On DoS Attacks Exploiting Input Representativeness in Mixed-Criticality Systems

Nicolas Benatti
Politecnico di Milano, Milan, Italy

Federico Reghenzani
Politecnico di Milano, Milan, Italy

Vittorio Zaccaria
Politecnico di Milano, Milan, Italy

Abstract

In order to satisfy power, area and cost constraints, modern Cyber-Physical Systems (CPSs) often consolidate tasks with vastly different assurance requirements onto a shared platform. Mixed-Criticality Systems (MCSs) enable this consolidation while maintaining timing guarantees on complex hardware through probabilistic timing analysis. Among these techniques, Measurement-Based Probabilistic Timing Analysis (MBPTA) has gained popularity in recent years; it works by deriving Worst-Case Execution Time (WCET) estimates from input samples collected at design time. The impossibility of fully covering tasks' input space during MBPTA can pave the way for Denial-of-Service (DoS) attacks: adversaries can, for example, craft data-oriented and sensor spoofing attacks that force execution along unobserved paths, inducing WCET overruns and deadline violations. This work demonstrates that temporal DoS attacks exploiting insufficient input representativeness pose a credible threat to MCSs, showing substantial impact on availability and criticality mode transitions through comprehensive simulations. Next, rather than pursuing unattainable perfect input coverage at design time, we investigate runtime detection via monitoring of execution-time distributions and propose a novel Goodness-of-Fit-based detection mechanism. Our detection approach exhibits improved accuracy and reduced variance compared to existing iid-based methods, offering more stable performance across heterogeneous workloads. The trade-off is higher detection latency, necessitating careful analysis of system-specific tolerance windows before deployment. Evaluation demonstrates both the practical threat posed by timing-based attacks and the viability of distribution-based anomaly detection, while acknowledging that synthetic evaluation cannot alone validate real-world applicability.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture; Computer systems organization → Embedded and cyber-physical systems; Security and privacy → Intrusion detection systems; Security and privacy → Denial-of-service attacks

Keywords and phrases Cyber-physical systems, mixed-criticality systems, pWCET, denial-of-service, spoofing, data-oriented attack

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2026.11

Funding This work has received partial funding from the European Chips Joint Undertaking under Framework Partnership Agreement No 101139789 (HAL4SDV).

1 Introduction

Cyber-Physical Systems (CPSs) underpin critical infrastructure in the automotive, aerospace, and medical sectors, with their prevalence growing alongside the proliferation of networked embedded devices. These systems must simultaneously deliver functionality and ensure safety, which necessitates consolidating tasks with vastly different assurance requirements onto shared platforms. A modern vehicle, for instance, hosts non-critical functions such as infotainment alongside safety-critical mechanisms (automatic emergency braking, anti-lock

braking, traction control) that must meet strict timing constraints with extremely low probabilities of deadline violation.

Ensuring temporal correctness in Mixed-Criticality Systems (MCSs) requires accurate estimation of each task's Worst-Case Execution Time (WCET). Probabilistic approaches, notably Measurement-Based Probabilistic Timing Analysis (MBPTA) [16], have gained traction because they yield less pessimistic bounds than traditional static or measurement-based techniques, enabling higher resource utilization. However, MBPTA derives its bounds from execution-time measurements over a subset of possible inputs, and achieving full representativeness of all possible execution scenarios at design time remains challenging. This gap creates a potential attack surface: Data-Oriented Attacks (DOAs) [21] and Sensor Spoofing Attacks (SSAs) [63] can manipulate inputs to force execution along paths not represented in the original measurements, potentially causing WCET overruns and, consequently, deadline violations. We therefore pose our first research question:

> $\mathcal{RQ}_1$: *To what extent can data-oriented and sensor spoofing attacks induce temporal anomalies by exploiting insufficient input representativeness in probabilistically analysed mixed-criticality systems?*

If such attacks pose a credible threat, a natural follow-up concerns mitigation. Rather than striving for arguably unattainable perfect input coverage at design time, we investigate whether anomalies can be detected at runtime by monitoring execution-time distributions, an approach explored in recent work [64]. This leads to our second research question:

> $\mathcal{RQ}_2$: *Given the heterogeneous temporal behaviour of tasks in mixed-criticality systems, can anomaly detection based on execution-time distributions be effective?*

To address these questions, we proceed as follows. Section 2 establishes the theoretical and practical context: we explain MBPTA, review how temporal anomalies arise in mixed-criticality systems, and describe the attack vectors we consider. Section 3 situates our work relative to existing microarchitectural and kernel-level timing attacks, distinguishing our application-level approach and surveying existing anomaly detection strategies. Section 4 develops our threat model and attack characterization methodology, defines metrics for quantifying availability impact, and presents our runtime detection mechanism based on Goodness-of-Fit testing. Section 5 reports experimental results on both attack impact and detection performance, providing concrete evidence for both research questions while explicitly discussing the limitations of synthetic evaluation. Finally, Section 6 synthesizes findings, acknowledges constraints on generalization, and outlines critical validation steps needed before real-world deployment.

2 Background

To better understand the practical consequences of temporal anomalies on mixed-criticality systems and how detection approaches work, this section provides a comprehensive introduction to the theoretical concepts behind mixed-criticality computing and explains how timing analysis is performed. We start by illustrating the different probabilistic timing analysis approaches in the literature, with a particular focus on MBPTA. Finally, we describe how temporal anomalies can be maliciously induced through data-oriented and sensor spoofing attacks.

2.1 Probabilistic WCET

Measurement-Based Probabilistic Timing Analysis (MBPTA) techniques were designed to overcome the WCET problem, i.e., the difficulties in estimating the WCET with static methods. The process of obtaining a probabilistic Worst-Case Execution Time (pWCET) through MBPTA begins with collecting representative execution time measurements, and each recorded observation becomes part of an execution time trace $\mathcal{T}$.

Before any statistical analysis can proceed, the trace must satisfy certain mathematical prerequisites that enable the application of Extreme Value Theory (EVT), which is the statical theory behind pWCET estimation. Specifically, the Generalised EVT formulation [55] requires three properties to hold. The first is *stationarity*, meaning the statistical characteristics of the trace remain consistent throughout its duration; this is verified using the Kwiatowski-Philips-Schmidt-Shin (KPSS) test [36]. The second and third requirements concern independence: observations must be free from both short-range correlations (assessed via the Brock-Dechert-Scheinkman-LeBaron test) and long-range dependencies (evaluated through the Hurst Exponent).

Once these conditions are satisfied, EVT can be applied to model the probability of observing extreme execution times. The Block-Maxima (BM) approach divides the trace $\mathcal{T}$ into fixed-size blocks, extracts the maximum value from each, and fits these maxima to a Generalized Extreme Value (GEV) distribution $G_\xi(z)$ [47], which is the probability of *not* observing execution times greater than z (by the Fisher-Tippett-Gnedenko theorem). The resulting shape parameter ξ is particularly important, as it determines the GEV *type*:

- Gumbel family (or Type I) if $\xi = 0$, $\forall z \in \mathbb{R}$
- Fréchet family (or Type II) if $\xi > 0$, $\forall z \in \left[\mu - \frac{\sigma}{\xi},\ \infty\right)$
- Weibull family (or Type III) if $\xi < 0$, $\forall z \in \left(-\infty, \mu + \frac{\sigma}{|\xi|}\right]$

While all three GEV types (Gumbel, Fréchet, and Weibull) are valid results [40, 54], the Fréchet type ($\xi > 0$) can produce notably pessimistic estimates due to its heavy tail.

Finally, the WCET value is extracted as the $(1-p)$-quantile of the fitted distribution $G_\xi(z)$ at the desired exceedance probability p, formally expressed as $WCET = G_\xi^{-1}(1-p)$. The resulting tuple $(WCET, p)$ constitutes the pWCET which must be read as "we expect that any observation will be less that WCET with $(1-p)$ probability".

Challenges of MBPTA. One of the central challenges of EVT is ensuring the short- and long-range independence of timing samples. This might be ensured by appropriate hardware design, i.e., when hardware resources use random policies (random cache replacement, randomised bus arbitration, randomised initial cache state), enforced either at the hardware or software level, the execution time of each measurement becomes independent of previous runs. Such techniques are being tested in modern mixed-criticality SoCs (*e.g.*, LEON3/LEON4 SPARC processors used by ESA [34]) to facilitate the timing analysis process.

Another long-standing MBPTA challenge is input *representativeness*: ensuring that the experimental inputs used at design time to gather pWCET stimulate significant coverage of the program and increase the probability of observing program paths that yield unexpectedly high execution times. MBPTA fundamentally assumes that test inputs are "representative" operational inputs; however, this is often difficult to define and verify. This might become an important attack vector for sensor spoofing and data-oriented attacks, where external (or even impossible) conditions not considered at design-time are introduced to mutate program behaviour.

2.2 Software and Sensor Security

The Internet of Things (IoT) brought a massive increase in connectivity between physical devices throughout the recent years [4]. This has introduced novel threat scenarios for CPSs [61] that simply had not been thought of before. In this work we focus on two prominent attack vectors: *data oriented attacks* and sensory input manipulation (*sensor spoofing*).

Data-oriented attacks. DOAs are a broad class of techniques that exploit memory corruption vulnerabilities. They were originally introduced in 2005 by Chen et al. [19] as a variant of the more common *control-flow hijacking* attacks. Instead of corrupting function return addresses (*backward-edge*) or indirect calls/branches (*forward-edge*), these attacks aim to manipulate non-control data. Therefore, DOAs carry the powerful property of preserving control-flow, bypassing traditional mitigations such as Control-Flow Integrity (CFI) or ASLR; this makes such attacks much harder to detect.

The taxonomy introduced in 2021 by Cheng et al. [21] categorises DOAs based on how the attacker manipulates data in the memory space of the victim task:

- *Direct data manipulation (DDM)* — This is the most straightforward kind of data-oriented exploit. The attacker corrupts non-control data (*e.g.*, decision-making variables, loop bounds, etc.) on the stack as an immediate result of a memory corruption. Such attacks are simple to mount; however, the attacker must know exactly where the target data resides in memory. DDMs can be either single or multi-step, *e.g.*, if the adversary needs complex interaction with the victim program, such as in web servers.
- *Data oriented programming (DOP)* — This attack allows for more expressive (effectively Turing-complete) exploits by leveraging existing gadgets in the victim program's code. We do not consider them in this paper.

Although DOAs preserve control flow, they produce measurable side effects [21]: (i) *incompatible branch behaviour*, where corrupted decision-making data causes logically infeasible execution paths (*e.g.*, nested `if` branches that observe different values of a locally constant variable); and (ii) *branch frequency anomalies*, where branch execution rates deviate from normal patterns. For example, executing incompatible branches could result in an anomalous number of cache misses, with a risk of exceeding the WCET.

Sensor spoofing attacks. SSAs trick CPSs into dangerous actions by manipulating their perception of the surrounding environment. This broad class encompasses GPS, camera, and LiDAR spoofing in autonomous vehicles [63], manipulation of medical devices [50], and authentication bypass on wearables [62]. In this paper, we focus on SSAs targeting availability.

3 Related Work

Timing security in CPSs is a relatively new research area. To the best of our knowledge, this is the first study that addresses maliciously exploiting execution path level jitter to induce temporal anomalies in CPSs. In this section, we highlight relevant studies investigating other sources of jitter causing timing interference and provide an overview of temporal anomaly detection approaches, encompassing both traditional threshold-based and more recent techniques—which we will refer to as *iid-based*—that monitor the independence and identical distribution (iid) property of the collected execution times.

3.1 Kernel and Hardware Timing Interference Attacks

From an offensive standpoint, previous efforts have primarily focused on crafting DoS attacks that exploit various architectural and operating-systems' related sources of jitter which might compromise assumptions on temporal-isolation; since these are well systematized in [37] we report in this section the most relevant works.

Considering kernel-level attacks, Mahfouzi et al. [42] presented the *butterfly attack*, a scheduler-level attack against MCSs, which shows that delaying incoming data to a low-criticality task can increase the response time of dependent high-criticality tasks, bringing the system to physical instability. This attack is similar to ours in that it affects any MCS regardless of hardware or OS properties; however, the underlying attack vector differs, and their evaluation focuses on control-system stability rather than real-time impact.

Mergendahl et al. [45] presented the *thundering-herd attack*, which disrupts temporal isolation on mixed-criticality microkernels[1] by causing contention on shared OS services. The attack exploits priority-sorted IPC endpoint queues and budget replenishment queues [58]; by exhausting these resources through numerous low-priority tasks performing IPC calls, delays are introduced in higher priority tasks. Evaluation on seL4-MCS shows significant impact, though the required 10^2–10^3 attacker-controlled threads may limit applicability. Shared I/O or network queues have also been identified as another impactful interference channel. In particular, virtualised environments—key enablers of consolidation in mixed-criticality computing—have been shown to be vulnerable to a plethora of attacks that cause interference across co-resident VMs [33]. Additionally, scheduler side-channel attacks [18, 35] can provide an attacker with knowledge of the system's task schedule, enabling precisely timed interference, such as the sensor spoofing and data-oriented attacks we consider, at the exact activation of victim tasks.

At the microarchitectural level, attacks primarily target multicore architectures. Despite long-known representativeness issues [41], the first CPS-targeted attack appeared in 2019 when Bechtel and Yun [10] demonstrated that the *Miss-Status Holding Register* (MSHR) of shared caches, if maliciously clogged, can cause execution time increases of up to 300x, bypassing cache partitioning systems.

Most closely related to our work, Duncan et al. [25] introduced the *Mad Monk* attack, in which a compromised low-criticality task exploits shared microarchitectural resources (*e.g.*, caches) to delay a higher-criticality *mark* task beyond its execution budget, thereby forcing arbitrary criticality mode switches without violating any MCS scheduling principle. While Mad Monk and our approach share the high-level goal of demonstrating that low-criticality tasks can undermine the temporal guarantees of higher-criticality ones, the two works differ in both attack vector and scope. Mad Monk operates at the hardware level, relying on resource contention (*e.g.*, cache flushing) to inflate the execution time of a specific mark task; our work instead operates at the application level, showing that data-oriented and sensor spoofing attacks can exploit insufficient input representativeness in MBPTA to trigger WCET overruns along execution paths that were never exercised at design time. The Mad Monk attack can in principle be mitigated through resource partitioning, whereas the class of attacks we consider persists even on temporally isolated platforms, since the root cause lies in the gap between the inputs observed during timing analysis and those encountered at runtime by higher criticality tasks. Furthermore, our work complements the offensive

[1] In microkernels, OS services run as user-space servers communicating via IPC with client tasks, and time budgeting mechanisms enforce temporal isolation.

analysis with a runtime detection mechanism, an aspect not addressed in [25].

Esposito et al. [56] presented the *bank & row conflict bomb* attack on shared DDR4 DIMM DRAM modules, where an attacker delays victim task execution by exhausting the DRAM row buffer through massive memory accesses, thereby increasing memory latency and exceeding the estimated worst-case execution time.

Building on these primitives, Li et al. [39] demonstrated real-world impact by hijacking an autonomous vehicle's trajectory via cache contention against the *localisation* subsystem. In 2022, the same authors presented *Polyrhythm* [37], which combines multiple interference channels tuned via reinforcement learning, effectively inducing deadline misses in autonomous vehicles. Later work [38] showed that fail-safe mechanisms can limit multicore interference effects, requiring fine-tuned attack intensity and timeliness. We must note that these approaches address *logical* timing errors rather than WCET overruns; however, evaluations are specific to autonomous systems and lack mixed-criticality considerations and real-time impact quantification. Besides, contrary to our work, such approaches resort to generic stressor workloads rather than realistic attacks. We believe that evaluating the interference-generating capability of realistic attacks is needed for a better characterization of temporal threats to CPSs.

3.2 Temporal Anomaly Detection in Cyber-Physical Systems

Existing approaches to anomaly detection in CPS can be broadly distinguished as follows: (i) time-based defences against attacks that cause task execution delays, and (ii) detection of malware that leaves traces in the system's temporal profile. In the following, we include time-based fault detection, as faults represent temporal anomalies; however, approaches that detect *only* DOAs [22, 60] or *only* SSAs [49] are not considered, nor are program-behavior-based approaches that examine system call distributions [65] or model execution as regular languages [22].

In this context, our main competitor approach is *BoostIID* [64], which detects runtime pWCET distribution variations via ensemble learning; more specifically, their approach builds a detector by composing three well-known iid tests (*KPSS*, *RS*, *Ljung-Box*) via the *boosting*[2] technique [20]. Here, a history buffer feeds the tests online, signalling anomalies when the boosting output rejects the i.i.d. hypothesis. Evaluation uses synthetic scenarios with modified GEV parameters, measuring detection latency, safety margin, and F_1 score. However, performance drops ($\approx 18\%$ F_1 loss) as soon as the GEV scale parameter σ increases, suggesting poor robustness to statistical variations. Practical considerations regarding the implementation of this runtime detector are also missing.

On the other hand, more traditional techniques primarily deal with setting temporal thresholds that, if exceeded, trigger user-supplied recovery policies. Hamad et al. [31] devised a two-threshold detector with spare time for recovery; tasks are monitored at the first threshold and terminated at the second. The main drawback is a strong dependence on the representativeness of timing analysis. Esposito et al. [27] applied two-threshold detection to performance counters in MCSs, translating thresholds into confidence levels on program-counter derived data distributions. While counter heterogeneity enables holistic monitoring, applicability depends on the available performance counters.

[2] Boosting is a sequential ensemble method that combines many weak learners (typically shallow decision trees) by iteratively training each new model to focus on the errors made by the previous ones, producing a strong overall predictor.

Another cluster of solutions [66, 11, 44] measures the WCET of program basic blocks by inserting timing annotations at compile time and enforcing them at runtime. Zimmer et al. [66] presented three time-based CFI techniques: *T-Rex* for detecting anomalous timing along return paths (suited for ROP attacks), *T-ProT* for buffer overflow protection via checkpoints, and *T-AxT* , which ensures the program counter stays within safe ranges through scheduler-level checks without instrumentation. Bellec et al. [11] extended this with hardware-based monitoring, while McDonald and Mueller [44] further extended T-ProT to protect kernel code and user-kernel transitions.

These WCET-based approaches still depend heavily on the representativeness of timing analysis. Furthermore, except for T-AxT, all require instrumentation, which can alter memory layout and harm predictability [46].

4 Approach

This section presents our methodology in assessing and detecting temporal anomalies caused by DOAs and SSAs. After (i) stating our system and threat model, we (ii) list useful strategies that can be used to effectively increase task execution times via DOAs and SSAs, and (iii) define metrics to quantify the resulting loss in availability. Finally, we present our pWCET-monitoring temporal anomaly detection approach.

4.1 System and Threat Model

We consider the following environment in the context of a timing DoS attack:

- *Computing Model* — A computing platform consisting of:
  - A uniprocessor P running a mixed-criticality task set Γ according to the traditional two-criticality Vestal model [59].
  - n tasks $\Gamma = \{\tau_1, \dots, \tau_n\}$ with $\tau_i = (T_i, D_i, \chi_i, \boldsymbol{C_i})$, where T_i is the period, D_i the relative deadline, $\chi_i \in \{C_i^{LO}, C_i^{HI}\}$ the task criticality level, and $\boldsymbol{C_i} = \{C_i^{LO}, C_i^{HI}\}$ the LO- and HI-criticality WCET. In particular, C_i^{LO} is estimated via MBPTA and C_i^{HI} via static analysis.
  - A Real-Time Operating System (RTOS) offering IPC primitives (*e.g.*, shared buffers).
  - A dynamic-priority, preemptive mixed-criticality scheduler, under an implicit deadline system, *i.e.*, $D_i = T_i$. The MCS scheduler follows the usual mode switch concept: the system is in one criticality mode (LO-mode or HI-mode) at any given time. When the system is in LO-mode, all tasks are guaranteed to run. If, however, any HI-criticality job $\tau_{i,j}$ executes for more than C_i^{LO} time units, the system transitions from LO-mode to HI-mode, where only HI-criticality tasks are guaranteed to run while LO-criticality tasks are dropped.
- *Deployed Mitigations* — The system is assumed to be vulnerable to DOAs and SSAs. We do not specifically require mitigations against these attack vectors to be absent, since they are not perfect and some intrusions would go undetected anyway.

To better contextualize our approach, we define the following threat model:

- *Intrusion Model* — The attacker can only interfere with HI-criticality tasks (referred to as *victim* tasks) through LO-criticality tasks (referred to as *violated* tasks).
- *Trusted Code Base (TCB)* — The RTOS kernel and underlying firmware code are considered to be trusted.
- *Attacker Capabilities* — The attacker is able to exploit memory corruption vulnerabilities in LO-criticality tasks by forging arbitrary frames, packets or sensory inputs; such data

is then delivered to a HI-criticality task through some IPC mechanism. Possible attack vectors can be vulnerabilities in the Bluetooth stack, remote keyless-entry systems, etc. [17]. Moreover, the attacker has also access to the source code of tasks (*i.e.*, no *security through obscurity*).

Finally, we can state the attacker's goal, which is to trigger temporal anomalies in order to bring the system into degraded mode. Malicious behaviour of different nature (*e.g.*, stealing secrets via side-channel attacks, breaking authentication of inter-device communication protocols, etc.) is out of the scope of this work.

4.2 Characterising Timing DoS Attacks

Given the threat model defined above, we describe the general strategies through which DOAs and SSAs can induce temporal anomalies. For each attack class we introduce two parameters that will be used to quantify attack effort throughout the evaluation: *intensity*, which measures the magnitude of the perturbation applied to a single job, and *frequency*, which measures the fraction of jobs that carry a malicious payload.

Data-oriented attacks. Our DOA strategy targets timing-sensitive non-control data—such as loop bounds and decision variables—through buffer overflow vulnerabilities, aiming to inflate execution times along paths not exercised during timing analysis. As a concrete vulnerability model, we take inspiration from real-world Zephyr kernel vulnerabilities such as CVE-2023-3725, where bound checks are inadvertently placed inside assertions that are removed by the compiler in *release* builds; the resulting unchecked variables are passed to `memcpy`, enabling an out-of-bounds copy from the buffer received via IPC. Depending on the nature of the corrupted data, we expect two classes of side effects (see Section 2.2): *branch frequency anomalies* when loop bounds are inflated, and *incompatible branch behaviour* when decision variables are corrupted.

Sensor spoofing attacks. Our SSA strategy is twofold. The first component targets input *distributions*: motivated by [40, 14], we evaluate whether altering the statistical profile of sensor readings—relative to a baseline of uniform sampling—can shift the pWCET. We consider Poisson and Gaussian distributions; for the latter, we distinguish a *wide* variant, whose $2\sigma_{wide}$ interval covers the full input space (*i.e.*, only $\approx 5\%$ of values fall outside), and a *narrow* variant with $\sigma_{narrow} = \sigma_{wide}/5$. The second component targets input *content*: for each benchmark we craft *worst-case inputs* designed to trigger the algorithm's most expensive execution path. Intensity here measures the degree to which the input approximates the theoretical worst case.

4.3 Measuring Attack Impact

To thoroughly capture DOA and SSA impact on temporal correctness, we define four complementary metrics addressing both static design-time assumptions and dynamic runtime behaviour:

(1) WCET overrun (ΔC, static analysis). Quantifies by how much observed execution times exceed the design-time LO-criticality WCET estimate. Given a timing trace, defined as a finite sequence of execution time observations $\{\mathcal{X}_i\}_{i=1}^{n}$, we compute:

$$\Delta C = \frac{\max_{1 \le i \le n} \{\mathcal{X}_i\}}{C^{LO}} - 1 \tag{1}$$

where C^{LO} is the estimated pWCET. A value $\Delta C > 0$ indicates WCET pessimism was insufficient.

(2) **Schedulability** (S, static analysis). Determines whether timing assumptions remain satisfied under attack by evaluating the schedulability condition for the chosen preemptive scheduler. Losing schedulability places the system in an inconsistent state where even HI-criticality WCETs may be exceeded.

(3) **Deadline miss ratio** (R_{miss}, dynamic simulation). Measures the fraction of jobs that miss their deadline over a fixed observation period ($0 \leq R_{miss} \leq 1$). Directly quantifies the fraction of timing constraint violations.

(4) **Degraded-mode ratio** (K_{HI}, dynamic simulation). Captures the fraction of execution time spent in HI-criticality (degraded) mode ($0 \leq K_{HI} \leq 1$). Indicates how frequently the system must activate stricter resource management in response to detected deadline threats.

4.4 Detecting Timing DoS Attacks at Runtime

As discussed in Section 1, our second research question ($\mathcal{RQ}_2$) asks whether anomaly detection based on execution-time distributions can be effective given the heterogeneous temporal behaviour of tasks in MCSs. Existing approaches such as BoostIID [64] rely on testing whether execution times remain independent and identically distributed (iid) when the system is running; however, as noted in Section 3.2, their accuracy degrades (up to 18% F_1 loss) when the GEV scale parameter σ increases—a common situation in mixed-criticality workloads where tasks exhibit diverse timing profiles.

We propose an alternative strategy: instead of testing for iid properties to hold, we directly compare the pWCET distribution of a task G_ξ—estimated at design time via MBPTA (see Section 2.1)—with the empirical, discrete Block-Maxima distribution of execution times $E(z)$, computed by frequency-counting the observed samples $\mathcal{X}_i$ at runtime. From an MBPTA perspective, our solution is orthogonal to [64] in that it performs online testing of EVT *postconditions* instead of *preconditions*.

The key insight of this approach is that, as long as a statistical test confirms *Goodness-of-Fit* (GoF) between these distributions, *i.e.*, $E(z) \approx G_\xi(z)$, we can be reasonably confident that the task behaves as expected at design time.

Choosing a GoF test. The effectiveness of pWCET-based detection depends on selecting an appropriate statistical test. While we aim to keep the approach flexible, we also care about time and space complexity of the actual implementation; since detection must run continuously alongside the usual system workload and therefore should have bounded latency and memory occupation, which can be taken into account since design time. In order to keep execution time variability at a minimum, we also advise for the detection task to be run in isolation from the rest of the system, for example by leveraging Trusted Execution Environments (*e.g.*, ARM TrustZone).

Several GoF tests exist, including Anderson-Darling and Cramér-von Mises. We select the Kolmogorov-Smirnov (KS) test for three reasons: (i) it is non-parametric, requiring no assumptions about the reference distribution's mathematical form; (ii) efficient online implementations exist [15]; and (iii) it was historically used to verify MBPTA applicability before Generalized EVT was introduced [24], making it a natural fit for runtime monitoring of pWCET distributions.

Specifically, we use the one-sample KS variant, which measures GoF between an empirical, discrete CDF E and a reference analytical CDF G_ξ. We adopt GreedyKS [15], a state-of-the-

art online implementation in which the main idea is tracking $D^+ = \max_z[G_\xi(z) - E(z)]$ and $D^- = \max_z[E(z) - G_\xi(z)]$, *i.e.*, the maximum positive and negative deviations between the two CDFs required for computing the Kolmogorov statistic $D = \max\{D^+, D^-\}$. To compute CDFs efficiently, the authors advise to store samples from the empirical distribution E in a data structure optimised for computing *prefix sums*, that is, the sum of every element up to a certain value; we choose Fenwick trees [28] for this purpose. The algorithm then partitions the probability interval $[0, 1]$ into m sub-intervals (*bins*) and assigns newly received sample $\mathcal{X}_i$ to their respective one, based on the value of $G_\xi(\mathcal{X}_i)$. Finally, minimum and maximum statistics for each bin are kept in order to compute positive and negative differences, updating D^+ and D^- accordingly.

Overall, GreedyKS has $\mathcal{O}(m)$ space and $\mathcal{O}(\log m)$ time complexity per call, which make it a desirable choice. Since m is an engineering parameter chosen at design time, both time and space complexities can be considered effectively constant.

Anomaly detection algorithm. The pseudocode for our approach is reported in Algorithm 1. The detection algorithm runs every time a job of task τ terminates, collecting its execution time x; all the information is collected separately for each task. We model the detection logic using two stateful objects: a BlockMaxima array (BM) of size S, which performs online Block-Maxima filtering with block size B when a new sample is received, and a GreedyKS estimator (KS), which maintains the empirical distribution E and compares it against the reference pWCET G_ξ. The empirical distribution E is updated once every B task activations. We claim that this added latency can be compensated by choosing a reasonably small block size B.

When a new Block-Maxima sample becomes available, we manipulate the KS object to remove the oldest sample and add the newest one to maintain the empirical distribution E circularly. From an implementation standpoint, this means adding and removing an element from an (internal to KS) Fenwick tree and updating the statistics for every bin. While the original GreedyKS implementation only supports KS.Add; we extend it with KS.Remove to enable this circular buffer management (Appendix B reports the modified algorithm's pseudocode). In particular, to keep bin statistics coherent upon removal, we employ per-bin min- and max-heaps. Both primitives have $\mathcal{O}(\log m)$ time complexity and $\mathcal{O}(m + S)$ space complexity. Finally, KS.Test computes the p-value p from the test statistic $D_{KS} = \max\{D^+, D^-\}$. If $p < \alpha$, the test rejects the null hypothesis and signals an anomaly.

Algorithm 1 pWCET-monitoring anomaly detection

Require: α: Confidence level for KS test
Require: BM: BlockMaxima object (initialized with block size B)
Require: KS: GreedyKS object (initialized with m bins, reference CDF G_ξ, maintaining empirical distribution E)

```
1: while system is running do
2:    x ← AcquireNewSample()
3:    BM.Update(x)
4:    if BM.HasNew() then
5:       KS.Remove(BM.Oldest())
6:       KS.Add(BM.Latest())
7:       if KS.Test() < α then
8:          ⟨anomaly detected⟩
9:       end if
10:   end if
11: end while
```

Table 1 Attack space exploration strategy.

Benchmark	DOA Strategy	SSA Strategy
Bubble sort (`bsort`)	Corrupt loop bounds	Reverse-sort input
Insertion sort (`insort`)	Corrupt loop bounds	Reverse-sort input
Quicksort (`qsort`)	Corrupt loop bounds	Reverse-sort input
Primality testing (`prime`)	–	Increase input magnitude
Complex nested loops (`janne_complex`)	–	Increase inner loop runs
Integral computation (`expint`)	–	Trigger rarely-executed inner loop
Car window lift control (`statemate`)	Corrupt FSM state variables	–

5 Evaluation

Following the order of reasoning laid out in Section 1, we split the evaluation procedure in two parts: Section 5.1 demonstrates the feasibility of DOAs and SSAs on chosen benchmark programs and measures their impact on temporal correctness, while Section 5.2 measures pWCET-based detection performance on synthetic mixed-criticality workloads.

5.1 Attack Impact Measurement

In this section, we (i) detail the hardware and software platform on which we run the attack scenarios (Section 5.1.1), (ii) explain which attack strategies can be used in such scenarios (Section 5.1.2), and (iii) report how each phase in the impact measurement pipeline is configured (Section 5.1.3).

5.1.1 Experimental Setup

The target platform for our experiments is a RISC-V emulated SoC running the Zephyr real-time operating system; specifically, we design an out-of-order, single-core 32-bit microcontroller with speculative execution enabled and 8KB set-associative cache (32B cache lines), on the TBM architecture simulator from Google [29] and emulate its execution on Renode [53], selecting the `riscv32_virtual` target. Such configuration is justified by the increasing interest of many industrial and automotive vendors in RISC-V and Zephyr [2, 57]. To show that execution-path level jitter is independent from the predictability of the hardware platform, all benchmarks are executed on a single core, without any kind of interference from concurrent tasks.

Benchmark programs. We use benchmark programs from the Mälardalen WCET benchmark suite [30] as a reference for impact analysis. We pick seven, categorising them based on (i) presence of complex loop or branch logic (`bsort`, `insort`, `qsort`, `janne`, `statemate`), (ii) presence of integer arithmetic (`prime`) or (iii) rarely taken execution paths (`statemate`, `expint`). The input space for each benchmark has been substantially enlarged from the original suite, with the intention to come closer to realistic CPS tasks—such as those employed in [26]. We port each modified benchmark into a Zephyr application consisting of two tasks: a *sensor* task, which simulates sensor readings, and a *control* task running the benchmark logic. The two tasks communicate through shared buffers.

Table 2 Input distribution parameters employed during the first phase of sensor spoofing attacks.

Benchmark	Input Space Size	Uniform	Normal "wide"	Normal "narrow"	Poisson
`bsort`	$32 \cdot 10^4 \approx 10^{5.5}$	$\mathcal{U}[5 \cdot 10^3, 15 \cdot 10^3]$	$\mathcal{N}\left(10^4, 2500\right)$	$\mathcal{N}\left(10^4, 500\right)$	$\text{Pois}\left(10^4\right)$
`insort`	$32 \cdot 10^4 \approx 10^{5.5}$	$\mathcal{U}[5 \cdot 10^3, 15 \cdot 10^3]$	$\mathcal{N}\left(10^4, 2500\right)$	$\mathcal{N}\left(10^4, 500\right)$	$\text{Pois}\left(10^4\right)$
`qsort`	$32 \cdot 10^4 \approx 10^{5.5}$	$\mathcal{U}[5 \cdot 10^3, 15 \cdot 10^3]$	$\mathcal{N}\left(10^4, 2500\right)$	$\mathcal{N}\left(10^4, 500\right)$	$\text{Pois}\left(10^4\right)$
`prime`	$32 \cdot 10^4 \approx 10^{5.5}$	$\mathcal{U}[5 \cdot 10^3, 15 \cdot 10^3]$	$\mathcal{N}\left(10^4, 2500\right)$	$\mathcal{N}\left(10^4, 500\right)$	$\text{Pois}\left(10^4\right)$
`janne_complex`	$5 \cdot 10^2 \approx 10^{2.5}$	$\mathcal{U}[0, 500]$	$\mathcal{N}\left(250, 125\right)$	$\mathcal{N}\left(250, 25\right)$	$\text{Pois}\left(300\right)$
`expint`	10^2	$\mathcal{U}(0, 100]$	$\mathcal{N}\left(50, 25\right)$	$\mathcal{N}\left(50, 5\right)$	$\text{Pois}\left(60\right)$

5.1.2 Attack Space Definition

Following the methodology of Section 4.2, we instantiate DOA and SSA strategies for each benchmark as summarised in Table 1, varying frequency and intensity between 25% and 100% at 25% steps for a total of 97 attack scenarios, categorised as follows.

DOA instantiation. For sorting benchmarks, intensity is defined as the increase in the corrupted loop bound; for `statemate`, intensity corresponds to the number of corrupted FSM state variables. Frequency is the fraction of jobs receiving a malicious IPC payload. Inputs are randomly drawn from a uniform distribution, as reported in Table 2.

SSA instantiation — distribution alteration (SSA-DA). Here, inputs are randomly drawn from distributions other than uniform. Specifically, four input distributions are considered per benchmark; their parameters are reported in Table 2. Note that the expectation λ for Poisson distributions has been chosen to be roughly 60% of the input range, except for sorting algorithms and `prime`, where there input space has more than one dimension, and this logic ceases to make sense.

SSA instantiation — worst-case inputs (SSA-WCI). Here, inputs are drawn uniformly at random. Malicious payloads are crafted per-benchmark:

- *Sorting* (`bsort`, `insort`, `qsort`) — send a portion of reversely-sorted data (*i.e.*, opposite to the sorting order); intensity is the fraction of reverse-sorted elements.
- *Primality* (`prime`) — arrays of large-magnitude primes sent with increasing frequency.
- *Nested loops* (`janne_complex`) — inputs that maximise inner-loop iterations; since a clean intensity metric is hard to define, we additionally consider a missing-bound-check vulnerability motivated by real-world exploits [50].
- *Integral computation* (`expint`) — inputs triggering the rarely-executed inner loop; intensity is the inner-loop bound increase.

Finally, we also measure a *ground truth* scenario for each benchmark, in which the system runs normally without being affected by any kind of malicious behaviour. Such measurements will then be used as a reference when computing the four metrics defined in Section 4.3.

5.1.3 Experimental Methodology

Our experimental pipeline consists of three phases: timing analysis feeds the WCET overrun (ΔC) and provides the inputs for schedulability analysis (S), while scheduling simulation yields the deadline miss ratio (R_{miss}) and the degraded-mode ratio (K_{HI}).

Timing analysis. LO-criticality WCET estimation is carried out on every ground truth scenario by employing the state-of-the-art Chronovise MBPTA tool [51], specifying (i) an exceedance probability of $p = 10^{-6}$, (ii) KPSS, BDS and RS tests—at $\alpha = 0.05$ confidence level—to check for EVT applicability, and collecting (iii) $N = 10^4$ execution time samples for each scenario.

As reported in Section 5.1.2, EVT applicability is ensured via multiple sources of variability; OS level interference caused by the Zephyr kernel, randomised input choices (see Section 5.1.2) and the presence of caches in our hardware model. Specifically, our choice for a random measurement protocol is motivated by [3], which shows that the randomised algorithms commonly employed in autonomous driving workloads allow to perform MBPTA on non-randomised platforms. We note that, despite not employing randomised caches, those have been shown to be neither necessary nor sufficient for EVT-based timing analysis [55, 54], and prior work has successfully applied EVT on non-randomised platforms [52, 43, 12]. Estimation results are reported in Appendix C.

Finally, we use the well-established Maximum Likelihood Estimator (MLE) to produce the pWCET distribution $G_\xi^{GEV}(z)$ starting from the Block-Maxima. The resulting estimates are used to compute ΔC—as defined in Section 4.3—for every benchmark listed in Table 1, running as the control task on our platform.

On the other hand, as stated in Section 4.1, we assume STA is applied to estimate the HI-criticality WCETs, thus yielding higher bounds even on single-core platforms [1]. Specifically, we use a fixed multiplier $C_i^{HI} = CF \cdot C_i^{LO}$ (referred to as *criticality factor*), aligned with previous work on mixed-criticality scheduling [9, 32].

Schedulability analysis. Timing analysis results are used as input for schedulability analysis, conducted on synthetic task sets containing $n = 10$ tasks. The task set LO-criticality utilisation is varied between 0.05 and 0.975. For each utilisation value, 10^4 task sets are generated with the following parameters:

- *Periods and Deadlines* — The period of each task is randomly chosen from the set of harmonic values $\{20, 25, 40, 50, 80, 100, 200, 250, 400, 800, 1000\}$, at base frequencies 20 and 25. Deadlines are assumed implicit (see Section 4.1).
- *Task Utilisations* — The UUniFast [13] algorithm is employed, ensuring an unbiased, uniform distribution of utilisation values.
- *Execution Time Budgets* — LO-criticality budgets C^{LO} are estimated via MBPTA. For HI-criticality budgets $C^{HI} = CF \cdot C^{LO}$, instead, we employ a criticality factor of $CF = 2$.
- *Criticality* — Tasks were chosen to be either LO- or HI-criticality, with a probability $P_{HI} = 0.5$ of being HI.

Note that, by setting $CF = 2$ and $P_{HI} = 0.5$, HI-criticality utilisation is approximately equal to LO-criticality utilisation, which is a desirable property.

Furthermore, utilisation of victim tasks was limited to 0.7, to avoid very unbalanced sets in which one task runs for the majority of the time, with the others having execution time budgets close to zero.

For each task set we compute an *attacked* variant by adjusting victim task's utilisation u_n proportionally to the ratio between its LO-criticality WCET C^{LO} and the maximum observed execution time (WCOT) C^{max} of the attack scenario, as explained in Equation (2)

$$u_a = \frac{C^{max}}{C^{LO}} \, u_n \tag{2}$$

where u_a is the utilisation of the victim task on an attacked system; this effectively "inflates" the victim task proportionally to ΔC. The other tasks are unchanged.

Finally, we schedule the task sets with EDF-VD [7]. Its proven optimality for two criticality levels and its polynomial-time schedulability test make it a desirable choice; the final schedulability curves are produced by testing the EDF-VD schedulability condition on each randomly-generated task set. Schedulability loss S is then computed as the difference in area between the ground truth and attack scenarios.

Scheduling simulation. To assess the dynamic behaviour of the system, we employ the state-of-the-art real-time scheduling simulator SimSo [23], on which we implement EDF-VD. We simulate 10^4 *ms* of system execution and measure the R_{miss} and K_{HI} metrics, as defined in Section 4.3. Task sets are generated in the same way as in schedulability analysis.

5.1.4 Attack Impact Results

Figure 1 provides an overview on the distribution of each measured metric, divided by attack type and computed over the 97 attack scenarios that we evaluated during our experiments (Section 5.1.2). First, we observe that there is a visible correlation between the amount of WCET overrun (ΔC) and the schedulability loss (S); this is because there is an analytic relationship between the utilisation delta $|u_a - u_n|$ and the difference in area under the *ground truth* and *attacked* schedulability curves. Taking this into account, we found that SSA-DAs have negligible impact on every metric, except for `janne_complex` without bound checks, which leads to a 151.82% overrun (visible as a peak on the light blue distribution) and a 29% schedulability loss. Hence, we confirm the importance of explicitly restricting the input space of a task with bound checks, to reduce chances of anomalous temporal behaviour. DOAs and SSA-WCIs instead both show considerable overrun and schedulability loss, even if differently; DOAs are generally more impactful, with benchmarks like `bsort` exceeding the estimated WCET by 152.72% at 75% intensity. The results of `statemate` also tell us that incompatible branch behaviour DDMs can as well induce an increase in execution times, with a maximum $\Delta C = 229.68\%$ and $S = 37.96\%$.

Second, the miss rate distribution reveals a bimodal trend for SSA-WCIs, with scenarios attesting either around 30%, or at extremely low values around 5% as, for example, `qsort`. DOAs instead all show miss rates around 37%, with very few low-impact scenarios. Miss rates of this magnitude mean that the system fails to meet approximately one deadline out of every three—a level that, in a safety-critical context, could translate into sustained control-loop violations or loss of actuation deadlines with potentially catastrophic consequences.

Finally, we observe a different trend for HI-criticality mode time, with figures that do not concentrate around a well-defined peak value; SSA-WCIs cause generally shorter periods of degraded operation, with the exception of `bsort` and `insort` that settle at $\approx 10\%$. For DOAs the average is around 7% with peaks at 19% for `statemate`. We attribute the low absolute values for K_{HI} to the fact that we restore the criticality mode back to LO at the first CPU idle tick which, especially on low-utilisation task sets, can lead to interrupting degraded operation quickly after a disruption. Nonetheless, 1 or 2 seconds of degraded operation during a 10 second attack may still lead to a drop in QoS on a real-world platform, consequences of which shall be assessed on a per-scenario basis.

Overall, these results might translate into a variety of different physical-world effects, such as trajectory deviation, huge latency increases, etc. which can pose human life to a risk; we believe it is important to thoroughly characterise such effects in future research.

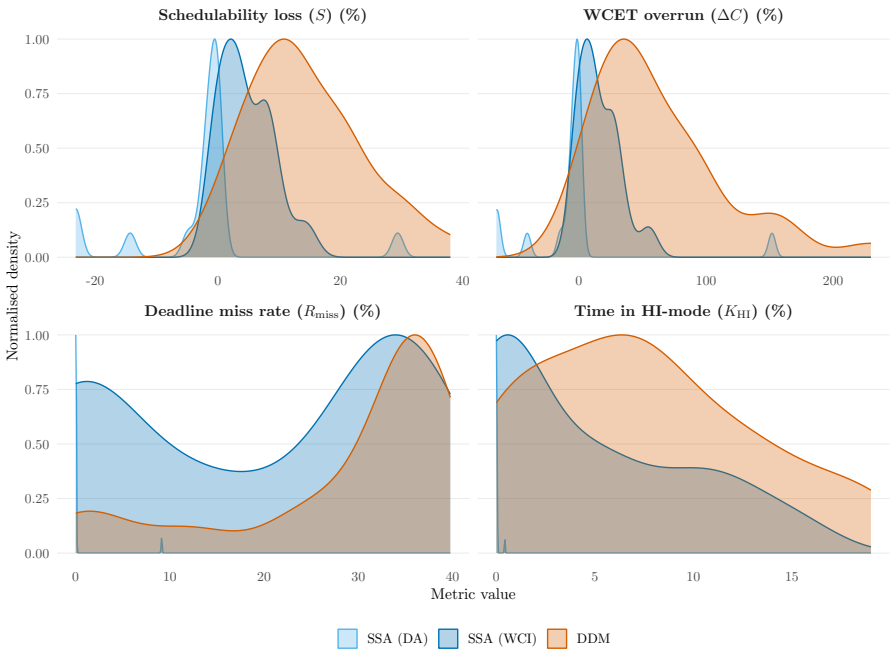


Figure 1 Distributions of attack impact metrics, computed over 97 distinct attack scenarios. Normalised density is computed by discretising the horizontal axis and then interpolated.

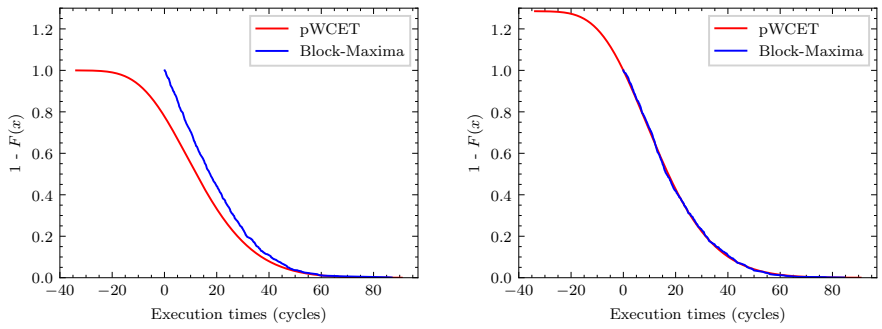
5.2 Temporal DoS Attack Detection Performance

To address $\mathcal{RQ}_2$, we compare our pWCET-based detector against the BoostIID approach by Yi and Dutt [64] using synthetic evaluation. This preliminary assessment focuses on algorithm behaviour under controlled conditions rather than claiming real-world applicability; Section 6 will discuss the implications of this synthetic setting. Since BoostIID source code and complete configuration parameters were unavailable, Appendix A details our implementation based on the published method. This reconstruction introduces inherent uncertainty in the comparison; performance differences may reflect implementation choices rather than fundamental algorithmic properties. However, we still find our results to confirm the BoostIID accuracy limitations pointed out by the original authors [64].

5.2.1 Experimental Setup

Tests are organised into different workloads, each corresponding to a synthetic task τ modelled after its pWCET distribution G_ξ. Each workload is composed of multiple execution scenarios, *i.e.*, different execution time traces $\mathcal{T}$. We generate a total of 36 tasks; for direct comparison, six of them have been taken directly from Yi and Dutt [64]. Another group of six tasks shares the same scale and location as SFM—the task identified as the most problematic in the BoostIID evaluation—but with increasing randomly-generated shape values between 0 and 50. Finally, we generate two additional sets of 12 random tasks each, having the same locations and scales but varying shapes between 0 and 50.

Execution time traces contain $M = 5000$ legitimate and $K = 500$ contiguous hijacked samples to simulate a temporal anomaly. Such alteration is obtained by randomly modifying pWCET distribution parameters up to 200% of their original values. This synthetic anomaly model differs from realistic attacks: real data-oriented and sensor spoofing attacks would



(a) Standard GEV distribution. **(b)** Truncated GEV distribution.

Figure 2 CCDFs of a 1000-sample Block-Maxima distribution and its reference GEV distribution.

induce deviations corresponding to specific execution paths, not uniformly random parameter shifts. Consequently, detection difficulty may be underestimated on synthetic benchmarks.

Anomaly detector implementation. We implement Algorithm 1 in Python. For our experiments, we set a $S = \frac{M}{B} = 1000$ sample Block-Maxima distribution with a block size of $B = 5$. We also implement GreedyKS-Add and GreedyKS-Remove, setting a granularity of $m = 10^3$ bins, and a confidence interval of $\alpha = 0.05$.

Timing traces are generated by extracting uniformly random execution time samples from the task's GEV distribution G_ξ. Specifically, extraction happens per-block; at first, B samples $bm_1, \ldots, bm_B$ are randomly extracted from G_ξ, to act as the maxima for each BM block. Next, $B - 1$ samples smaller than bm are extracted for each block. This ensures that the BM computed at runtime will effectively converge to G_ξ. We note that the problem of generating a synthetic execution time distribution from a reference GEV distribution is not trivial, due to biases introduced by extracting only positive samples; this effect is much more pronounced as the GEV distribution gets significant probability mass in correspondence of negative values. To overcome this, we restrict the support set over the non-negative real numbers, obtaining a Truncated GEV (TGEV) [6] distribution which ensures GoF as shown in Figure 2.

We instead configure our BoostIID implementation as detailed in Appendix A, and expand the detection buffer from 500 to 1000 samples for fair comparison.

Evaluation metrics. Both approaches are evaluated along the dimensions of *latency*, *accuracy* and *shape*. We define latency as the number of hijacked samples necessary to detect an anomaly, while accuracy is measured via the F_1 score, as described in Section 4.4. We refer both these metrics as a function of the shape (σ) of each task's pWCET distribution, which expresses the amount of variability in execution times. For each task, accuracy is computed over 20000 different scenarios, while latency is averaged. For each of these dimensions, we further define the notion of *robustness* (R) as the reciprocal of standard deviation across all tested workloads.

5.2.2 Anomaly Detection Performance Results

Results are displayed in Figure 3. We present two different views of the multi-dimensional evaluation space composed by latency (L), accuracy (F_1) and GEV shape (σ). Each data point on the graph corresponds to a different execution scenario.

Overall, we can state that the performance of our detection approach is orthogonal to that of current state-of-the-art approaches [64] with respect to the accuracy variability and average latency; this translates to completely disjoint clusters and a considerably high inter-cluster distance, clearly noticeable in Figure 3a.

From Figure 3a, we also observe that pWCET-based detection significantly improves accuracy. The lowest registered F_1 score is of 0.83, against the 0.74 of BoostIID; that is, a 12.16% increase. Average accuracy increases by 12%, transitioning from 0.82 to 0.93, while its robustness increases; standard deviation of the entire cluster of workloads goes from 0.07 to 0.033, marking a 47.14% reduction. Moreover, this accuracy drop happens as the pWCET distribution's shape *increases*, confirming the trend hinted in [64].

Both Figure 3a and Figure 3b, though, show a substantial increase in average latency for pWCET-based detection, going from a range of $[19.6, 147.45]$ in BoostIID to to $[177.14, 281.56]$. Following the same behaviour observed for accuracy, latency also increases as the pWCET distribution's shape increases. Each data point in the graph reports average latency for the respective workload; empirically, the standard deviation *within* data point is around 110 samples for our approach, and roughly 35 samples for the BoostIID validation prototype. Latency robustness for BoostIID is at 40.12 samples, while our pWCET monitoring is at 33.78 samples, improving by 15.8%.

Following these observations, we can state that such an increase in latency is partly attributable to the delay brought by the Block-Maxima technique, as well as the inherent simplicity of the KS test which, capturing more coarse-grained variations, requires more samples in order to detect a deviation of the Block-Maxima distribution from its reference GEV and signal an anomaly.

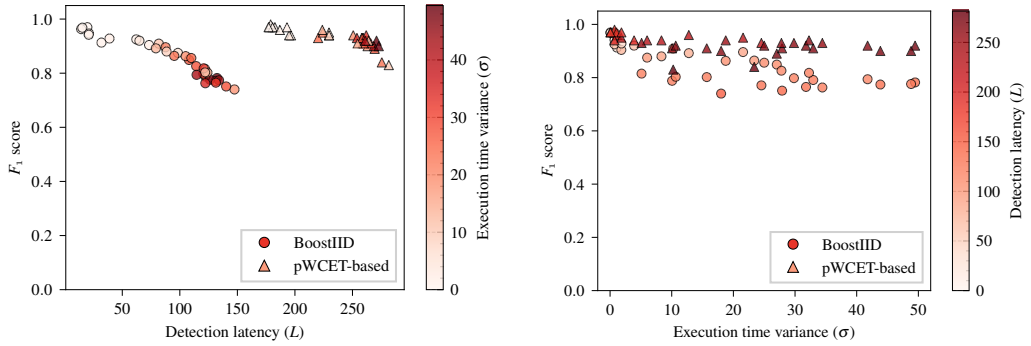
This increased latency presents a critical design trade-off that cannot be assessed in isolation from real-world constraints. As noted by Yi and Dutt [64], each task exhibits a maximum tolerable detection latency determined by the system's end-to-end latency, *i.e.*, the time available before a malicious perturbation reaches output actuators. Whether our $\approx$ 200-sample latency is acceptable depends entirely on this system-specific parameter, which varies dramatically across domains (autonomous vehicles may tolerate seconds; safety-critical medical devices may not). Consequently, deployment decisions will require validation on concrete systems with specified tolerance windows.

Memory consumption. It is worth noting that this approach, with the parameter choices selected for the evaluation (see Section 5.2.1), yields an average memory consumption of $\approx$88KB per-task; this is the result of employing data structures which memory complexities are purely a function of m and S, which are sizeable at design time. We believe our approach to be suitable for moderately resource-constrained MCUs equipped with tens of MBs of internal SRAM (a system with 10 tasks would indeed require less than 1MB to accomodate it), as well as more high-performance automotive development platforms that ship with external DDR4/5 memory interfaces [48].

6 Conclusion and Future Work

We affirmatively answer both research questions within the scope of our synthetic evaluation.

For $\mathcal{RQ}_1$: Input representativeness gaps in MBPTA create genuine attack vectors. Data-oriented and sensor spoofing attacks exploiting these gaps achieve substantial impact—deadline miss rates up to 39.76% and high-criticality mode duration of 18.63% over 10 seconds. These results, obtained through static and dynamic simulations, establish that



(a) Latency vs accuracy.

(b) Execution time variability vs accuracy.

Figure 3 pWCET-based vs BoostIID [64] temporal anomaly detection performance.

temporal DoS is a credible threat to mixed-criticality systems.

For $\mathcal{RQ}_2$: pWCET-based detection via Goodness-of-Fit testing shows promise but presents trade-offs. Compared to BoostIID, our approach achieves 12% average accuracy improvement and 61% reduced standard deviation, indicating more stable performance across heterogeneous workloads. However, detection latency increases to approximately 200 samples, an order of magnitude higher than existing methods. This trade-off is fundamental to the Block-Maxima technique and cannot be resolved without algorithmic redesign.

Our evaluation is mostly synthetic and therefore cannot fully validate real-world applicability. Anomalies are simulated via uniform parameter perturbations rather than realistic attack-induced execution path deviations. Detection latency is measured assuming contiguous attack samples, whereas real attackers would distribute malicious inputs across multiple task invocations, substantially increasing detection difficulty. The BoostIID comparison relies on our reconstruction of the original method, introducing potential implementation artifacts. These limitations mean our results demonstrate *algorithmic feasibility* under idealized conditions, not practical effectiveness. Nonetheless, we find our results to highlight the same limitations that were briefly introduced in [64], so we believe this investigation to still be meaningful.

Despite synthetic limitations, both research questions warrant validation on real systems. Practical Deployment of the detection approach requires that practitioners first determine end-to-end latency constraints for their specific domain—a system-dependent parameter that fundamentally determines viability. Therefore, we recommend future efforts to focus on the following areas:

- **Attack validation** — Implement realistic data-oriented and sensor spoofing attacks on commercial platforms (*e.g.*, ARM automotive challenge [5]), measuring temporal impact and physical-world consequences [39, 38].
- **Detection on non-synthetic traces** — Evaluate both approaches on timing traces from real attack scenarios, not synthetic distributions.
- **Kernel-level implementation** — Validate time and space complexity through actual kernel integration rather than userspace prototypes, assessing feasibility on resource-constrained platforms.
- **System-level analysis** — Study detection within full mixed-criticality mode-switching dynamics, capturing feedback between detection triggers and scheduling behavior.

- **Influence of scheduling policy and criticality levels** — As pointed out in Section 5.1.1, in this work we have chosen the optimal mixed-criticality scheduler for a two-level preemptive system; however, investigating how DOA and SSA impact varies across different scheduling policies—*e.g.*, the fixed priority AMC [8]. Considering systems with more than two levels of criticality would also be valuable.

This work establishes that temporal DoS attacks are feasible and characterizes a detection approach with distinct trade-offs. However, we stress on the fact that successful deployment of these techniques on concrete CPSs requires the real-world validation outlined above.

References

1 Jaume Abella, Damien Hardy, Isabelle Puaut, Eduardo Quiñones, and Francisco J. Cazorla. On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 266–275. IEEE Computer Society, 2014. doi:10.1109/ECRTS.2014.16.

2 Infineon Technologies AG. Infineon brings RISC-V to the automotive industry and is first to announce an automotive RISC-V microcontroller family. URL: https://www.infineon.com/press-release/2025/infatv202503-067.

3 Miguel Alcon, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. Timing of Autonomous Driving Software: Problem Analysis and Prospects for Future Solutions. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 267–280. IEEE, 2020. doi:10.1109/RTAS48715.2020.000-1.

4 IoT Analytics. Number of connected IoT devices growing 14% to 21.1 billion. URL: https://iot-analytics.com/number-connected-iot-devices/.

5 Matteo Andreozzi, Giacomo Gabrielli, Balaji Venu, and Giacomo Travaglini. Industrial Challenge 2022: A High-Performance Real-Time Case Study on Arm. In *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*, volume 231 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECRTS.2022.1.

6 Sándor Baran, Patrícia Szokol, and Marianna Szabó. Truncated generalized extreme value distribution-based ensemble model output statistics model for calibration of wind speed ensemble forecasts. *Environmetrics*, 32(6):e2678, 2021. doi:10.1002/env.2678.

7 Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Preemptive Uniprocessor Scheduling of Mixed-Criticality Sporadic Task Systems. *Journal of the ACM*, 62(2):14:1–14:33, 2015. doi:10.1145/2699435.

8 S.K. Baruah, A. Burns, and R.I. Davis. Response-Time Analysis for Mixed Criticality Systems. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 34–43. IEEE Computer Society, 2011. doi:10.1109/RTSS.2011.12.

9 Iain Bate, Alan Burns, and Robert I. Davis. A Bailout Protocol for Mixed Criticality Systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 259–268. IEEE Computer Society, 2015. doi:10.1109/ECRTS.2015.30.

10 Michael Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 357–367. IEEE, 2019. doi:10.1109/RTAS.2019.00037.

11 Nicolas Bellec, Simon Rokicki, and Isabelle Puaut. Attack Detection Through Monitoring of Timing Deviations in Embedded Real-Time Systems. In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:22, Dagstuhl, Germany, 2020. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECRTS.2020.8.

12 Kostiantyn Berezovskyi, Fabrice Guet, Luca Santinelli, Konstantinos Bletsas, and Eduardo Tovar. Measurement-based probabilistic timing analysis for graphics processor units. In *Architecture of Computing Systems – ARCS 2016*, volume 9637 of *Lecture Notes in Computer Science*, pages 223–236. Springer, 2016. URL: https://doi.org/10.1007/978-3-319-30695-7_17, doi:10.1007/978-3-319-30695-7_17.

13 Enrico Bini and Giorgio C. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, 30(1-2):129–154, 2005. doi:10.1007/s11241-005-0507-9.

14 Boudewijn Braams, Sebastian Altmeyer, and Andy D. Pimentel. EDiFy: An Execution time Distribution Finder. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 32:1–32:6, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3061639.3062233.

15 Douglas O. Cardoso and Thalis D. Galeno. Online evaluation of the Kolmogorov–Smirnov test on arbitrarily large samples. *Journal of Computational Science*, 67:101959, 2023. doi:10.1016/j.jocs.2023.101959.

16 Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. Probabilistic Worst-Case Timing Analysis: Taxonomy and Comprehensive Survey. *ACM Computing Surveys*, 52(1):14:1–14:35, 2019. doi:10.1145/3301283.

17 Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *20th USENIX Security Symposium (USENIX Security 11)*. USENIX Association, 2011. URL: http://static.usenix.org/events/sec11/tech/full_papers/Checkoway.pdf.

18 Chien-Ying Chen, Sibin Mohan, Rodolfo Pellizzoni, Rakesh B. Bobba, and Negar Kiyavash. A Novel Side-Channel in Real-Time Schedulers. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 90–102, Montreal, QC, Canada, 2019. IEEE. doi:10.1109/RTAS.2019.00016.

19 Shuo Chen, Jun Xu, and Emre C. Sezer. Non-Control-Data Attacks Are Realistic Threats. In *14th USENIX Security Symposium (USENIX Security 05)*. USENIX Association, 2005. URL: https://www.usenix.org/conference/14th-usenix-security-symposium/non-control-data-attacks-are-realistic-threats.

20 Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, San Francisco California USA, 2016. Association for Computing Machinery. doi:10.1145/2939672.2939785.

21 Long Cheng, Salman Ahmed, Hans Liljestrand, Thomas Nyman, Haipeng Cai, Trent Jaeger, N. Asokan, and Danfeng (Daphne) Yao. Exploitation Techniques for Data-oriented Attacks with Existing and Potential Defense Approaches. *ACM Transactions on Privacy and Security*, 24(4):26:1–26:36, 2021. doi:10.1145/3462699.

22 Long Cheng, Ke Tian, Danfeng Daphne Yao, Lui Sha, and Raheem A. Beyah. Checking is Believing: Event-Aware Program Anomaly Detection in Cyber-Physical Systems. *IEEE Transactions on Dependable and Secure Computing*, 18(2):825–842, 2021. Conference Name: IEEE Transactions on Dependable and Secure Computing. doi:10.1109/TDSC.2019.2906161.

23 Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms. In *Proceedings of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, page 6 p., Madrid, Spain, 2014. URL: https://inria.hal.science/hal-01052651/.

24 Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi -path Programs. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 91–101. IEEE Computer Society, 2012. doi:10.1109/ECRTS.2012.31.

25 Mitchell Duncan, Ao Li, Nathan Fisher, Ning Zhang, Ryan Gerdes, Tanmaya Mishra, and Thidapat Chantem. Mad monk: Arbitrary criticality escalation in mixed criticality real-time systems. In *2025 28th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 1–12. IEEE, 2025. doi:10.1109/ISORC65339.2025.00020.

26 Andreas Ermedahl, Johan Fredriksson, Jan Gustafsson, and Peter Altenbernd. Deriving the Worst-Case Execution Time Input Values. In *2009 21st Euromicro Conference on Real-Time Systems*, pages 45–54. IEEE Computer Society, 2009. doi:10.1109/ECRTS.2009.32.

27 Stefano Esposito, Massimo Violante, Marco Sozzi, Marco Terrone, and Massimo Traversone. A Novel Method for Online Detection of Faults Affecting Execution-Time in Multicore-Based Systems. *ACM Transactions on Embedded Computing Systems*, 16(4):94:1–94:19, 2017. doi:10.1145/3063313.

28 Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3):327–336, 1994. doi:10.1002/spe.4380240306.

29 Google. sim/tbm - Git at Google. URL: https://opensecura.googlesource.com/sim/tbm/.

30 Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET Benchmarks: Past, Present And Future . In *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, volume 15 of *Open Access Series in Informatics (OASIcs)*, pages 136–146, Dagstuhl, Germany, 2010. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.WCET.2010.136.

31 Mohammad Hamad, Zain A. H. Hammadeh, Selma Saidi, Vassilis Prevelakis, and Rolf Ernst. Prediction of abnormal temporal behavior in real-time systems. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 359–367, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3167132.3167172.

32 Biao Hu, Kai Huang, Pengcheng Huang, Lothar Thiele, and Alois Knoll. On-the-fly fast overrun budgeting for mixed-criticality systems. In *Proceedings of the 13th International Conference on Embedded Software*, EMSOFT '16, pages 25:1–25:10, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2968478.2968491.

33 Qun Huang and Patrick P.C. Lee. An experimental study of cascading performance interference in a virtualized environment. *SIGMETRICS Perform. Eval. Rev.*, 40(4):43–52, 2013. doi:10.1145/2479942.2479948.

34 Leonidas Kosmidis, Matina Maria Trompouki, Pau López Castillón, Eric Rufart Blasco, Javier Fernandez Salgado, and Andreas Jung. Open Source Software Randomisation Framework for Probabilistic WCET Prediction on Multicore CPUs, GPUs and Accelerators. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, volume 15227 of *Lecture Notes in Computer Science*, pages 247–260, Cham, 2024. Springer. URL: https://doi.org/10.1007/978-3-031-78380-7_20, doi:10.1007/978-3-031-78380-7_20.

35 Kristin Krüger, Marcus Völp, and Gerhard Fohler. Vulnerability Analysis and Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Systems. *LIPIcs, Volume 106, ECRTS 2018*, 106:22:1–22:17, 2018. doi:10.4230/LIPIcs.ECRTS.2018.22.

36 Denis Kwiatkowski, Peter C. B. Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1):159–178, 1992. doi:10.1016/0304-4076(92)90104-Y.

37 Ao Li, Marion Sudvarg, Han Liu, Zhiyuan Yu, Chris Gill, and Ning Zhang. PolyRhythm: Adaptive Tuning of a Multi-Channel Attack Template for Timing Interference. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 225–239. IEEE, 2022. doi:10.1109/RTSS55097.2022.00028.

38 Ao Li, Jinwen Wang, Sanjoy Baruah, Bruno Sinopoli, and Ning Zhang. An Empirical Study of Performance Interference: Timing Violation Patterns and Impacts. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 320–333. IEEE, 2024. doi:10.1109/RTAS61025.2024.00033.

39 Ao Li, Jinwen Wang, and Ning Zhang. Chronos: Timing Interference as a New Attack Vector on Autonomous Cyber-physical Systems. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 2426–2428, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3460120.3485350.

40 George Lima, Dário Dias, and Edna Barros. Extreme Value Theory for Estimating Task Execution Time Bounds: A Careful Look. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 200–211. IEEE Computer Society, 2016. doi:10.1109/ECRTS.2016.20.

41 Tamara Lugo, Santiago Lozano, Javier Fernández, and Jesus Carretero. A Survey of Techniques for Reducing Interference in Real- Time Applications on Multicore Platforms. *IEEE Access*, 10:21853–21882, 2022. doi:10.1109/ACCESS.2022.3151891.

42 Rouhollah Mahfouzi, Amir Aminifar, Soheil Samii, Mathias Payer, Petru Eles, and Zebo Peng. Butterfly Attack: Adversarial Manipulation of Temporal Properties of Cyber-Physical Systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 93–106. IEEE, 2019. doi:10.1109/RTSS46320.2019.00019.

43 Blau Manau, Sergi Vilardell, Isabel Serra, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. Detecting low-density mixtures in high-quantile tails for pWCET estimation. In *37th Euromicro Conference on Real-Time Systems (ECRTS 2025)*, volume 335 of *LIPIcs*, pages 20:1–20:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ECRTS.2025.20.

44 Brayden McDonald and Frank Mueller. T-SYS: Timed-Based System Security for Real-Time Kernels. In *2022 ACM/IEEE 13th International Conference on Cyber- Physical Systems (ICCPS)*, pages 247–258. IEEE, 2022. doi:10.1109/ICCPS54341.2022.00029.

45 Samuel Mergendahl, Samuel Jero, Bryan C. Ward, Juliana Furgala, Gabriel Parmer, and Richard Skowyra. The Thundering Herd: Amplifying Kernel Interference to Attack Response Times. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 95–107. IEEE, 2022. doi:10.1109/RTAS54340.2022.00016.

46 Enrico Mezzetti, Niklas Holsti, Antoine Colin, Guillem Bernat, and Tullio Vardanega. Attacking the sources of unpredictability in the instruction cache behavior. In *16th International Conference on Real-Time and Network Systems (RTNS 2008)*, 2008. URL: https://inria.hal.science/inria-00336526/.

47 R. MISES. La distribution de la plus grande de n valeurs. *Rev. Math. Union Interbalcanique*, 1:141–160, 1936. URL: https://cir.nii.ac.jp/crid/1571980074415754368.

48 NVIDIA. DRIVE AGX Autonomous Vehicle Development Platform. URL: https://developer.nvidia.com/drive/agx.

49 Sangbin Park, Youngjoon Kim, and Dong Hoon Lee. SCVMON: Data-oriented attack recovery for RVs based on safety-critical variable monitoring. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 547–563, Hong Kong China, 2023. Association for Computing Machinery. doi:10.1145/3607199.3607221.

50 Youngseok Park, Yunmok Son, Hocheol Shin, Dohyun Kim, and Yongdae Kim. This Ain't Your Dose: Sensor Spoofing Attack on Medical Infusion Pump. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. USENIX Association, 2016. URL: https://www.usenix.org/conference/woot16/workshop-program/presentation/park.

51 Federico Reghenzani, Giuseppe Massari, and William Fornaciari. chronovise: Measurement-Based Probabilistic Timing Analysis framework. *Journal of Open Source Software*, 3(28):711, 2018. doi:10.21105/joss.00711.

52 Federico Reghenzani, Luca Santinelli, and William Fornaciari. Dealing with uncertainty in pWCET estimations. *ACM Transactions on Embedded Computing Systems*, 19(5):33:1–33:23, 2020. doi:10.1145/3396234.

53 Renode. Renode. URL: https://renode.io/.

54 Luca Santinelli, Fabrice Guet, and Jerome Morio. Revising Measurement-Based Probabilistic Timing Analysis. In *2017 IEEE Real-Time and Embedded Technology and Applications*

Symposium (RTAS), pages 199–208. IEEE Computer Society, 2017. doi:10.1109/RTAS.2017.16.

55 Luca Santinelli, Jérôme Morio, Guillaume Dufour, and Damien Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *14th International Workshop on Worst-Case Execution Time Analysis*, volume 39 of *Open Access Series in Informatics (OASIcs)*, pages 21–30, Dagstuhl, Germany, 2014. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.WCET.2014.21.

56 Antonio Savino, Gautam Gala, Marcello Cinque, and Gerhard Fohler. Multicore DRAM Bank-& Row-Conflict Bomb for Timing Attacks in Mixed-Criticality Systems. In *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 1–10. IEEE, 2024. doi:10.1109/ISORC61049.2024.10551337.

57 NXP Semiconductors. Why NXP Is All In on Zephyr: A Real-Time OS Built for the Future. URL: https://www.nxp.com/company/about-nxp/smarter-world-blog/BL-NXP-ZEPHYR-REAL-TIME-OS.

58 Jonathan S. Shapiro. Vulnerabilities in synchronous IPC designs. In *2003 Symposium on Security and Privacy, 2003.*, pages 251–262. IEEE Computer Society, 2003. doi:10.1109/SECPRI.2003.1199341.

59 Steve Vestal. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 239–243. IEEE Computer Society, 2007. doi:10.1109/RTSS.2007.47.

60 Yujie Wang, Ao Li, Jinwen Wang, Sanjoy Baruah, and Ning Zhang. Opportunistic Data Flow Integrity for Real-time Cyber -physical Systems Using Worst Case Execution Time Reservation. In *33rd USENIX Security Symposium (USENIX Security '24)*, pages 6615–6632. USENIX Association, 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/wang-yujie.

61 Marilyn Wolf and Dimitrios Serpanos. Safety and Security in Cyber-Physical Systems and Internet -of-Things Systems. *Proceedings of the IEEE*, 106(1):9–20, 2018. doi:10.1109/JPROC.2017.2781198.

62 Jingyu Xin, Vir V. Phoha, and Asif Salekin. Combating False Data Injection Attacks on Human-Centric Sensing Applications. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 6(2):83:1–83:22, 2022. doi:10.1145/3534577.

63 Yuan Xu, Xingshuo Han, Gelei Deng, Jiwei Li, Yang Liu, and Tianwei Zhang. SoK: Rethinking Sensor Spoofing Attacks against Robotic Vehicles from a Systematic View. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 1082–1100. IEEE, 2023. doi:10.1109/EuroSP57164.2023.00067.

64 Saehanseul Yi and Nikil Dutt. BoostIID: Fault-Agnostic Online Detection of WCET Changes in Autonomous Driving. In *Proceedings of the 29th Asia and South Pacific Design Automation Conference*, ASPDAC '24, pages 704–709. IEEE, 2024. doi:10.1109/ASP-DAC58780.2024.10473866.

65 Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Mihai Christodorescu, and Lui Sha. Learning Execution Contexts from System Call Distribution for Anomaly Detection in Smart Embedded System. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 191–196. Association for Computing Machinery, 2017. doi:10.1145/3054977.3054999.

66 Christopher Zimmer, Balasubramany Bhat, Frank Mueller, and Sibin Mohan. Intrusion Detection for CPS Real-Time Controllers. In *Cyber Physical Systems Approach to Smart Electric Power Grid*, pages 329–358. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. doi:10.1007/978-3-662-45928-7_12.

Table 3 Hyperparameter tuning configuration for BoostIID's detector.

Hyperparameter	Value Space
`max_depth`	$\{3, 5, 7\}$
`min_child_weight`	$\{1, 3, 5\}$
`gamma`	$\{0, 0.1, 0.2\}$
`subsample`	$\{0.8, 0.9, 1.0\}$
`colsample_bytree`	$\{0.8, 0.9, 1.0\}$
`reg_alpha`	$\{0.0, 0.1, 1.0\}$

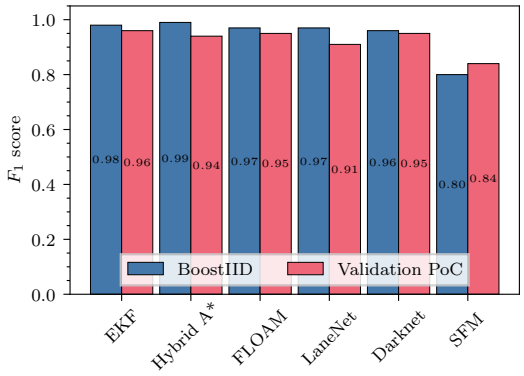


Figure 4 Validation of the results obtained by Yi and Dutt [64].

A BoostIID Validation

As reported in Section 5.2, the authors of [64] have not made the source code publicly available; moreover, the exact configuration for the classifier (implemented using XGBoost [20]) is not thoroughly specified. To perform our measurements, we implemented a validation prototype making reasonable design assumptions where they were unclear or absent. First, we set the confidence level for iid tests to $\alpha = 0.05$ and we train a separate model for each task. Second, we perform hyperparameter tuning of the boosting configuration by tweaking the needed parameters with the GridSearchCV library, as reported in Table 3.

We supply the following feature vector as an input to the classifier

$$(KPSS_s, KPSS_p, KPSS_r, RS_s, RS_r, LB_s, LB_p, LB_r, \mu)$$

where s indicates the test statistic, p the resulting p-value and r is the test outcome (*i.e.*, null hypothesis accepted/rejected); μ instead refers to the arithmetic mean of the execution time samples in the scenario.

We run this implementation against the same set of tasks used in the original evaluation, obtaining a maximum F_1-score relative error of 6.18%, in correspondence of the LaneNet task, as reported in Figure 4. Therefore, we believe to have obtained a suitable candidate for performance comparison.

B Modified GreedyKS Algorithm

The listings at Algorithm 3 and Algorithm 2 detail our modified version of GreedyKS [15], which is employed as the implementation of KS.Add and KS.Remove in Algorithm 1. For

Algorithm 2 GreedyKS-Add

Require: x: Sample to process
Require: n: Number of elements processed so far
Require: H_b^{min}, H_b^{MAX}: min-/max-heaps storing ordered observations for bin b
$f \leftarrow G_\xi(x)$
$n \leftarrow n + 1$
$b \leftarrow \lfloor f \cdot m \rfloor$
FenwickTreeUpdate $(E, b, 1)$
HeapPushMax $\left(H_b^{MAX}, f\right)$
HeapPushMin $\left(H_b^{min}, f\right)$
$D^- \leftarrow$ HeapTop $\left(H_{oldN}^{min}\right) - \left(\sum_{i=0}^{oldN-1} num_i\right)/n$ {Update current D^-}
$D_{new}^- \leftarrow$ HeapTop $\left(H_b^{min}\right) - \left(\sum_{i=0}^{b-1} num_i\right)/n$ {Compute $G_\xi(x) - E(x)$}
if $D_{new}^- > D^-$ **then**
    $D^- \leftarrow D_{new}^-$
    $oldN \leftarrow b$
end if
$D^+ \leftarrow \left(\sum_{i=0}^{oldP} num_i\right)/n -$ HeapTop $\left(H_{oldP}^{MAX}\right)$ {Update current D^+}
$D_{new}^+ \leftarrow \left(\sum_{i=0}^{b} num_i\right)/n -$ HeapTop $\left(H_b^{MAX}\right)$ {Compute $E(x) - G_\xi(x)$}
if $D_{new}^+ > D^+$ **then**
    $D^+ \leftarrow D_{new}^+$
    $oldP \leftarrow b$
end if

Algorithm 3 GreedyKS-Remove

Require: x: Sample to remove
Require: n: Number of elements processed so far
Require: H_b^{min}, H_b^{MAX}: min-/max-heaps storing ordered observations for bin b
$f \leftarrow G_\xi(x)$
$n \leftarrow n - 1$
$b \leftarrow \lfloor f \cdot m \rfloor$
FenwickTreeUpdate $(E, b, -1)$
HeapDelete $\left(H_b^{MAX}, f\right)$
HeapDelete $\left(H_b^{min}, f\right)$

the sake of clarity, the signatures of these functions are more detailed than those reported in Algorithm 1.

C MBPTA Results

Table 4 details results from the timing analysis phase of the measurement pipeline (Section 5.1.3).

Table 4 MBPTA estimation results. Execution times are reported in clock cycles.

Benchmark	KPSS	BDS	RS	WCOT	WCET
`bsort`	✓	✓	✓	36685	36905.7
`insort`	✓	✓	✓	20703	21152
`qsort`	✗	✓	✓	3412	3496.4
`prime`	✓	✓	✓	5442	6317.64
`janne_complex`	✓	✓	✓	941	941.9
`expint`	✓	✓	✓	20129	20232.6

✓: test passed. ✗: test failed.