Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# Function Recognition from Raw Power Side Channels in Bare-Metal Embedded Systems: Feasibility and Limits

**Mohammed Mezaouli**
IMT Atlantique
**Yehya Nasser**
IMT Atlantique
**Samir Saudi**
IMT Atlantique
**Marc-Oliver Pahl**
marc-oliver.pahl@imt-atlantique.fr

IMT Atlantique

**Research Article**

**Keywords:** Power Side Channels, Embedded Systems Security, Bare-Metal Embedded Systems, Function Recognition, Runtime Monitoring, IoT Security

**Posted Date:** November 10th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-11272948/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** No competing interests reported.

# Function Recognition from Raw Power Side Channels in Bare-Metal Embedded Systems: Feasibility and Limits

Mohammed Mezaouli [1,2], Yehya Nasser [1,2], Samir Saoudi [1,2], Marc-Oliver Pahl [1,3,4*]

[1]IMT Atlantique, Brest, France.
[2]Lab-STICC, UMR CNRS 6285, Brest, France.
[3]IRISA, UMR CNRS 6074, Rennes, France.
[4]Chaire Cybersecurity of Critical National Infrastructures (Cyber CNI), France.

*Corresponding author(s). E-mail(s):
marc-oliver.pahl@imt-atlantique.fr;
Contributing authors: mohammed.mezaouli@imt-atlantique.fr;
yehya.nasser@imt-atlantique.fr; samir.saoudi@imt-atlantique.fr;

**Abstract**

Bare-metal microcontrollers underpin resource-constrained IoT, industrial, and cyber-physical systems, yet their lack of operating-system-level instrumentation limits runtime security visibility. This raises a fundamental question: how much security-relevant execution behavior can be recovered independently from the physical signals exposed by the device?

We investigate whether selected C functions can be recognized directly from raw power traces of continuous, unmodified bare-metal workloads. Using an ARM Cortex-M4 platform and ChipWhisperer for power acquisition, we compare Transformer- and LSTM-based sequence-classification baselines with a parallel architecture of class-specific autoencoders operating directly on fixed-size power windows. The latter learns a separate reconstruction model and decision threshold for each function, enabling unrelated workload segments to be rejected rather than forcibly classified.

For five retained functions (`memcpy`, `memset`, `memmove`, `strncpy`, and `strncat`), class-specific reconstruction provides the best observed operating point, achieving 96% TPR and 4% FNR at a 39.79% FPR. Across all evaluated architectures,

`malloc` and `strcpy` remain difficult to distinguish because of, respectively, low intra-function signal variation and high inter-function similarity.
These results establish both the feasibility and the physical limits of raw-power function recognition. Rather than a standalone vulnerability detector, the approach provides an external behavioral signal that can complement software-derived evidence in trustworthy embedded-system monitoring.

**Keywords:** Power Side Channels; Embedded Systems Security; Bare-Metal Embedded Systems; Function Recognition; Runtime Monitoring; IoT Security

# Statements and Declarations

## Funding

This research was co-funded by the French Defense Procurement Agency (DGA) and ARED Région Bretagne under the CREACH LABS label (Convention No. 23 81 0068), and by the Chaire Cyber CNI.

## Competing Interests

The authors have no relevant financial or non-financial interests to disclose.

## Author Contributions

Mohammed Mezaouli: Methodology, Investigation, Formal analysis, Writing – original draft, Writing – review & editing. Yehya Nasser: Conceptualization, Methodology, Formal analysis, Writing – original draft, Supervision, Funding acquisition, Writing – review & editing. Samir Saoudi: Supervision. Marc-Oliver Pahl: Conceptualization, Methodology, Writing – original draft, Supervision, Funding acquisition, Writing – review & editing. All authors reviewed and approved the final manuscript.

## Data Availability

The datasets generated and analysed during the current study are not publicly available but are available from the corresponding author on reasonable request.

# 1 Introduction

Bare-metal systems based on ARM Cortex-M microcontrollers are widely deployed in resource-constrained IoT devices, automotive controllers [1], and industrial systems [2]. Their direct interaction with hardware provides efficient and deterministic execution but limits the runtime visibility otherwise available through operating-system services or software instrumentation. Memory-related C functions such as `memcpy` and `memset` frequently occur in code affected by memory-safety weaknesses [3, 4]. Observing their execution does not establish that an invocation is unsafe or malicious, but can provide independent behavioral evidence to a security-monitoring system.

Physical side channels offer an external source of such evidence: power and electromagnetic leakage have been used to infer software activities, functions, instructions, basic blocks, and control flow [5–9]. However, comparatively little is known about recognizing selected source-level C functions directly from raw power windows in continuous bare-metal workloads, without engineered spectral features or workload modification at inference time, and about the physical limits of such recognition.

We investigate this problem on ARM Cortex-M4 systems and formulate it as *function-level behavioral monitoring*: recognizing selected security-relevant C functions from externally acquired raw power traces. The monitor is passive and independent of software instrumentation; we assume trusted measurement hardware and no active manipulation of the measured signal. The task is function recognition rather than vulnerability or exploit detection, which would additionally require information such as arguments, memory bounds, data provenance, or control-flow context.

We address three research questions:

- **RQ1:** To what extent can selected security-relevant C functions be recognized from raw power traces in continuous, bare-metal workloads?
- **RQ2:** How do recognition architecture and acquisition parameters affect the sensitivity–false-alarm trade-off?
- **RQ3:** Which execution- and signal-level properties limit the distinguishability of function signatures?

Our contributions are:

1. **Raw-power function recognition.** We recognize selected security-relevant C functions directly from fixed-size raw power windows in continuous bare-metal workloads, without software instrumentation or spectral feature engineering.
2. **Class-specific reconstruction.** We compare Transformer and LSTM-based sequence-classification baselines with parallel class-specific autoencoders. The latter provide the best observed sensitivity–false-alarm trade-off on the independent continuous workloads.
3. **Physical observability limits.** We characterize the effects of acquisition parameters and function-specific signal properties, showing that the persistent difficulty of `malloc` and `strcpy` across architectures is consistent with limits in the information exposed by the measured signal, rather than model choice alone.

Together, these contributions characterize both the feasibility and the limits of raw-power function recognition as an independent behavioral signal for trustworthy monitoring of resource-constrained embedded systems.

## 2 Related Work

Physical side channels have been used to infer software execution at multiple granularities, including activities, functions, instructions, basic blocks, control flow, and operation boundaries. Prior approaches nevertheless differ substantially in their inference targets, profiling assumptions, signal representations, and execution conditions.

***Function and activity recognition.***

The closest function-level precedent is Chen et al. [5], who recognize MCU functional classes from near-field EM leakage using frequency-domain constellation images and a CNN, with GPIO markers used to align and crop function cycles. Robyns et al. [8] classify and temporally localize cryptographic operations from filtered or time–frequency EM representations, while Le et al. [10] classify IoT software activities and demonstrate the sensitivity of recognition to signal representation and measurement conditions. Jensen et al. [9] provide a particularly relevant raw-window precedent, using short raw power and EM segments to classify execution-block categories rather than named source-level functions. Together, these works establish physical-signal recognition of software activity, but under different label granularities, representations, or acquisition assumptions than considered here.

***Execution and integrity monitoring.***

At finer granularity, Liu et al. [6] recover instruction sequences from power traces using a control-flow-constrained HMM, while Park et al. [11] and Ugurlu et al. [12] infer instructions from profiled power or EM signatures. Vidal et al. [7] reconstruct basic-block execution from power traces using profiled templates, dynamic time warping, and control-flow constraints. Physical signals have also been used as out-of-band integrity monitors: Han et al. [13] identify PLC execution paths from EM emanations, Bai et al. [14] verify instruction execution using power measurements, EM-Flow [15] verifies compiled basic-block execution, and Haldar et al. [16] reconstruct task schedules from raw power traces. These approaches primarily infer or verify instructions, blocks, paths, or schedules using program structure or profiling. Moreover, physical execution evidence does not itself establish integrity or resistance to evasion, as demonstrated by Han et al. [17].

Table 1 summarizes the positioning of our approach against representative physical side-channel execution-monitoring methods. Rather than comparing performance values across non-equivalent experimental settings, the table highlights the dimensions that distinguish the approaches: physical signal, recognition granularity, signal representation, and inference assumptions. Our work specifically targets source-level C-function recognition from raw power windows in continuous, independently acquired workloads, complementing prior approaches based on engineered representations, aligned captures, or program-structure constraints.

**Table 1** Positioning against representative physical execution-monitoring approaches.

| Work | Signal | Target | Representation | Inference context |
|---|---|---|---|---|
| Chen et al. [5] | EM | Functions | Frequency-domain | Marker-aligned |
| Robyns et al. [8] | EM | Crypto operations | Filtered/STFT | Profiled operations |
| Jensen et al. [9] | Power/EM | Block categories | Raw windows | Short segments |
| Liu et al. [6] | Power | Instructions | Preprocessed | CFG-constrained |
| Vidal et al. [7] | Power | Basic blocks | Profiled templates | CFG-constrained |
| He et al. [18] | Power | C functions | Power traces | Inserted function traces |
| **This work** | **Power** | **C functions** | **Raw windows** | **Continuous unseen workloads** |

***Positioning.***

Our work addresses a complementary operating point: recognition of selected security-relevant C functions on bare-metal ARM Cortex-M systems directly from fixed-size raw power windows in continuous, independently acquired unseen workloads. Unlike approaches based on engineered spectral representations, marker-aligned function captures, or program-structure-constrained reconstruction, inference requires neither spectral feature engineering nor modification of the monitored workload. We compare sequence-classification baselines with class-specific reconstruction and explicitly characterize functions whose signatures remain insufficiently distinguishable.

The contribution is therefore not function recognition or physical execution monitoring in isolation, but their combination under these bare-metal constraints together with an empirical characterization of the resulting observability limits. The output is an external behavioral signal indicating which function appears to have executed; determining whether that invocation is unsafe or malicious requires additional contextual evidence.

# 3 Function-Recognition Methodology

## 3.1 Acquisition and Dataset Construction

We investigate function-level behavioral monitoring from raw power traces. Traces are acquired externally, segmented into fixed-size windows, and classified either as a target function or as out-of-distribution. Our primary method uses parallel class-specific autoencoders; Transformer and LSTM sequence classifiers serve as baselines.

Power traces are collected from an ARM Cortex-M4-K82F on a ChipWhisperer-308 motherboard using a ChipWhisperer Lite oscilloscope as shown in Figure 1. A software trigger immediately preceding the target function and a cycle counter provide execution boundaries. Measurements are sampled at the 8 MHz CPU frequency unless otherwise stated. Because the acquisition buffer is limited to 24,000 samples, longer executions are reconstructed from consecutive captures using corresponding offsets. Function inputs are randomized to capture parameter-dependent variation in execution length and power consumption. All binaries are compiled at `-O0`, providing a controlled baseline with explicit function boundaries rather than representing optimized production firmware.

The acquisition campaign produced 2,700 traces per function under identical measurement conditions, varying only the function input parameters. From these, 1,000

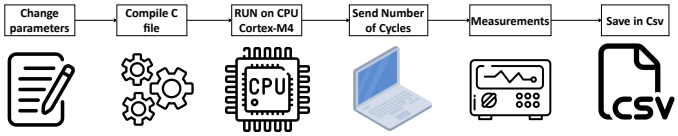



**Fig. 1** Power consumption measurement chain

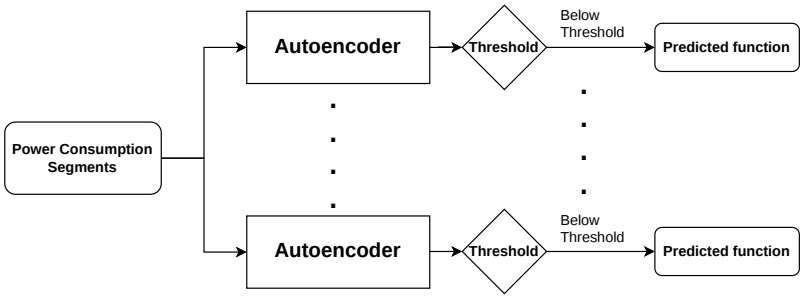



**Fig. 2** Parallel class-specific autoencoder recognition.

traces per function were randomly selected for dataset construction. Traces are segmented using sliding windows with 0%, 25%, 50%, and 75% overlap. The primary configuration uses 50-sample windows; 200-sample windows and sampling at $4 \times F_{\text{cpu}}$ are evaluated separately. Each overlap configuration contributes 5,000 windows per function, yielding 20,000 windows per function and a final balanced dataset of 140,000 windows.

The 70/10/20 training/validation/test partition is performed at the *raw-trace level before window extraction*; hence, all windows derived from one physical trace remain within the same partition. This prevents overlap-induced leakage across training, validation, and test data. Validation data are used for hyperparameter selection, early stopping, and threshold determination; the test partition is held out from model development.

## 3.2 Class-Specific Autoencoder Recognition

Our primary architecture addresses a limitation of closed-set classification: a model trained only on target functions may assign unrelated workload windows to one of the known classes. We instead train one autoencoder per target function, as shown in Figure 2. Each model therefore learns the reconstruction distribution of a single function's raw power signature.

Each autoencoder uses fully connected encoder layers of 128 units followed by a 64-dimensional bottleneck with ReLU activations; the decoder mirrors this structure and is optimized using mean-squared reconstruction error. For function $c$, the decision threshold $\tau_c$ is the 95th percentile of reconstruction errors on its validation data. At inference, each workload window is evaluated by all class-specific autoencoders. A function is accepted when its reconstruction error falls below the corresponding $\tau_c$; a window exceeding all thresholds is rejected as out-of-distribution. No workload

window in our evaluation satisfied the acceptance thresholds of multiple class-specific autoencoders simultaneously.

This class-specific decision mechanism allows unrelated workload segments to be rejected rather than forcing every window into a known function class.

### 3.3 Sequence-Classification Baselines

We compare class-specific reconstruction with two sequence-classification baselines. The *Transformer baseline* combines a shared autoencoder rejection stage, trained jointly on all target functions, with a four-layer Transformer encoder ($d_{\text{model}} = 128$, eight attention heads, feed-forward dimension $4d_{\text{model}}$, dropout $p = 0.1$) and attention pooling for function classification.

The *LSTM baseline* additionally tests whether explicit modeling of function transitions reduces boundary-induced errors. Complete function traces are concatenated in different orders, and 50-sample windows crossing function boundaries form a *Mixed* class. Training uses 3,000 function-order permutations sampled from 13,692 possible combinations. The classifier comprises three stacked LSTM layers with hidden size 64. We evaluate both the standalone LSTM and a variant preceded by the shared autoencoder rejection stage.

### 3.4 Training Protocol

Hyperparameters are optimized with Optuna using a Tree-structured Parzen Estimator over 15 trials, covering learning rate ($10^{-5}$–$10^{-3}$), batch size (8–32), bottleneck dimension (16–64), weight decay ($10^{-6}$–$10^{-3}$), optimizer (Adam, AdamW, RMSprop), and dropout (0–0.5). Trials run for at most 100 epochs with early stopping after 10 epochs without validation improvement; ReduceLROnPlateau halves the learning rate after five stagnant epochs. The lowest-validation-loss configuration is retrained from scratch before held-out evaluation. Python, NumPy, and PyTorch/CUDA are seeded at 42, with deterministic cuDNN execution enabled.

The held-out function-trace partition and the independent W1–W20 workloads serve distinct purposes: the former evaluates recognition on unseen physical function traces, whereas W1–W20 provide the primary inference-time evaluation on complete continuous application traces, as detailed in Section 4.

***Use of Generative AI.***

Generative AI tools were used during manuscript preparation to support language editing and restructuring of author-generated text. All scientific content, interpretations, analyses, and final wording were reviewed and approved by the authors, who take full responsibility for the manuscript.

## 4 Experimental Evaluation

### 4.1 Workloads and Evaluation Protocol

We evaluate the recognition pipelines at two independent levels. First, individually acquired function traces are partitioned at the raw-trace level into 70% training, 10%

validation, and 20% held-out test data *before* sliding-window extraction. Training data are used to fit the models; validation data are used for hyperparameter optimization, early stopping, and reconstruction-threshold selection; and the held-out test partition is used to evaluate recognition on unseen physical function traces. Consequently, overlapping windows derived from the same physical acquisition cannot occur in different partitions.

Second, we perform the primary workload-level evaluation on twenty independently acquired continuous workloads (W1–W20) executed on the ARM Cortex-M4-K82F. None of these workload traces is used for training, validation, hyperparameter optimization, early stopping, or threshold selection. During inference, each complete application trace is processed window by window over its full execution time. TP, TN, FP, and FN counts are accumulated across all windows of the complete W1–W20 application traces, from which the reported workload-level metrics are computed.

The workloads exercise the target functions in program structures associated with common embedded-software weaknesses: signed-to-unsigned conversion errors (CWE-195; W1, W3, W6, W7), buffer underread or overread (CWE-126; W2, W4, W9, W15), insufficient bounds checking and related buffer overflows (CWE-120; W5, W10–W14), buffer underwrite (CWE-124; W16), format-string misuse (CWE-134; W19, W20), and insecure handling of sensitive data (CWE-312; W8, W17, W18). These CWE associations provide a *security-relevant execution context*, not vulnerability ground truth: the evaluated task is function recognition, not determining whether a particular invocation is unsafe or exploited.

All workload-level recognition metrics are computed at the window level over the complete application traces. A true positive (TP) is a target-function window assigned to its correct class; a false negative (FN) is a target-function window not recognized as that class; a false positive (FP) is a non-target or differently labelled window incorrectly assigned to a target class; and a true negative (TN) is a non-target window correctly rejected. We report TPR, TNR, FPR, and FNR accordingly. These metrics quantify function recognition and do not indicate whether the underlying invocation was benign, unsafe, vulnerable, or exploited.

## 4.2 Recognition Performance

Table 2 summarizes the principal operating points of the three architectures on the independent W1–W20 workload evaluation set. With all seven target functions, false positives remain high across all approaches. The parallel autoencoder yields the lowest observed FPR (52.49%), compared with 68.44% for the Transformer and 67.52% for the LSTM, indicating that model choice alone does not make all seven function signatures reliably separable.

Two functions account for a substantial part of this difficulty. On the evaluated Cortex-M4-K82F implementation, `malloc` exhibits a near-invariant execution profile of approximately 166 cycles across the examined allocation sizes as shown in Figure 3, providing little parameter-dependent signal variation. Despite different input sizes, `strcpy` and `memcpy` exhibit similar repetitive power patterns, as illustrated in Figure 4. This inter-function similarity makes the local raw-power windows difficult

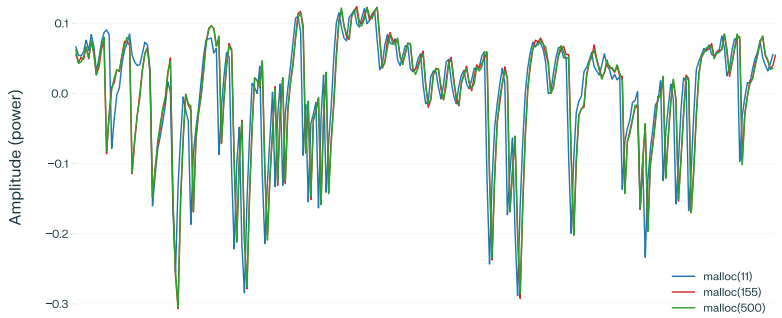



**Fig. 3** Representative malloc traces for allocation sizes 11, 155, and 500, illustrating the near-invariant power pattern.

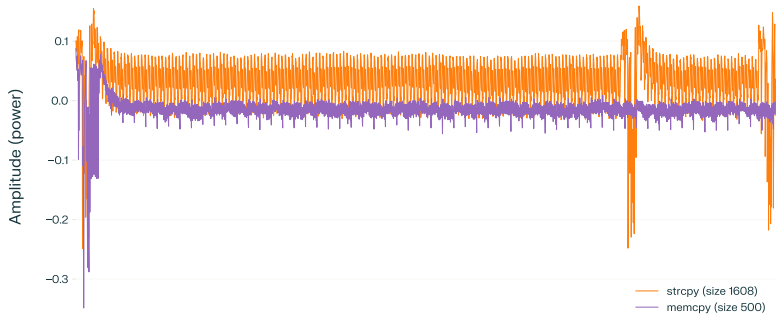



**Fig. 4** Representative `strcpy` and `memcpy` traces with different input sizes, illustrating similar repetitive power patterns despite the difference in executed byte counts.

to discriminate and is consistent with the persistent classification errors observed for `strcpy`.

For the retained five-function set (`memcpy`, `memset`, `memmove`, `strncpy`, and `strncat`), all three architectures improve. The parallel class-specific autoencoder provides the best observed operating point, reaching 96.00% TPR and 4.00% FNR at 39.79% FPR over the complete W1–W20 application traces. The Transformer reaches 90.00% TPR at 44.95% FPR, while the best LSTM configuration reaches 100% TPR at 59.38% FPR. Thus, class-specific reconstruction provides the best observed sensitivity–false-alarm trade-off in the evaluated configuration, although the remaining 39.79% FPR is still too high for standalone alerting.

## 4.3 Acquisition Parameters and Signal Limits

The primary configuration uses 50-sample windows with 75% overlap at $F_{\text{sampling}} = F_{\text{cpu}}$. Table 3 reports the sensitivity of the parallel-autoencoder pipeline to sampling rate and window size. Increasing the sampling rate to $4 \times F_{\text{cpu}}$ does not improve recognition in this setup. Even when the window is increased proportionally from 50 to

**Table 2** Window-level recognition performance over the complete W1–W20 workload traces on the Cortex-M4-K82F.

| Pipeline | Functions | TPR (%) | TNR (%) | FPR (%) | FNR (%) |
|---|---|---|---|---|---|
| Transformer | 7 | 92.86 | 31.56 | 68.44 | 7.14 |
| LSTM | 7 | 79.38 | 19.98 | 67.52 | 20.62 |
| Parallel AE | 7 | 80.63 | 47.51 | 52.49 | 19.37 |
| Transformer | 5 | 90.00 | 55.05 | 44.95 | 10.00 |
| LSTM + AE filter | 5 | 100.00 | 23.96 | 59.38 | 0.00 |
| Parallel AE | 5 | **96.00** | **60.00** | **39.79** | **4.00** |

**Table 3** Acquisition-parameter sensitivity of the parallel-autoencoder pipeline on W1–W20.

| $F_{\text{sampling}}/F_{\text{cpu}}$ | Window | Functions | TPR (%) | FPR (%) |
|---|---|---|---|---|
| 1 | 50 | 7 | 80.63 | 52.49 |
| 1 | 50 | 5 | 96.00 | 39.79 |
| 4 | 50 | 7 | 81.04 | 60.33 |
| 4 | 50 | 5 | 100.00 | 57.96 |
| 4 | 200 | 7 | 71.32 | 46.99 |
| 4 | 200 | 5 | 84.72 | 45.48 |

200 samples, the five-function configuration reaches only 84.72% TPR at 45.48% FPR. We therefore interpret the observed degradation as specific to the present acquisition setup rather than as a general property of oversampling.

The consistent difficulty of `malloc` and `strcpy` reveals two complementary limits of physical observability. For `malloc`, low intra-function variation leaves little signal structure for distinguishing parameter-dependent executions. For `strcpy`, similarity to other memory-transfer operations creates inter-function ambiguity. Their difficulty across Transformer, LSTM, and autoencoder architectures therefore suggests that additional model capacity alone cannot compensate for insufficient or overlapping physical information.

These observations are bounded to the evaluated Cortex-M4-K82F implementation, library implementation, `-O0` compilation, and ChipWhisperer acquisition configuration. They do not imply that either function is intrinsically unrecognizable on other processors, compiler configurations, allocator implementations, or measurement setups.

## 4.4 Contextual Evaluation on a Cortex-M4-F303

To assess whether the recognition procedure can be retrained beyond the primary board and to provide context against CNDSW by He et al. [18], we repeat the parallel-autoencoder evaluation on an ARM Cortex-M4-F303 using the five functions shared with their study: `strncpy`, `strncat`, `strcpy`, `memcpy`, and `memmove`. Under this configuration, the pipeline obtains 100% TPR, 0% FNR, and 33.35% FPR.

This experiment is a contextual cross-platform evaluation rather than a head-to-head performance comparison. CNDSW reports accuracy and tracks experimenter-inserted function traces, whereas our pipeline performs recognition in unmodified

**Table 4** Per-function results on the Cortex-M4-F303 contextual evaluation.

| Function | TP | FP | Recall (%) | Precision (%) |
|---|---|---|---|---|
| `memmove` | 1 | 15 | 100 | 6.25 |
| `strncpy` | 2 | 2 | 100 | 50.00 |
| `memcpy` | 1 | 14 | 100 | 6.67 |
| `strcpy` | 2 | 7 | 100 | 22.22 |
| `strncat` | 5 | 8 | 100 | 38.46 |

workload traces and reports sensitivity and false-alarm metrics separately. The reported numbers are therefore not directly comparable. The experiment instead provides evidence that the proposed recognition procedure can be retrained on a second Cortex-M4 implementation, while also confirming that substantial false-positive behavior remains. It does not establish direct cross-platform transfer.

Aggregate sensitivity also hides strong function-dependent reliability. Table 4 shows that all true occurrences are recognized in this experiment, but precision ranges from only 6.25% for `memmove` to 50.00% for `strncpy`. Given the small number of positive occurrences per function in this contextual experiment, these precision/recall values are descriptive rather than stable estimates of per-function performance. Based on these data, high sensitivity can coexist with a large number of false recognition events, reinforcing the conclusion that physical observability is function-dependent.

Overall, the evaluation establishes both feasibility and a clear operating boundary. Raw power contains sufficient information to recognize selected function executions across complete continuous workloads under the evaluated conditions, and class-specific reconstruction provides the best observed sensitivity–false-alarm trade-off among the evaluated architectures. At the same time, the substantial FPR and strong per-function variation show that raw-power function recognition should be interpreted as an external behavioral signal rather than a standalone security detector.

# 5 Discussion

## 5.1 Physical Observability and Security Implications

Our results reinforce that physical side channels expose software execution behavior, but also show that observability is function-dependent. Prior work has demonstrated instruction- and basic-block tracking, activity and function recognition, and operation localization from power or EM leakage [5–12]. We extend this evidence to selected security-relevant C functions recognized directly from raw power windows in continuous bare-metal workloads.

The persistent difficulty of `malloc` and `strcpy` across the three recognition architectures exposes two complementary limits: insufficient intra-function variation and excessive inter-function similarity. Thus, recognition is constrained not only by model capacity but also by the information present in the physical signal. This interpretation is consistent with prior evidence that side-channel recognition depends strongly on representation and acquisition conditions [10]; our findings therefore characterize

the evaluated hardware, compiler, library, and measurement configuration rather than intrinsic recognizability of these functions.

The resulting signal should be interpreted as *behavioral evidence of execution*, not evidence that an invocation is vulnerable or malicious. Physical signals have similarly been used for out-of-band execution and control-flow monitoring [13–15], while Han et al. [17] show that such monitors can themselves be evaded. Raw-power function recognition is therefore best viewed as an independent behavioral sensor that can complement software-derived evidence. Its independence from operating-system support or software instrumentation is attractive for bare-metal systems, but the observed false-positive rates and function-dependent precision preclude standalone alerting.

### 5.2 Generalization and Deployment Constraints

The evaluation provides a controlled feasibility baseline rather than a production deployment. Power signatures depend on hardware and acquisition conditions; evaluation on Cortex-M4-K82F and Cortex-M4-F303 shows that the method can be retrained on a second implementation, but not that signatures transfer without recalibration. Moreover, the primary experiments use `-O0`; compiler optimizations may alter or remove function signatures, consistent with related observations for software transformations in EM side channels [19]. Environmental variation, measurement placement, interrupts, concurrency, and realistic firmware complexity may likewise shift the observed signal distribution [10].

Trustworthy deployment therefore requires reducing false positives and establishing robustness across compilation settings, operational conditions, and hardware platforms. Until then, our results characterize the feasibility and physical limits of raw-power behavioral monitoring rather than a production-ready security detector.

## 6 Conclusion and Future Work

This work investigated whether raw power side channels can provide an independent behavioral signal for monitoring security-relevant function execution in bare-metal embedded systems. Our results show that they can, but also establish important limits on what such monitoring can reliably reveal.

For the retained target functions (`memcpy`, `memset`, `memmove`, `strncpy`, and `strncat`), the class-specific autoencoder architecture provides the best observed operating point among the evaluated approaches, achieving 96% TPR and 4% FNR on continuous, unseen workloads. Its 39.79% FPR remains substantial, however, and prevents interpreting the method as a standalone security detector. The experiments further show that recognition depends on acquisition and model design: short raw-power windows contain discriminative information, while increased sampling did not improve separability in our configuration, and class-specific reconstruction thresholds reduced false positives relative to the evaluated shared classification approaches.

More fundamentally, our results show that function recognition is bounded by the information exposed through the physical signal. The near-invariant execution profile of `malloc` and the overlapping memory-access patterns of `strcpy` remained difficult across all evaluated architectures. These failures are therefore not merely classification

errors to be addressed by a more powerful model; under the evaluated hardware, compiler, and acquisition conditions, they are consistent with limits in the discriminative information available in the observable signal. Raw-power monitoring is consequently function- and platform-dependent, and its capabilities must be characterized rather than assumed.

Recognizing a function execution does not establish that its invocation is vulnerable or malicious. Instead, raw-power recognition can provide an external execution signal that is independent of software instrumentation and can complement other evidence in a multi-stage security-monitoring architecture. Future work should therefore focus on improving precision and combining physical observations with contextual evidence, while evaluating robustness across compiler optimizations, environmental conditions, concurrent firmware execution, and additional hardware platforms.

Ultimately, the value of a physical side channel lies not only in what it reveals, but also in knowing where its information ends. Our results identify both sides of that boundary: raw power can expose security-relevant software behavior without relying on the monitored software stack, but trustworthy monitoring requires explicitly accounting for the functions and operating conditions under which that physical evidence remains distinguishable.

# References


[1] Sharma, C., Moylan, S., Vasserman, E.Y., Amariucai, G.T.: Review of the security of backward-compatible automotive inter-ecu communication. IEEE Access **9**, 114854–114869 (2021) https://doi.org/10.1109/ACCESS.2021.3104854

[2] Makrakis, G.M., Kolias, C., Kambourakis, G., Rieger, C., Benjamin, J.: Industrial and critical infrastructure security: Technical analysis of real-life security incidents. IEEE Access **9**, 165295–165325 (2021) https://doi.org/10.1109/ACCESS.2021.3133348

[3] Pereira, J.D., Lourenço, N., Vieira, M.: On the use of deep graph cnn to detect vulnerable c functions. In: Proceedings of the 11th Latin-American Symposium on Dependable Computing. LADC '22, pp. 45–50. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3569902.3569913

[4] Wilander, J., Nikiforakis, N., Younan, Y., Kamkar, M., Joosen, W.: Ripe: runtime intrusion prevention evaluator. In: Proceedings of the 27th Annual Computer Security Applications Conference. ACSAC '11, pp. 41–50. Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2076732.2076739

[5] Chen, L., Wu, J., Zhao, S., Jiang, P., Liu, J.: Mcu function recognition method based on the frequency-domain constellation trajectory figure of chip electromagnetic leakage and convolutional neural networks. IEEE Transactions on Instrumentation and Measurement (2025) https://doi.org/10.1109/TIM.2025.

3585214

[6] Liu, Y., Wei, L., Zhou, Z., Zhang, K., Xu, W., Xu, Q.: On code execution tracking via power side-channel. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (2016) https://doi.org/10.1145/2976749.2978299

[7] Vidal, B., Moreno, C., Fischmeister, S., Carvajal, G.: Monitoring software execution flow through power consumption and dynamic time warping. IEEE Embedded Systems Letters (2023) https://doi.org/10.1109/LES.2022.3197092

[8] Robyns, P., Martino, M.D., Giese, D., Lamotte, W., Quax, P., Noubir, G.: Practical operation extraction from electromagnetic leakage for side-channel analysis and reverse engineering. Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (2020) https://doi.org/10.1145/3395351.3399362

[9] Jensen, L., Brown, G., Wang, X., Harer, J.A., Chin, S.: Deep learning for minimal-context block tracking through side-channel analysis. ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2019) https://doi.org/10.1109/ICASSP.2019.8683845

[10] Le, Q., Miralles-Pechuán, L., Sayakkara, A., Le-Khac, N.-A., Scanlon, M.: Identifying internet of things software activities using deep learning-based electromagnetic side-channel analysis. Digit. Investig. (2021) https://doi.org/10.1016/j.fsidi.2021.301308

[11] Park, J., Xu, X., Jin, Y., Forte, D., Tehranipoor, M.: Power-based side-channel instruction-level disassembler. 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC) (2018) https://doi.org/10.1145/3195970.3196094

[12] Ugurlu, E., Yilmaz, B., Zajić, A., Prvulović, M.: Pitem: Permutations-based instruction tracking via electromagnetic side-channel signal analysis. IEEE Transactions on Computers (2022) https://doi.org/10.1109/TC.2021.3076354

[13] Han, Y., Etigowni, S., Liu, H., Zonouz, S., Petropulu, A.: Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (2017) https://doi.org/10.1145/3133956.3134081

[14] Bai, Y., Park, J., Tehranipoor, M., Forte, D.: Real-time instruction-level verification of remote iot/cps devices via side channels. Discover Internet of Things (2022) https://doi.org/10.1007/s43926-022-00021-2

[15] Stillman, C., Sheldon, J., Garrett, I.Y., Traynor, P., Gerdes, R., Rampazzi, S., Butler, K.R.B.: Em-flow: Advanced electromagnetic control flow verification for embedded systems. 2025 IEEE Annual Computer Security Applications

Conference (ACSAC) (2025) https://doi.org/10.1109/ACSAC67867.2025.00067

[16] Haldar, M., Patel, J., Fischmeister, S.: Power-based security monitoring: Non-invasive detection of task execution violations in real-time systems. 2026 29th International Symposium on Real-Time Distributed Computing (ISORC) (2026) https://doi.org/10.1109/ISORC70347.2026.11605786

[17] Han, Y., Chan, M., Aref, Z., Tippenhauer, N.O., Zonouz, S.: Hiding in plain sight? on the efficacy of power side Channel-Based control flow monitoring. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 661–678. USENIX Association, Boston, MA (2022). https://www.usenix.org/conference/usenixsecurity22/presentation/han

[18] He, D., Wang, H., Deng, T., Liu, J., Wang, J.: Improving iiot security: Unveiling threats through advanced side-channel analysis. Comput. Secur. **148**(C) (2025) https://doi.org/10.1016/j.cose.2024.104135

[19] Pham, D.-P., Marion, D., Mastio, M., Heuser, A.: Obfuscation revealed: Leveraging electromagnetic signals for obfuscated malware classification. In: Proceedings of the 37th Annual Computer Security Applications Conference. ACSAC '21, pp. 706–719. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3485832.3485894