response duration or expected misalignment magnitude, the parameters for the Sakoe-Chiba band are computed to limit the [DTW](#) search space and enforce locality [45].
- **Shrunk-DBA Execution:** The downsampled traces are used to compute an approximate alignment path via constrained [DTW](#). This alignment is then projected back to the original resolution and applied to the full-length traces, enabling robust and efficient averaging.

By combining smoothing, downsampling, and constrained [DTW](#) search, we offer a practical approach for long traces. It enables time-aligned averaging at scale and improves resilience to temporal distortions, making it suitable for high-resolution power traces.

Shrunk-DBA Algorithm

Computational limitations of [DTW](#) averaging are often underexplored [86]. Many studies leave optimisation for dynamic programming or assume unlimited time and memory

available, overlooking the importance of data transformation algorithms before and after the DTW path computations. This section introduces a data processing module that can be combined with existing DTW averaging techniques to enhance their computational efficiency while preserving alignment accuracy. Experimental results illustrate the proposed approach’s practical gains, suggesting its potential to make real-time DTW averaging feasible for large-scale or resource-constrained systems.

Shrunk-DBA is an improved DBA algorithm that reduces computational complexity by performing DTW on downsampled sequences before full-resolution refinement.

DTW is a well-established technique for measuring the similarity between time series sequences by allowing flexible temporal alignments [75]. Given two sequences:

$$A = (a_1, a_2, \dots, a_S), \quad B = (b_1, b_2, \dots, b_T), \quad (5.21)$$

DTW constructs an optimal alignment by computing a cost matrix $D(i, j)$ recursively as:

$$D(i, j) = d(a_i, b_j) + \min \begin{cases} D(i-1, j-1) & \text{(Match)} \\ D(i, j-1) & \text{(Insertion in } A) \\ D(i-1, j) & \text{(Insertion in } B) \end{cases} \quad (5.22)$$

where $d(a_i, b_j)$ represents a local distance measure, often chosen as the squared Euclidean distance [109]:

$$d(a_i, b_j) = (a_i - b_j)^2. \quad (5.23)$$

A key byproduct of DTW is the warping path, which specifies how points in sequence A correspond to points in sequence B . This path provides a temporal mapping that is fundamental for computing a meaningful barycenter when averaging multiple time series.

DBA is an iterative refinement algorithm that computes an average sequence, or barycenter, of a set of time series under DTW alignment [87]. Given a dataset of time series $\mathcal{X} = \{X_1, X_2, \dots, X_N\}$, DBA aims to find a representative sequence $C = (c_1, \dots, c_T)$ that minimizes the sum of squared DTW distances:

$$C^* = \arg \min_C \sum_{i=1}^N DTW^2(C, X_i). \quad (5.24)$$

DBA operates iteratively:

1. Initialize the barycenter C using an arbitrary sequence from $\mathcal{X}$.

2. Compute DTW alignments between C and each sequence X_i .
3. Assign each barycenter point c_t the mean of all aligned points from $\mathcal{X}$.
4. Update C and repeat until convergence.

This process ensures that the barycenter preserves meaningful temporal alignments. DBA is computationally expensive due to the underlying quadratic complexity of DTW.

To reduce DBA's computational time and resources, we introduce an optimized approach, Shrunk-DBA. This approach leverages downsampling to perform alignments on lower-resolution versions of the sequences before refining the barycenter at full resolution.

The Shrunk-DBA algorithm follows these steps:

1. Initialize the barycenter C and its shrunk version C_h using an arbitrary sequence from $\mathcal{X}$.
2. Downsample all time series X_i by a factor of k (selecting every k -th point).
3. Compute DTW alignments between the shrunk sequences and a shrunk barycenter.
4. Expand the shrunk DTW path back to the full resolution.
5. Assign original-resolution points based on the expanded DTW path.
6. Compute the mean of assigned points, updating the full-resolution barycenter.
7. Repeat until convergence.

The standard DBA algorithm has a time complexity of:

$$O(NT^2), \tag{5.25}$$

where N is the number of time series and T is their length. This is due to the need to compute N pairwise DTW alignments at each iteration.

In contrast, Shrunk-DBA first computes DTW on downsampled sequences of length $T' = T/k$, which reduces DTW complexity per iteration to:

$$O\left(N\frac{T^2}{k^2}\right). \tag{5.26}$$

Expanding the DTW path back to full resolution adds an $O(NT)$ cost, which is linear in T . The full assignment update and barycenter computation steps also remain $O(NT)$. Thus, the total complexity of Shrunk-DBA per iteration is:

$$O\left(N\frac{T^2}{k^2} + NT\right). \quad (5.27)$$

The speedup factor of Shrunk-DBA compared to standard DBA is approximately $O(k^2)$. This demonstrates that the approach provides significant computational savings, particularly for datasets where T is large. The memory requirements are also reduced, as the cost matrix for DTW is of size $O(T^2)$ in standard DBA but only $O(T^2/k^2)$ in Shrunk-DBA.

The optimized DBA method was built on top of tslearn DBA implementation [109], and is formally outlined in Algorithm 10.

Algorithm 10: Shrunk DTW Barycenter Averaging (Shrunk-DBA)

Input: $\mathcal{X} = \{X_1, \dots, X_N\}$: set of time series,
shrink_factor k : downsampling factor,
max_iter: maximum iterations,
tol: convergence threshold

Output: Final barycenter C

```

1 Convert  $\mathcal{X}$  to shrunk dataset  $\mathcal{X}'$  (downsampled)
2 Initialize barycenter  $C$  using a sequence from  $\mathcal{X}$ 
3 Compute initial shrunk barycenter  $C' = C[:, k]$ 
4 Initialize  $cost_{prev} = \infty$ 
5 for  $iter = 1$  to  $max\_iter$  do
6     Compute DTW alignments between  $C'$  and each  $X'_i$ 
7     Expand DTW paths to full resolution
8     Assign full-resolution points based on the expanded path
9     Compute new barycenter  $C$  as the mean of assigned points
10    Compute shrunk barycenter  $C' = C[:, k]$ 
11    Compute new cost  $cost = \sum_{i=1}^N DTW^2(C, X_i)$ 
12    if  $|cost_{prev} - cost| < tol$  then
13        break
14     $cost_{prev} \leftarrow cost$ 
15 return  $C$ 

```

The proposed Shrunk-DBA algorithm was evaluated across various shrink factors to assess its impact on classification accuracy, computational speed, and memory consumption. The results indicate a clear trade-off between computational efficiency and accuracy. Specifically, the highest accuracy of 0.523 was obtained using standard DBA ($k = 0$) and Shrunk-DBA with a shrink factor of $k = 2$. For moderate shrink factors ranging from $k = 2$ to $k = 6$, accuracy remained relatively stable between 0.500 and 0.523. Beyond a shrink factor of $k = 10$, accuracy experienced a more pronounced decline, reaching a minimum of 0.460 at $k = 14$. However, no clear correlation emerged between accuracy degradation and trace length within the evaluated range (1000–3000 samples), suggesting that accuracy losses are more closely associated with the density of informative features rather than trace duration alone.

Significant improvements in computational performance were observed with increasing shrink factors. Execution time decreased notably, from 18.027 seconds for standard DBA to approximately 7.375 seconds at $k = 6$, reflecting a $2.4\times$ improvement. Peak memory usage exhibited even more substantial reductions, dropping from 6524.018 MB at $k = 0$ to just 11.965 MB at $k = 6$, representing a 99.8% decrease. While further down-sampling continued to yield incremental performance gains, substantial accuracy losses beyond $k = 10$ provided diminishing returns. Thus, Shrunk-DBA with moderate shrink factors ($k \leq 6$) demonstrates the optimal balance, substantially enhancing computational efficiency and memory usage with minimal impact on accuracy, making it particularly suitable for resource-constrained environments and real-time applications. It is important to note that these results were obtained on relatively short traces compared to actual PowerFuzzer data. For the significantly longer traces typical in our measurement pipeline, larger shrink factors can be considered more readily without compromising overall alignment quality.

5.4.6 Singular Spectrum Analysis

SSA achieved the second-highest final mitigation score among all evaluated techniques, demonstrating strong performance across five distinct noise categories, including EMI, thermal noise, and ADC-related disturbances. This broad applicability and moderate complexity balance highlight SSA as a robust and practical choice for mitigating side-channel interference.

SSA is a powerful, nonparametric technique for time series analysis and component separation. The method operates in two key stages: decomposition and reconstruction [7].

Given a real-valued time series $F = (f_0, f_1, \dots, f_{N-1})$, the first step is to embed the

series into a trajectory matrix $X \in \mathbb{R}^{L \times K}$ by defining a window length L such that $1 < L < N$, and setting $K = N - L + 1$. The trajectory matrix is formed as:

$$X = \begin{bmatrix} f_0 & f_1 & \cdots & f_{K-1} \\ f_1 & f_2 & \cdots & f_K \\ \vdots & \vdots & \ddots & \vdots \\ f_{L-1} & f_L & \cdots & f_{N-1} \end{bmatrix}$$

This matrix is then subjected to [SVD](#):

$$X = \sum_{j=1}^d \sqrt{\lambda_j} U_j V_j^\top,$$

where λ_j are the singular values in decreasing order, U_j are the left singular vectors, and V_j are the right singular vectors. Each triple $(\sqrt{\lambda_j}, U_j, V_j)$ is known as an eigentriple.

In the reconstruction phase, a subset $J \subset \{1, \dots, d\}$ of eigentriples is selected. These components are aggregated and converted back into a time series via diagonal averaging, reconstructing a signal component:

$$X_J = \sum_{j \in J} \sqrt{\lambda_j} U_j V_j^\top \quad \Rightarrow \quad G = (g_0, \dots, g_{N-1})$$

The choice of window length L and component subset J determines which features (e.g., trends, periodicities) are captured. Alexandrov [\[7\]](#) proposed [SSA](#) enhancement for trend extraction, where components are selected based on their low-frequency spectral contribution:

$$C(U_j, \omega_0) = \frac{\sum_{k: 0 \leq \frac{k}{L} \leq \omega_0} I_{U_j} \left(\frac{k}{L} \right)}{\sum_{k=0}^{\lfloor L/2 \rfloor} I_{U_j} \left(\frac{k}{L} \right)} > C_0,$$

where $I_{U_j}(\cdot)$ denotes the periodogram of the eigenvector U_j , and ω_0, C_0 are thresholds for frequency and energy contribution respectively. This approach ensures that only components dominated by slow, smooth variations are retained.

For our evaluation, we implemented [SSA](#) using the `pyts` Python library [\[38\]](#), which provides efficient routines for time series transformation and analysis. This enabled automated assessment and parameter tuning across a wide range of noise profiles.

5.4.7 Shared Noise Removal

Algorithmic noise refers to deterministic components within side-channel traces that consistently appear regardless of the specific command executed. These components do not contribute meaningful information about command-specific behavior and are neither random nor transient. Instead, algorithmic noise arises from shared computational overhead, common I/O setup routines, repetitive library function calls, or consistent control-flow operations.

We propose two variants for mitigating algorithmic noise: a simpler singular method and a more sophisticated multivariate approach.

Singular Variant: The singular method involves computing the mean trace across all available unknown-command samples and subsequently subtracting this average from each individual trace. While computationally efficient, this approach assumes that the shared noise remains consistent across all traces and ignores potential variability within subsets of the data.

Multivariate Variant: The multivariate approach seeks to improve on the limitations of the singular method by recognising that algorithmic noise may exhibit structured variability across different execution contexts or temporal segments. Instead of assuming uniformity, this method applies clustering techniques, such as K-means, hierarchical clustering, or Gaussian Mixture Models, to the set of traces, treating each trace as a vector in a high-dimensional feature space.

Removing algorithmic noise is essential for successful post-classification, particularly in scenarios where shared execution patterns can obscure command-specific signal characteristics. We propose a differential averaging strategy inspired by the [DPA](#) principles to address this challenge. Unlike [DPA](#), our approach is not aimed at key recovery but at enhancing signal clarity by suppressing deterministic, data-independent components present across all traces. While the method requires further empirical validation and refinement, it demonstrates strong theoretical promise as a preprocessing step to improve the discriminative quality of traces in classification pipelines.

5.4.8 Trace Length Adjustment

While explicit trace length adjustment is not strictly required, cropping out irrelevant sections containing primarily noise can significantly enhance subsequent statistical analyses by amplifying the contrast between genuinely differing trace segments. Automating this selective cropping process is feasible by employing [Algorithm 6](#) introduced in [Section 4.3.2](#).

Specifically, by identifying the precise response windows and cropping traces according to the longest detected response window, one ensures a consistent and meaningful temporal alignment, thus enhancing the effectiveness of further side-channel evaluations.

5.4.9 Classification

Choosing an appropriate classification method depends significantly on the specific denoising strategy adopted. Different preprocessing methods affect side-channel traces differently, particularly regarding trace alignment, jitter removal, and the preservation of distributional characteristics. Therefore, classification approaches must align carefully with the preprocessing outcomes to achieve accurate and reliable results.

When employing the Shrunk-DBA method for jitter removal and alignment, the resulting dataset is represented effectively by a single, optimally aligned trace. Shrunk-DBA inherently selects or synthesises the most representative trace, reducing jitter-related variance and improving the SNR. In contrast, SSA preserves the distributional properties of the noise-reduced signals but does not inherently address alignment issues such as jitter or temporal stretching. Hence, subsequent alignment methods might be required post-SSA or distributional classification.

Two principal classification strategies suitable for analysing side-channel data following these denoising approaches might be based on T-Test or clustering.

The original T-Test employed for Powertrace-based fuzzing [32] compares the average trace corresponding to a specific command against the global average of traces from the unknown command distribution. This method assumes that an ideal, representative average trace sufficiently characterises the signal’s behaviour for classification purposes. The Original T-test is effective when trace averaging techniques are employed, as these methods tend to smooth transient artefacts without significantly distorting overall signal characteristics, preserving variance indicative of distinct command behaviours.

We also explored alternative clustering approaches, such as K-means clustering based on time-domain with frequency-domain features (Section 3.5.2) or raw power traces. Empirical evaluations indicate these methods yield comparable classification results without substantially improving or degrading statistical significance (p-values) compared to the original T-test.

Traditional average-based classification techniques become suboptimal when employing SSA or other noise reduction methods that retain distributional characteristics without explicitly handling jitter and temporal misalignment. We require classification techniques

designed to compare distributions directly. The multivariate T-Test represents a suitable alternative classification method in such contexts, as it explicitly compares distributions rather than single representative averages.

In summary, the selection of a classification strategy must be closely matched to the chosen denoising method. Shrunk-DBA effectively complements single-average approaches like the original T-Test. Meanwhile, distribution-preserving denoising techniques, such as SSA, benefit significantly from distributional classification methodologies like the multivariate T-Test.

Chapter 6

Conclusion

This thesis systematically addressed critical challenges associated with powertrace-based fuzzing by building upon the initial PowerFuzzer framework. Our research improved the robustness and applicability of powertrace-based fuzzing methodologies by introducing novel strategies, algorithms, and frameworks to overcome practical limitations observed in early implementations. Four challenges from Section 2.4, namely offset and sampling rate calibration (p_1), response window identification (p_2), trace quality enhancement (p_3), and anomaly detection (p_4), were examined thoroughly in this thesis. Additionally, the development of synthetic signal generation framework addressed the lack of representative data.

6.1 Challenge-Specific Results

Data Generation Limitation – SigVarGen: A critical limitation identified early was the lack of representative data, particularly signals characterising embedded systems’ idle and active response states. The developed SigVarGen framework addresses this limitation by synthesising realistic signals with configurable noise, perturbations, and augmentations. Experimental validation demonstrated SigVarGen’s ability to produce statistically representative synthetic datasets closely matching real-world measurements, effectively bridging theoretical research and empirical validation.

p_1 – **Offset & Sampling-Rate Calibration** The [SR&OS](#) algorithm presented in this thesis provides a robust, adaptive methodology for calibrating sampling offsets and rates,

which are critical for accurate trace recording in powertrace-based fuzzing. By combining adaptive binary search, multi-offset evaluation, and sliding-window anomaly scoring, [SR&OS](#) rapidly converges on an optimal $(t_{\text{offset}}, f_{\text{sample}})$ configuration for each target device. Empirical evaluations conducted across ten representative scenarios established the algorithm’s effectiveness and computational efficiency: adaptive binary search achieved an average detection score of 0.6944 compared to 0.4783 for traditional grid search, while requiring dramatically fewer iterations (average 18.6 versus 905) and substantially lower total execution time (average 549 seconds versus 49,943 seconds). Tests conducted across four heterogeneous embedded platforms confirmed it. This contribution ensures the precise capture of critical response intervals despite variable hardware-induced and communication-related delays, forming a robust foundation for reliable downstream anomaly detection.

p_2 – Response-Window Identification The [SR&OS](#) algorithm operates based on an advanced response-detection stage based on sliding-window statistical analysis to dynamically identify the optimal response window length T_{response} . This stage employs [IQR](#)-based outlier tests within a similarity-scanning framework to isolate the most prolonged and stable deviations attributable to command-induced activity. Empirical evaluations demonstrated that this approach robustly separated genuine responses from background noise across diverse embedded systems. Correct response windows were successfully isolated in all but the noisiest GSM scenario, where analyst-in-the-loop validation remained necessary to maintain high detection accuracy, thereby highlighting a practical boundary for full automation.

p_3 – Trace-Quality Enhancement To systematically address trace-quality challenges in powertrace-based fuzzing, we performed a matrix risk assessment evaluating various noise types identified through [SCA](#) literature review and the effectiveness of available mitigation strategies. The comprehensive risk matrix in Table 5.4 reveals that clock jitter, [EMI](#), with power-supply, device and OS noises together account for the majority of critical and high-risk interferences. Guided by the counter-measure ranking in Table 5.7, this thesis proposes Trace Averaging and [SSA](#) as baseline engines, augmented with [DTW](#) for jitter correction and shared-noise subtraction.

Two complementary quality metrics are recommended to decide how many traces must be collected. Proposed [CP](#) metric tracks the marginal power gained between successive averages, and reacts quickly to the diminishing returns characteristic of non-stationary embedded traces. A [CP](#) slope approaching zero signals that additional acquisitions will not substantially raise [SNR](#), allowing early termination. Correlation-based trace predic-

tion techniques estimate the number of traces required for successful side-channel analysis by modelling the relationship between measurement noise, power model variance, and achievable correlation. When adapted carefully, these methods offer theoretically grounded estimations for trace requirements.

p_4 – **Anomaly Detection:** Anomaly classification method in powertrace-based fuzzing should be selected based on the type of denoising or signal preparation metric employed, distinguishing between two cases: classification based on a representative ideal single signal or based on the distribution of traces per command sample. When the denoising strategy produces a stable, representative signal, such as standard averaging combined with elastic alignment, the original T-test method used in the initial PowerFuzzer implementation remains recommended. This approach has been proven effective in practical scenarios and offers strong statistical grounding without introducing unnecessary complexity. Alternative clustering techniques such as K-means applied to raw averaged traces or to extracted feature spaces, were tried but showed neither improvement nor degradation in classification performance relative to the T-test baseline, suggesting limited added value for this application domain. In contrast, when no elastic alignment is performed and variability across traces remains preserved such as after applying [SSA](#), a distribution-preserved T-test or other distributional comparison techniques should be implemented to account for intra-class variance properly.

6.2 Recommendations for Enhancing PowerFuzzer Performance and Robustness

Drawing on the experimental findings and methodological advances developed throughout this thesis, we outline a consolidated roadmap for improving the PowerFuzzer workflow against noise, bias, and data shortage. The recommendations are grouped into three synergistic layers – calibration, signal conditioning, and decision making.

Calibration Layer

- **[SR&OS](#) Integration.** Incorporate SR&OS as the default trigger-offset and sampling-rate search routine. Adaptive binary search and multi-offset evaluation reliably align response windows across heterogeneous targets.

- **Bias-Aware Acquisition.** Employ randomised trace ordering, temperature control, and high-privilege account to minimise systematic drift and OS-induced artefacts before they propagate into later stages.

Signal-Conditioning Layer

- **Adaptive Sample-Size Estimation.** Leverage [CP](#) and correlation-based quality metrics to terminate acquisition once marginal [SNR](#) gains fall below α and pairwise correlation surpasses β thresholds, balancing robustness with runtime cost.
- **Distributional Preprocessing.** Detect and remove statistical outliers to protect averaging and decomposition steps from extreme, non-representative traces.
- **Advanced Noise Removal.**
 - **Shrunk-DBA** to correct jitter and create a representative mean signal.
 - **SSA** to separate structured noise from underlying responses while preserving variance necessary for distributional classifiers.
- **Shared-Noise Identification and Subtraction.** Extract deterministic, repeatable artefacts common to all class traces and subtract them to boost effective [SNR](#).
- **Targeted Cropping.** Remove pre- and post-response regions that do not carry information about the issued command, thereby reducing variance and computational load.

3. Decision-Making Layer

- **Representative-Signal Case.** When elastic alignment and denoising yield a stable mean trace, retain the original T-test used in PowerFuzzer.
- **Distribution-Preserved Case.** If elastic alignment is skipped, classify anomalies with a distribution-preserved T-test.
- **Future Extensions.** To further diversify the classification toolbox, a systematic literature review and empirical benchmarking of existed classification methods for [SCA](#) should be performed.

SigVarGen-Driven Algorithm Development

All algorithmic components above were conceived and stress-tested on extensive SigVarGen datasets that emulate multiple targets and corner cases, then fine-tuned on real PowerFuzzer measurements. This synthetic and empirical data synthesis proved essential for avoiding overfitting to a single device and for revealing edge conditions, such as extreme jitter or random bursts, that would otherwise surface only late in deployment.

6.3 Limitations and Future Work

Despite the significant advancements presented in this thesis, practical limitations remain, particularly concerning data acquisition robustness. While post-acquisition noise mitigation techniques effectively suppress substantial portions of measurement noise, they critically depend on the quality of the initially collected traces.

Random algorithmic noise and unpredictable OS scheduling activities often introduce artefacts that overlap temporally and spectrally with genuine command responses, complicating anomaly detection. Moreover, modern embedded systems increasingly embed purposeful algorithmic noise to hinder side-channel analysis, further limiting the effectiveness of traditional denoising approaches. Scheduled tasks and OS-driven noise patterns frequently exhibit power signatures indistinguishable from legitimate device responses, so simple denoising strategies often fail to address these challenges without introducing classification errors. Therefore, achieving robust system characterisation firstly requires rigorous initial data acquisition practices. Interrupt-driven trace acquisition, executed at the highest possible priority levels, remains essential to minimising asynchronous disturbances. Additionally, adaptive jitter tracking beyond basic elastic alignment, combined with the integration of domain-specific knowledge such as kernel scheduling information into SR&OS scoring metrics, offers a promising path for improving trace alignment and fidelity under adversarial conditions.

Extreme non-stationarity, as observed in platforms such as GSM modules, further motivates the exploration of hybrid analyst-in-the-loop and active-learning approaches. Such methods could dynamically steer acquisition calibration and anomaly detection processes, particularly in highly variable and noisy environments where automated procedures alone may not reliably distinguish between genuine device responses and background artefacts.

References

- [1] Signalz 0.8 documentation. <https://matousc89.github.io/signalz/>, 2017.
- [2] Welcome to pyarbtools’s documentation! <https://pyarbtools.readthedocs.io/en/latest/>, 2018.
- [3] Fourier’s theorem and the spectrum of complex tones. https://mtsu.edu/faculty/wroberts/teaching/fourier_4.php, 2023.
- [4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [5] Dakshi Agrawal, Bruce Archambeault, Josyula R Rao, and Pankaj Rohatgi. The em side—channel (s). In *International workshop on cryptographic hardware and embedded systems*, pages 29–45. Springer, 2002.
- [6] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.
- [7] Theodore Alexandrov. A method of trend extraction using singular spectrum analysis. *arXiv preprint arXiv:0804.3367*, 2008.

- [8] Rahaf Alkhadra, Joud Abuzaid, Mariam AlShammari, and Nazeeruddin Mohammad. Solar winds hack: In-depth analysis and countermeasures. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2021.
- [9] ADC AN118-Improving. Resolution by oversampling and averaging, 2019.
- [10] Arundo Analytics. tsaug — tsaug 0.2.1 documentation. <https://tsaug.readthedocs.io/en/stable/>, 2017.
- [11] Dennis Andrade, Ferran Martorell, Marc Pons, Francesc Moll, and Antonio Rubio. Power supply noise and logic error probability. In *2007 18th European Conference on Circuit Theory and Design*, pages 152–155. IEEE, 2007.
- [12] S. D. Apte. *Fourier Transform Representation of Aperiodic Signals*. Cambridge University Press, 2016.
- [13] S. Arar. Use signal averaging to increase the accuracy of your measurements - technical articles. <https://www.allaboutcircuits.com/technical-articles/use-signal-averaging-to-increase-the-accuracy-of-your-measurements/>, 2023.
- [14] Vipul Arora, Ileana Buhan, Guilherme Perin, and Stjepan Picek. A tale of two boards: On the influence of microarchitecture on side-channel leakage. In *International Conference on Smart Card Research and Advanced Applications*, pages 80–96. Springer, 2021.
- [15] Georg T Becker and Raghavan Kumar. Active and passive side-channel attacks on delay based puf designs. *Cryptology ePrint Archive*, 2014.
- [16] Abhinav Bhatele, Kathryn Mohror, Steven H Langer, and Katherine E Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2013.
- [17] Boualem Boashash and Samir Ouelha. Designing high-resolution time–frequency and time–scale distributions for the analysis and classification of non-stationary signals: a tutorial review with a comparison of features performance. *Digital Signal Processing*, 77:120–152, 2018.
- [18] Poul Bourke. Frequency synthesis of landscapes (and clouds), 1997.

- [19] Giovanni Camurati, Aurélien Francillon, and François-Xavier Standaert. Understanding screaming channels: From a detailed analysis to improved attacks. *IACR transactions on cryptographic hardware and embedded systems*, (3), 2020.
- [20] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 163–177, 2018.
- [21] Patrick Celka and Elly Gysels. Smoothly adjustable denoising using a priori knowledge. *Signal Processing*, 86(9):2233–2242, 2006.
- [22] Xavier Charvet and Herve Pelletier. Improving the dpa attack using wavelet transform. In *NIST Physical Security Testing Workshop*, volume 46, pages 1–15, 2005.
- [23] Jingdong Chen, Jacob Benesty, Yiteng Huang, and Simon Doclo. New insights into the noise reduction wiener filter. *IEEE Transactions on audio, speech, and language processing*, 14(4):1218–1234, 2006.
- [24] Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. There’s always a bigger fish: A clarifying analysis of a machine-learning-assisted side-channel attack. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 204–217, 2022.
- [25] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
- [26] João Carlos GD Costa, Paulo José G Da-Silva, Renan Moritz VR Almeida, and Antonio Fernando C Infantosi. Validation in principal components analysis applied to eeg data. *Computational and mathematical methods in medicine*, 2014(1):413801, 2014.
- [27] Paul Cuffe, Harold Kirkham, Chris Dent, and Amy Wilson. Data visualization: The signal and the noise. *IEEE Potentials*, 37(4):28–34, 2018.
- [28] M. K. Delimayanti et al. Classification of brainwaves for sleep stages by high-dimensional fft features from eeg signals. *Applied Sciences*, 10(5):1797, 2020.
- [29] Manoj Diwakar and Manoj Kumar. A review on ct image noise and its denoising. *Biomedical Signal Processing and Control*, 42:73–88, 2018.

- [30] Andrzej Drygajlo and Rudolf Haraksim. Biometric evidence in forensic automatic speaker recognition. *Handbook of Biometrics for Forensic Science*, pages 221–239, 2017.
- [31] Ruo-Nan Duan, Jia-Yi Zhu, and Bao-Liang Lu. Differential entropy feature for eeg-based emotion classification. In *2013 6th international IEEE/EMBS conference on neural engineering (NER)*, pages 81–84. IEEE, 2013.
- [32] Murray Dunne and Sebastian Fischmeister. Powertrace-based fuzzing of can connected hardware. In *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 239–244. IEEE, 2022.
- [33] Salvador Dura-Bernal, Thomas Wennekers, and Susan L Denham. Top-down feedback in an hmax-like cortical model of object perception based on hierarchical bayesian networks and belief propagation. *PloS one*, 7(11):e48216, 2012.
- [34] Max Eisele, Marcello Maugeri, Rachna Shriwas, Christopher Huth, and Giampaolo Bella. Embedded fuzzing: a review of challenges, tools, and solutions. *Cybersecurity*, 5(1):18, 2022.
- [35] Ericsson. How to stop side channel analysis attacks, April 2023. Accessed: 2025-04-15.
- [36] Sankaralingam Esakkirajan, Thangaraj Veerakumar, and Badri N Subudhi. *Digital Signal Processing: Illustration Using Python*. Springer Nature Singapore, 2024.
- [37] Everything RF. Lte frequency bands. <https://www.everythingrf.com/community/lte-frequency-bands>, 2024. Accessed: 2024-02-04.
- [38] Johann Faouzi and Hicham Janati. pyts: A python package for time series classification. *Journal of Machine Learning Research*, 21(46):1–6, 2020.
- [39] Mingliang Feng, Yongbin Zhou, and Zhenmei Yu. Emd-based denoising for side-channel attacks and relationships between the noises extracted with different denoising methods. In *Information and Communications Security: 15th International Conference, ICICS 2013, Beijing, China, November 20-22, 2013. Proceedings 15*, pages 259–274. Springer, 2013.
- [40] Dor Fledel and Avishai Wool. Sliding-window correlation attacks against encryption devices with an unstable clock. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*, pages 193–215. Springer, 2019.

- [41] Lorenzo Galleani and Leon Cohen. Nonstationary and stationary noise. In *Automatic Target Recognition XVI*, volume 6234, pages 360–384. SPIE, 2006.
- [42] Yiwen Gao and Yongbin Zhou. Side-channel attacks with multi-thread mixed leakage. *IEEE Transactions on Information Forensics and Security*, 16:770–785, 2020.
- [43] Cesar Pereida García, Sohaib Ul Hassan, Nicola Tuveri, Iaroslav Gridin, Alejandro Cabrera Aldaya, and Billy Bob Brumley. Certified side channels. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2021–2038, 2020.
- [44] Catherine H Gebotys and Catherine H Gebotys. Side channel attacks on the embedded system. *Security in Embedded Devices*, pages 163–222, 2010.
- [45] Zoltan Geler, Vladimir Kurbalija, Mirjana Ivanović, Miloš Radovanović, and Weihui Dai. Dynamic time warping: Itakura vs sakoe-chiba. In *2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)*, pages 1–6. IEEE, 2019.
- [46] Theodoros Giannakopoulos and Aggelos Pikrakis. Chapter 4 - audio features. In Theodoros Giannakopoulos and Aggelos Pikrakis, editors, *Introduction to Audio Analysis*, pages 59–103. Academic Press, Oxford, 2014.
- [47] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.
- [48] Ognjen Glamočanin, Hajira Bazaz, Mathias Payer, and Mirjana Stojilović. Temperature impact on remote power side-channel attacks on shared fpgas. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.
- [49] Gurfinkel and Kuderski. Automated test-case generation: Fuzzing. *Testing, Quality Assurance, and Maintenance*, Fall 2023.
- [50] Xueyun Han, Xiaosong Li, Yingping Zhou, Pengfei Zheng, Haohua Cui, and Xiaonan Zhang. Microstrip bandstop filter for preventing conduction electromagnetic information leakage of high-power transmission line. *International Journal of Antennas and Propagation*, 2022(1):4915492, 2022.
- [51] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg,

- Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [52] Thomas Haslwanter. *Hands-on signal analysis with python*. Springer, 2021.
 - [53] Annelie Heuser, Michael Kasper, Werner Schindler, and Marc Stöttinger. A new difference method for side-channel analysis with high-dimensional leakage models. In *Topics in Cryptology–CT-RSA 2012: The Cryptographers’ Track at the RSA Conference 2012, San Francisco, CA, USA, February 27–March 2, 2012. Proceedings*, pages 365–382. Springer, 2012.
 - [54] J Hewitt, AE Gelfand, NJ Quick, WR Cioffi, BL Southall, SL DeRuiter, and RS Schick. Kernel density estimation of conditional distributions to detect responses in satellite tag data. *anim biotelem*. 2022; 10 (1): 28.
 - [55] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers 12*, pages 219–235. Springer, 2014.
 - [56] NewAE Technology Inc. Tips and tricks. <https://rtfm.newae.com/Tips%20and%20Tricks>. Accessed: 2025-04-15.
 - [57] Intel. 2.4 ghz vs. 5 ghz vs. 6 ghz: What’s the difference? <https://www.intel.com/content/www/us/en/products/docs/wireless/2-4-vs-5ghz.html>, 2024.
 - [58] Rashid Jahangir, Ying Wah Teh, Henry Friday Nweke, Ghulam Mujtaba, Mohammed Ali Al-Garadi, and Ihsan Ali. Speaker identification through artificial intelligence techniques: A comprehensive review and research challenges. *Expert Systems with Applications*, 171:114591, 2021.
 - [59] Robert Jenke, Angelika Peer, and Martin Buss. Feature extraction and selection for emotion recognition from eeg. *IEEE Transactions on Affective computing*, 5(3):327–339, 2014.
 - [60] Ruzica Jevtic, Pablo Perez-Tirador, Carmen Cabezaolias, Pablo Carnero, and Gabriel Caffarena. Side-channel attack countermeasure based on power supply modulation. In *2022 30th European Signal Processing Conference (EUSIPCO)*, pages 618–622. IEEE, 2022.

- [61] Morgan Jones. Chapter 6 - the power amplifier. In Morgan Jones, editor, *Valve Amplifiers (Fourth Edition)*, pages 435–546. Newnes, Oxford, fourth edition edition, 2012.
- [62] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO’99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.
- [63] Sang Gyu Kwak and Jong Hae Kim. Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144, 2017.
- [64] Thanh-Ha Le, Jessy Clédière, Christine Servièrè, and Jean-Louis Lacoume. Noise reduction in side channel attack using fourth-order cumulant. *IEEE Transactions on Information Forensics and Security*, 2(4):710–720, 2007.
- [65] Saksorn Limwichean, Wipawanee Leung, Pemika Sataporncha, Nongluck Hounkamhang, On-Uma Nimittrakoolchai, Bunpot Saekow, Tawee Pogfay, Pacharamon Somboonsaksri, Jia Yi Chia, Raju Botta, et al. Label free detection of multiple trace antibiotics with sers substrates and independent components analysis. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy*, 295:122584, 2023.
- [66] JR Maat, A Malali, and P Protopapas. Timesynth: A multipurpose library for synthetic time series in python. 2017.
- [67] Houssem Maghrebi and Emmanuel Prouff. On the use of independent component analysis to denoise side-channel measurements. In *Constructive Side-Channel Analysis and Secure Design: 9th International Workshop, COSADE 2018, Singapore, April 23–24, 2018, Proceedings 9*, pages 61–81. Springer, 2018.
- [68] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.
- [69] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.
- [70] Kavish Mehta. Exploring power fuzzing in embedded systems: Architecture, challenges, and enhancements. 2024.

- [71] Santos Merino Del Pozo and François-Xavier Standaert. Blind source separation from single measurements using singular spectrum analysis. In *Cryptographic Hardware and Embedded Systems—CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings 17*, pages 42–59. Springer, 2015.
- [72] Nadica Miljković, Nikola Milenić, Nenad B Popović, and Jaka Sodnik. Data augmentation for generating synthetic electrogastrogram time series. *Medical & Biological Engineering & Computing*, 62(9):2879–2891, 2024.
- [73] Dalibor Mitrović, Matthias Zeppelzauer, and Christian Breiteneder. Chapter 3 - features for content-based audio retrieval. In *Advances in Computers: Improving the Web*, volume 78 of *Advances in Computers*, pages 71–150. Elsevier, 2010.
- [74] Amir Moradi. *Advances in side-channel security*. PhD thesis, Bochum, Ruhr-Universität Bochum, Habil.-Schr., 2015, 2016.
- [75] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.
- [76] Pouya Narimani, Meng Wang, Ulysse Planta, and Ali Abbasi. Exploring power side-channel challenges in embedded systems security. *arXiv preprint arXiv:2410.11563*, 2024.
- [77] Alexander Nikitin, Letizia Iannucci, and Samuel Kaski. Tsgm: A flexible framework for generative modeling of synthetic time series. *arXiv preprint arXiv:2305.11567*, 2023.
- [78] Mabon Ninan, Evan Nimmo, Shane Reilly, Channing Smith, Wenhai Sun, Boyang Wang, and John M Emmert. A second look at the portability of deep learning side-channel attacks over em traces. In *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 630–643, 2024.
- [79] NIST. Signal to noise ratio, Jun 2017.
- [80] J O’Brien and B Gould. Achieve higher vertical resolution for more precise measurements. *White Paper, Tektronix*, 2020.
- [81] Colin O’Flynn and Zhizhang Chen. Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. *Journal of Cryptographic Engineering*, 5:53–69, 2015.

- [82] Victor M Panaretos and Yoav Zemel. Statistical aspects of wasserstein distances. *Annual review of statistics and its application*, 6(1):405–431, 2019.
- [83] Kostas Papagiannopoulos, Ognjen Glamočanin, Melissa Azouaoui, Dorian Ros, Francesco Regazzoni, and Mirjana Stojilović. The side-channel metrics cheat sheet. *ACM Computing Surveys*, 55(10):1–38, 2023.
- [84] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [85] Joshua Pereyda. boofuzz documentation. 2019.
- [86] François Petitjean, Germain Forestier, Geoffrey I Webb, Ann E Nicholson, Yanping Chen, and Eamonn Keogh. Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. *Knowledge and Information Systems*, 47:1–26, 2016.
- [87] François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern recognition*, 44(3):678–693, 2011.
- [88] Fabrizio Petrini, Darren J Kerbyson, and Scott Pakin. The case of the missing supercomputer performance. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing (SC)*, 2003.
- [89] Joel Phillips and Ken Kundert. An introduction to cyclostationary noise. *The Designer’s Guide community*, 2000.
- [90] Paul Pokotylo. Advancements in synthetic data generation techniques.
- [91] Chethan Ramesh, Shivukumar B Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. Fpga side channel attacks without physical access. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52. IEEE, 2018.
- [92] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):461–491, 2004.

- [93] Fabrizio Romanelli and Francesco Martinelli. Synthetic sensor measurement generation with noise learning and multi-modal information. *IEEE Access*, 11:111765–111788, 2023.
- [94] J. Romero. What frequency do drones use? a comprehensive guide 2023. <https://droneguider.com/what-frequency-do-drones-use/>, 2023. Accessed: 2024-01-16.
- [95] Dor Salomon, Amir Weiss, and Itamar Levi. Improved filtering techniques for single- and multi-trace side-channel analysis. *Cryptography*, 5(3):24, 2021.
- [96] Q. Samples et al. What frequency is bluetooth. <https://robots.net/tech/what-frequency-is-bluetooth>, 2024. Accessed: 2024-01-16.
- [97] Sylvain Sardy, Paul Tseng, and Andrew Bruce. Robust wavelet denoising. *IEEE transactions on signal processing*, 49(6):1146–1152, 2001.
- [98] Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. Leveraging electromagnetic side-channel analysis for the investigation of iot devices. *Digital Investigation*, 29:S94–S103, 2019.
- [99] Siddharth Shekar, Chen-Chi Chien, Andreas Hartel, Peijie Ong, Oliver B Clarke, Andrew Marks, Marija Drndic, and Kenneth L Shepard. Wavelet denoising of high-bandwidth nanopore and ion-channel signals. *Nano letters*, 19(2):1090–1097, 2019.
- [100] Wanyong Shin, Paul Taylor, and Mark J Lowe. Estimation and removal of residual motion artifact in retrospectively motion-corrected fmri data: A comparison of inter-volume and intravolume motion using gold standard simulated motion data. *Aperture Neuro*, 4, 2024.
- [101] Petr Socha, Vojtěch Miškovský, and Martin Novotný. A comprehensive survey on the non-invasive passive side-channel analysis. *Sensors*, 22(21):8096, 2022.
- [102] Youssef Souissi, M Abdelaziz Elaabid, Nicolas Debande, Sylvain Guilley, and Jean-Luc Danger. Novel applications of wavelet transforms based side-channel analysis. In *Non-Invasive Attack Testing Workshop*, 2011.
- [103] Youssef Souissi, Sylvain Guilley, Jean-luc Danger, Sami Mekki, and Guillaume Duc. Improvement of power analysis attacks using kalman filter. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1778–1781. IEEE, 2010.

- [104] Philip Sperl and Konstantin Böttinger. Side-channel aware fuzzing. In *Computer Security—ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24*, pages 259–278. Springer, 2019.
- [105] F X Standaert, Eric Peeters, Cédric Archambeau, and J J Quisquater. Towards security limits in side-channel attacks: With an application to block ciphers. In *Cryptographic Hardware and Embedded Systems—CHES 2006: 8th International Workshop, Yokohama, Japan, October 10–13, 2006. Proceedings 8*, pages 30–45. Springer, 2006.
- [106] Petre Stoica, Randolph L Moses, et al. *Spectral analysis of signals*, volume 452. Citeseer, 2005.
- [107] Yifan Sun. Denoising of power traces in side-channel analysis, 2024.
- [108] Kevin T Sweeney, Tomás E Ward, and Seán F McLoone. Artifact removal in physiological signals—practices and possibilities. *IEEE transactions on information technology in biomedicine*, 16(3):488–500, 2012.
- [109] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, and Eli Woods. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6, 2020.
- [110] D Terlep. How quantization and thermal noise determine an adc’s effective noise figure, 2002.
- [111] Michael E Tipping and Christopher M Bishop. Mixtures of probabilistic principal component analyzers. *Neural computation*, 11(2):443–482, 1999.
- [112] Shahid Ul Haq, Yashwant Singh, Amit Sharma, Rahul Gupta, and Dipak Gupta. A survey on iot & embedded device firmware security: architecture, extraction techniques, and vulnerability analysis frameworks. *Discover Internet of Things*, 3(1):17, 2023.
- [113] Wim Van Drongelen. *Signal processing for neuroscientists*. Academic press, 2018.
- [114] Jasper GJ Van Woudenberg, Marc F Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In *Topics in Cryptology—CT-RSA 2011: The Cryptographers’ Track at the RSA Conference 2011, San Francisco, CA, USA, February 14–18, 2011. Proceedings*, pages 104–119. Springer, 2011.

- [115] Saeed V Vaseghi. *Advanced digital signal processing and noise reduction*. John Wiley & Sons, 2008.
- [116] Robert Vautard, Pascal Yiou, and Michael Ghil. Singular-spectrum analysis: A toolkit for short, noisy chaotic signals. *Physica D: Nonlinear Phenomena*, 58(1-4):95–126, 1992.
- [117] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
- [118] Valeriu Vrabie, Pierre Granjon, and Christine Serviere. Spectral kurtosis: from definition to application. In *6th IEEE international workshop on Nonlinear Signal and Image Processing (NSIP 2003)*, page xx, 2003.
- [119] Peter Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on audio and electroacoustics*, 15(2):70–73, 2003.
- [120] IR Young. The determination of confidence limits associated with estimates of the spectral peak frequency. *Ocean engineering*, 22(7):669–686, 1995.
- [121] Zigbee Alliance. Zigbee faq. <http://www.zigbee.org/Specifications/ZigBee/FAQ.aspx>, 2024. Accessed: 2024-01-16.

APPENDICES

Appendix A

SigVarGen: KDE Plots

This appendix presents the distributional comparisons between synthetic signals generated by SigVarGen v0.1 and real measurement data collected from various embedded devices, including Raspberry Pi, PhiHue, HikCam, and GSM-based platforms. Each figure shows the estimated probability density function for a specific feature using KDE, allowing visual inspection of the alignment between synthetic and empirical signal statistics.

Figure A.1 shows the distribution of the mean Euclidean distance between sequential signals, capturing global structural variability.

Figure A.2 presents the standard deviation of Euclidean distance values, reflecting how much the variation magnitude fluctuates between signal pairs.

Figure A.3 illustrates the skewness of the Euclidean distance distribution, indicating asymmetry in variation magnitudes.

Figure A.4 compares the kurtosis of Euclidean distances, showing the prevalence of extreme deviations in each dataset.

Figure A.5 visualises lag-1 autocorrelation in Euclidean distance sequences, revealing temporal smoothness or volatility.

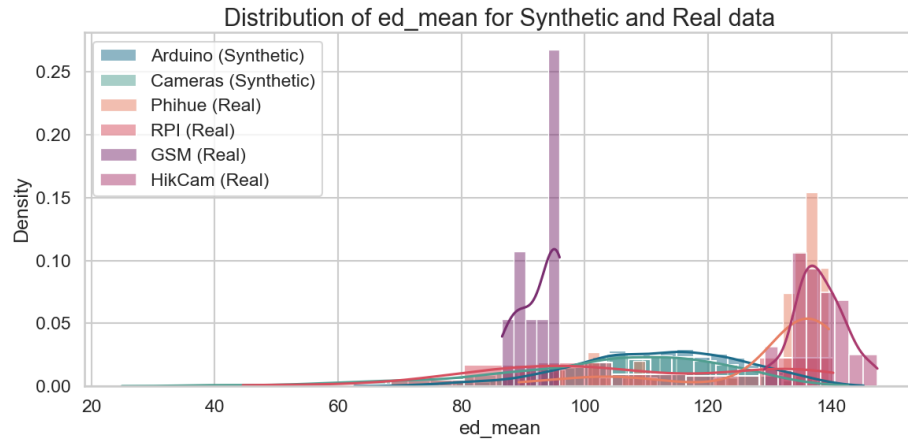


Figure A.1: Distribution of `ed_mean` for synthetic and real data

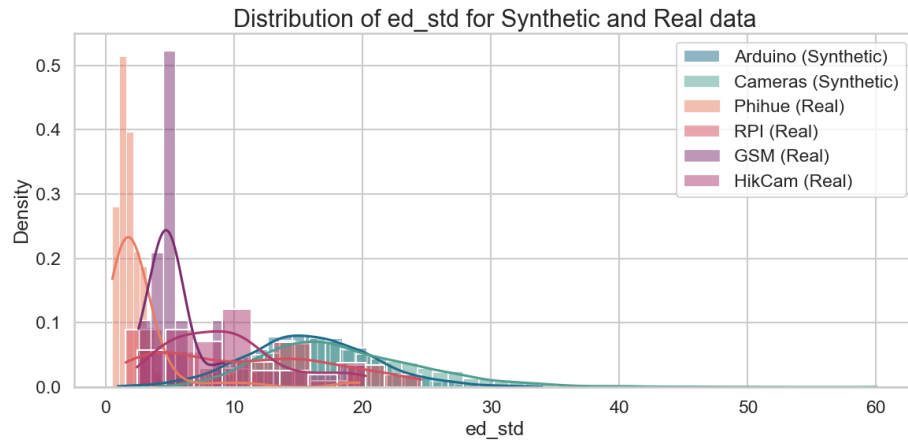


Figure A.2: Distribution of `ed_std` for synthetic and real data

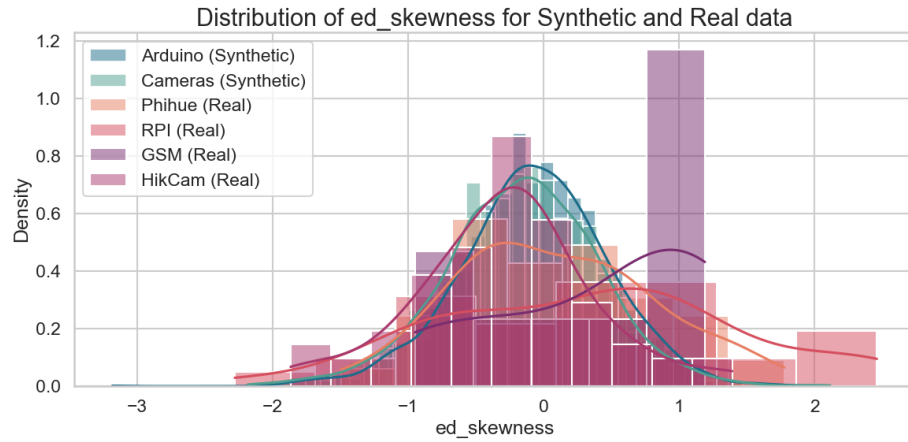


Figure A.3: Distribution of `ed_skewness` for synthetic and real data

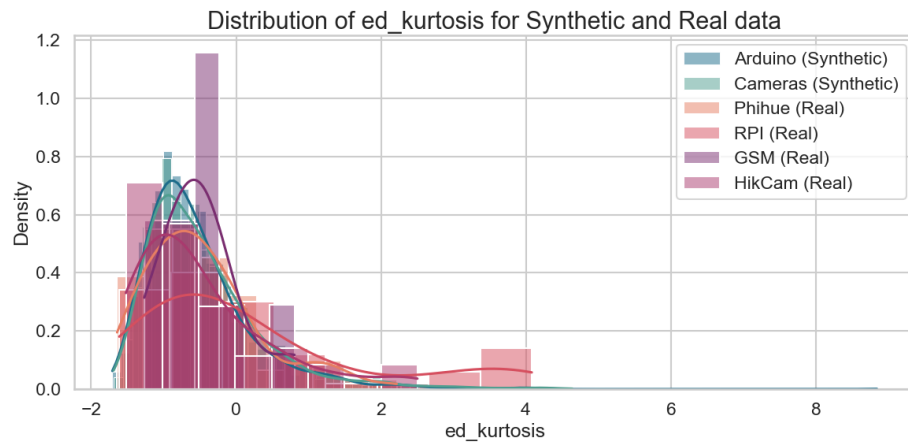


Figure A.4: Distribution of `ed_kurtosis` for synthetic and real data

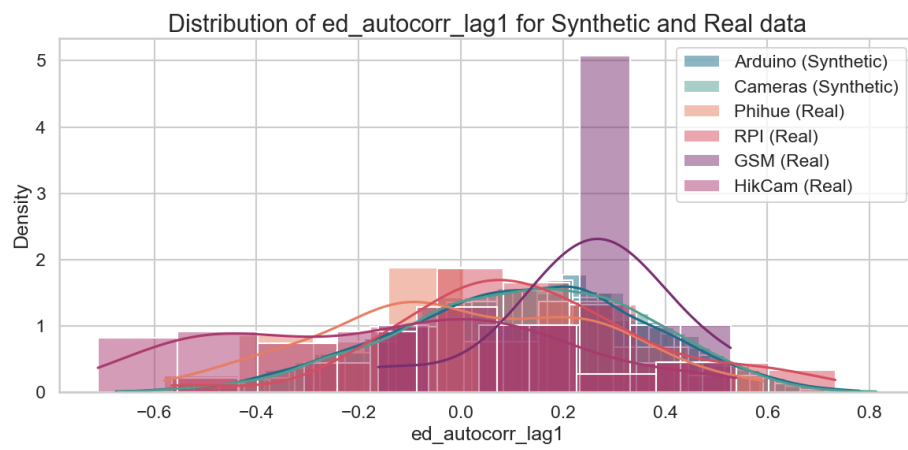


Figure A.5: Distribution of `ed_autocorr_lag1` for synthetic and real data

Appendix B

SigVarGen: ED-CP Pair Comparisons

This appendix presents the set of scatter plots comparing pairwise relationships between statistical features extracted from real and SigVarGen-generated signal variations. These visualisations complement the evaluation in Chapter 3.6.2.

The figures in this appendix display pairwise comparisons between features derived from [ED](#) and [CP](#) metrics. Synthetic data points are colour-coded by their corresponding [SNR](#), while real data are grouped by device (PhiHue, RPI, HikCam, GSM). Each plot highlights the statistical alignment or divergence between synthetic and empirical feature spaces, offering a clear assessment of the realism achieved by the SigVarGen signal generation framework.

Figure [B.1](#) illustrates relationships between incremental statistical features such as mean, standard deviation, kurtosis, and skewness for [ED](#) and [CP](#). These comparisons assess the realism of signal-to-signal variability patterns.

Figure [B.2](#) presents fundamental signal variation features, comparing average values and spread across the synthetic and real datasets.

Figure [B.3](#) captures higher-order statistical relationships, revealing how well the synthetic signals emulate the tail behaviour and asymmetry of actual signal variation distributions.

Figure [B.4](#) focuses on time-series structure and frequency-domain power. It visualises autocorrelation at lag 1 and the mean [PSD](#).

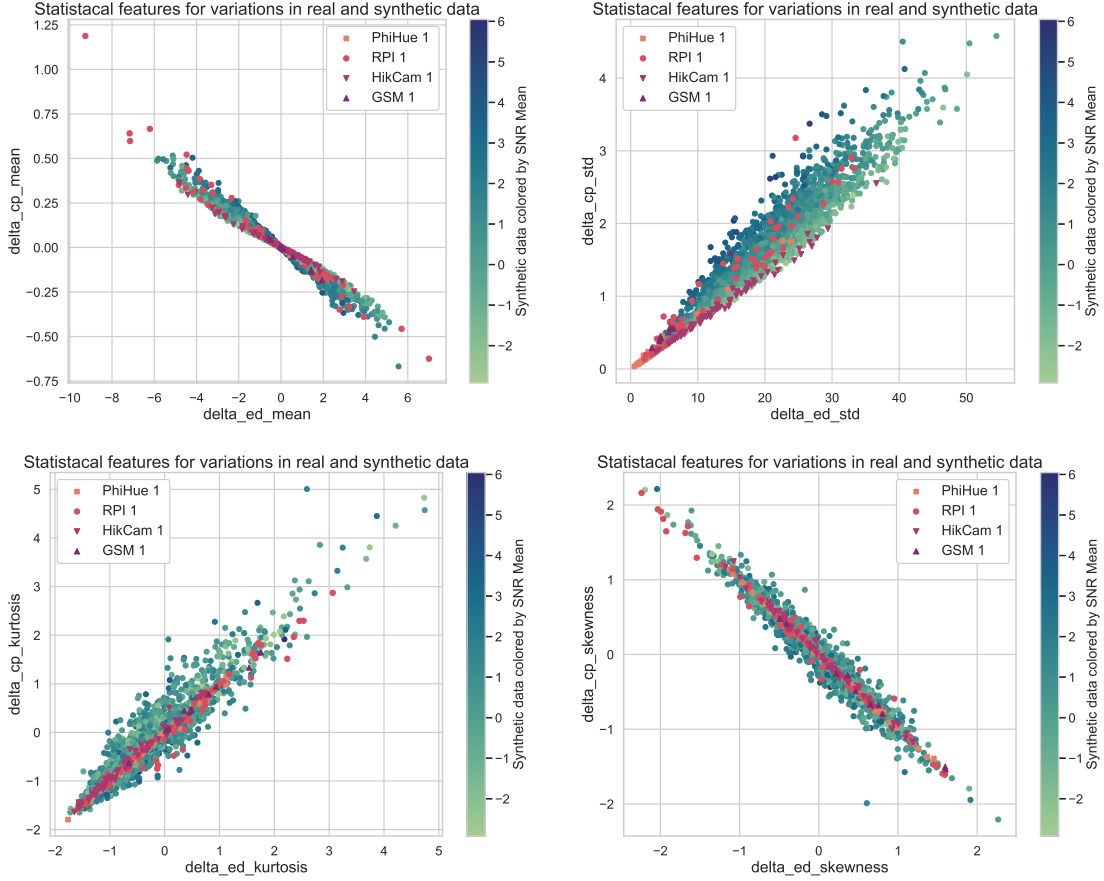


Figure B.1: Pairwise scatter plots comparing incremental (Δ) ED-based and CP-based statistical features (mean, standard deviation, kurtosis, and skewness) for synthetic and real embedded-device signals. Synthetic data points are coloured by mean SNR.

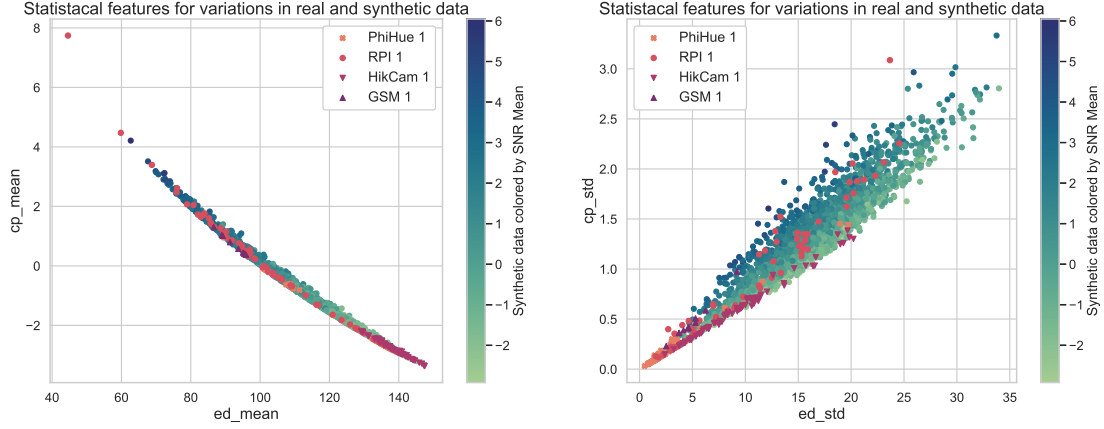


Figure B.2: Scatter plots comparing mean and standard deviation of ED and CP sequences for synthetic and real embedded-device measurements. The colour scale for synthetic data points reflects the mean SNR values.

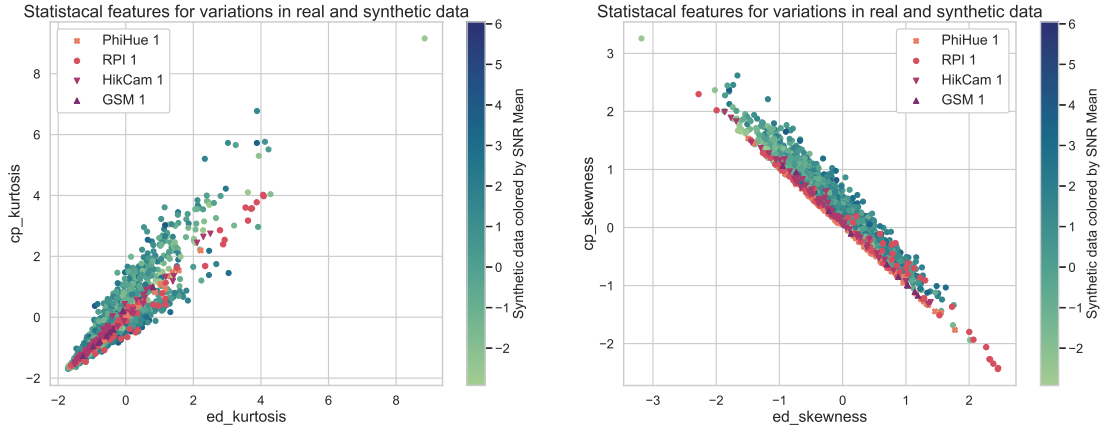


Figure B.3: Scatter plots depicting relationships between kurtosis and skewness derived from ED and CP sequences. Synthetic signals demonstrate realistic alignment with real embedded-device data distributions.

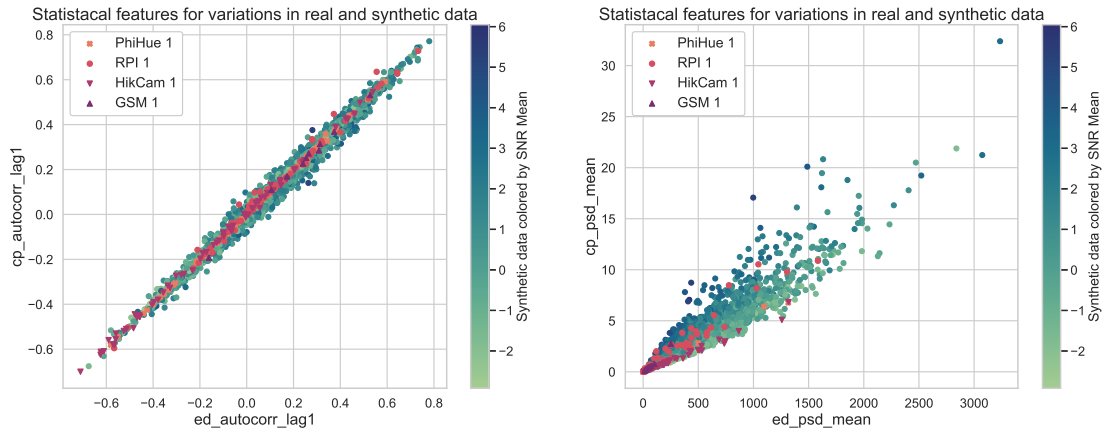


Figure B.4: Scatter plots for spectral (PSD mean) and temporal dependency (autocorrelation lag-1) features derived from ED and CP sequences, comparing synthetic and real embedded-device data. Synthetic data points are color-coded according to their mean SNR values.

Appendix C

SR&OS Algorithm Flow Charts

This appendix presents the detailed flowcharts corresponding to the [SR&OS](#) algorithm from Chapter 4, illustrating key processes involved in response trace identification and optimal parameter search using binary search. These diagrams serve as a complementary visual aid to the methodology chapter, providing a structured representation of the algorithm’s decision-making logic, iterative search process, and response window refinement.

Figure [C.1](#) illustrates the process of identifying response traces within power measurements using sliding window comparison and outlier detection.

Figure [C.2](#) represents the first phase of the binary search optimisation process for determining the optimal sampling rate and offset configuration.

Figure [C.3](#) depicts the continuation of the search process, including adaptive offset adjustments and termination conditions.

Figure [C.4](#) illustrates the iterative multi-offset evaluation process, where multiple offsets are systematically tested using binary search ‘search_optimal_params’.

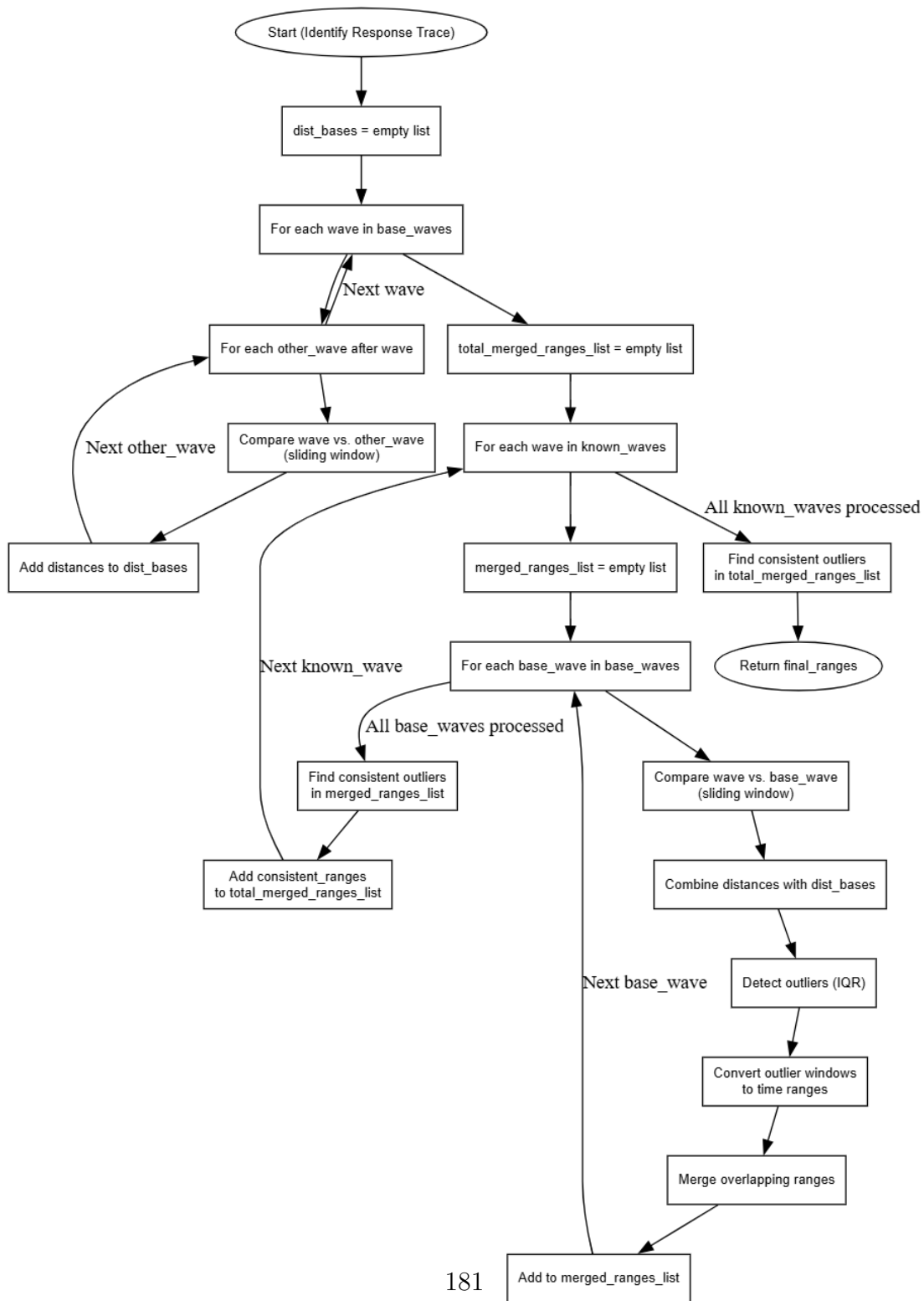


Figure C.1: Identify response trace

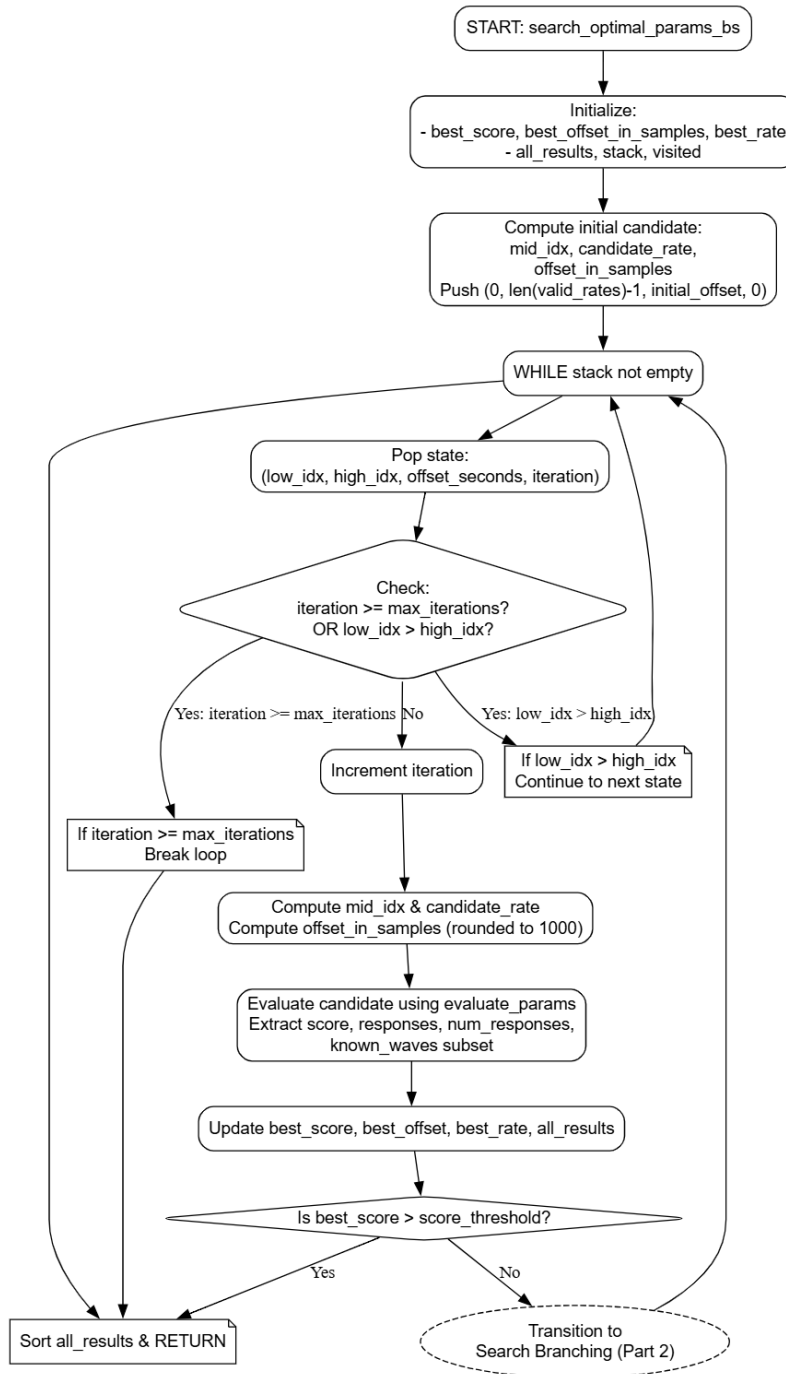


Figure C.2: Search optimal params binary search, part 1

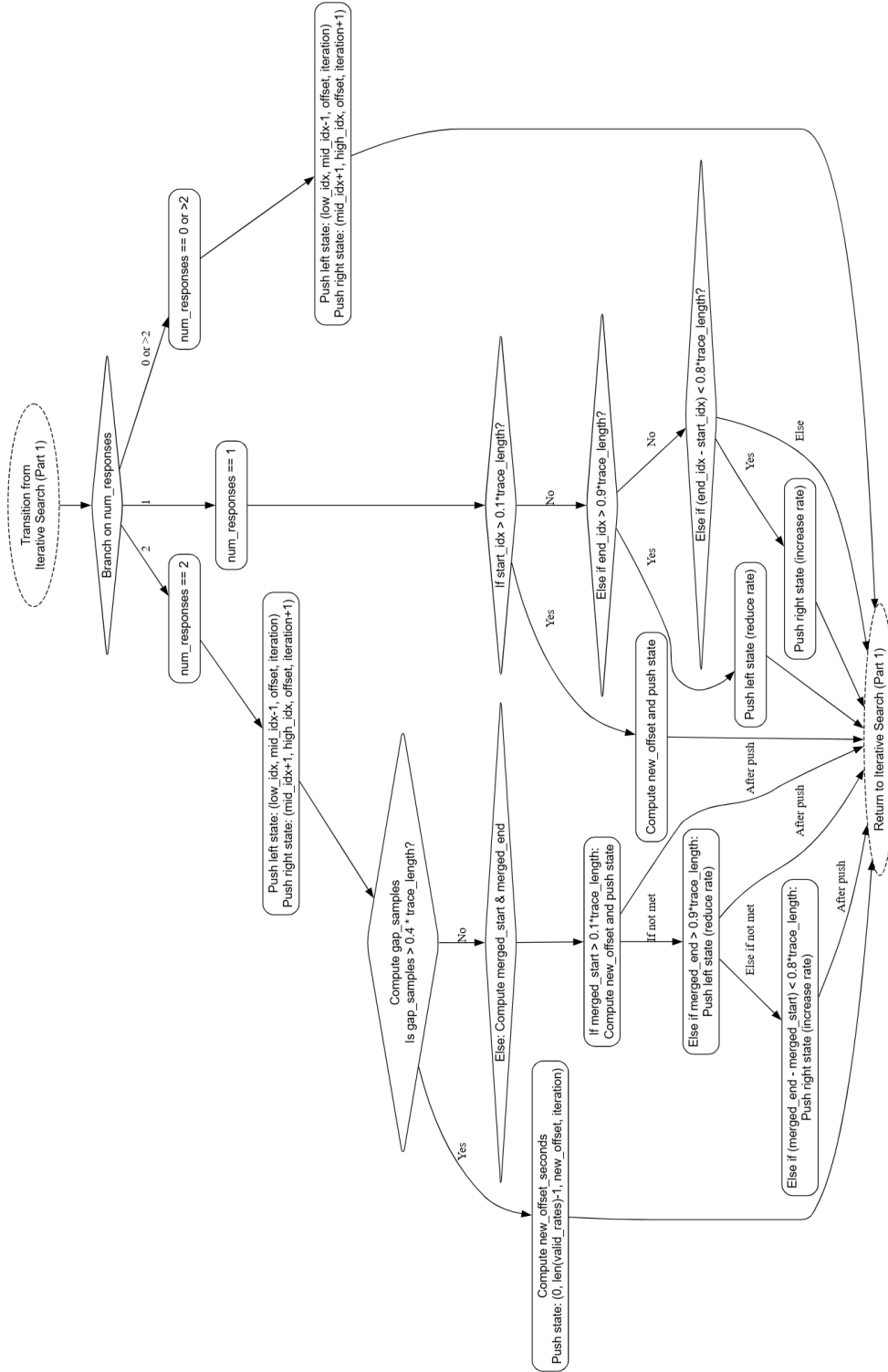


Figure C.3: Search optimal¹⁸³ params binary search, part 2

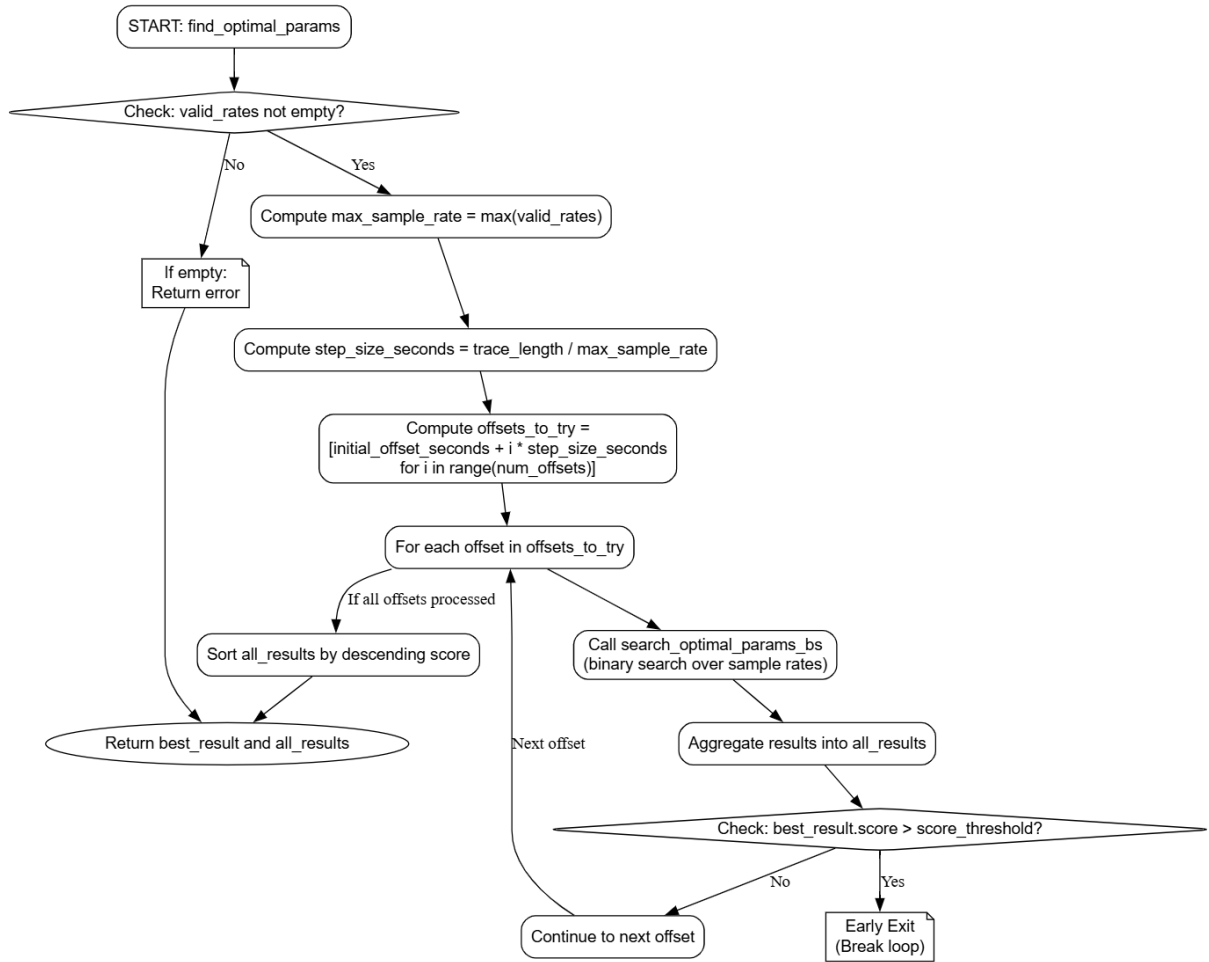


Figure C.4: Multi-offset evaluation