RingTrace: Model-Free Spatio-Temporal Detection and Core Attribution for Cache Side-Channels

Zhaoxue Zhang[1], Yiwen Gao[1,✉], Yuejun Liu[1], Wei Cheng[1,2], Yongbin Zhou[1,3], and Sylvain Guilley[2,4]
[1]School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing, China
[2]LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France
[3]Institute of Information Engineering, Chinese Academy of Sciences, China
[4]Secure-IC S.A.S., Cesson-Sévigné, France, sylvain.guilley@secure-ic.com
{zhangzhaoxue, gaoyiwen, liuyuejun, wei.cheng, zhouyongbin}@njust.edu.cn

***Abstract*—The widespread adoption of shared last-level caches in modern multi-core processors improves memory efficiency but introduces timing channels between untrusted programs. In particular, Flush+Reload serves as a high-resolution cache side channel and a fundamental data leakage mechanism in transient-execution attacks. Existing detection approaches predominantly rely on aggregate hardware performance counters, which discard spatial context, fail to locate the malicious core, and often confuse stealthy attacks with benign cache-intensive workloads. This paper addresses these gaps by proposing RingTrace, a system-level monitoring framework that synergizes core-local and uncore telemetry. We capture and normalize per-core cache misses alongside distributed slice activity across uncore CBoxes. Through Joint Spatio-Temporal Attribution (JSTA), we formulate an interpretable joint-activity matrix that captures spatial concentration and co-activity without requiring model training. In the hierarchical response, the largest entry of this matrix serves as the initial screening score, while its row sums identify a candidate core. Our approach reliably identifies victim-independent flush activity prior to secret leakage, effectively suppresses false alarms under severe concurrent memory pressure, and achieves rapid process confirmation with negligible application runtime overhead on commodity processors. This work introduces an interpretable and tamper-resilient hardware-assisted defense model, bridging local CPU execution with physical uncore traffic to provide robust protection against cache side channels.**

***Index Terms*—cache side channels, Flush+Reload, hardware performance counters, uncore, attack detection, core attribution**

I. Introduction

Shared last-level caches (LLCs) in modern CPUs improve utilization but create timing channels among mutually untrusted programs. Flush+Reload (F+R) infers access to a shared cache line from reload latency [1]; Prime+Probe observes cache-set contention without shared memory [2]; and Flush+Flush uses the timing of the flush instruction itself [3]. Beyond direct spying, F+R serves as a foundational receiver across transient-execution attacks, including Spectre, Meltdown, and Foreshadow [4]–[6]. Importantly, because the receiver loop repeatedly executes cache operations regardless of victim timing, monitoring this recurring receiver provides a focused defense point without requiring recognition of each speculative gadget.

Hardware Performance Counters (HPCs) provide a practical observation mechanism. Prior detectors have used cache misses, cache references, instruction counts, and related events to identify anomalous execution [7]–[10]. However, system-wide aggregation removes spatial context and can produce false alarms under complex workloads, whereas continuous per-process monitoring introduces substantial runtime overhead. Distinguishing attacks from benign cache-intensive activity while remaining resistant to evasion therefore remains a central challenge for HPC-based defenses [11].

Intel processors expose a complementary telemetry domain. Core PMUs attribute events to logical processors, whereas uncore CBox PMUs report model-specific activity at distributed LLC and home-agent slices. Ring contention is a known side channel [12], and recent work has shown that mesh traffic carries spatial signatures of cache attacks on server CPUs [13]. These findings indicate that uncore traffic can complement core-local evidence across different interconnect topologies. We therefore investigate whether a lightweight monitor can detect victim-independent flush activity while joint Core–CBox telemetry provides interpretable spatial evidence on commodity ring architectures.

RingTrace addresses this question by converting concurrent per-core and per-CBox counts into normalized activity vectors and accumulating their per-interval outer products. On processors exposing compatible Core and uncore PMUs, RingTrace uses the existing Linux `perf` interface and requires no processor hardware modification. The resulting joint-activity map yields a screening score for anomaly detection while preserving Core information for attacker-core attribution. This construction establishes an inspectable, cross-domain spatial footprint without requiring complex machine learning models, offering a principled foundation for tamper-resilient cache side-channel defense.

This paper makes three contributions:

- **Cross-Domain Telemetry Pipeline:** We design a lightweight, non-invasive telemetry framework that concurrently captures and normalizes core-local LLC misses and distributed uncore CBox traffic on commodity hardware. Per-interval normalization effectively suppresses common-mode background fluctuations while preserving critical spatial concentration.
- **Model-Free Hierarchical Detection:** We propose Joint Spatio-Temporal Attribution (JSTA), an interpretable,

mathematically bounded framework that requires no machine learning training. By decoupling anomaly screening via the joint peak from culprit identification via the analytical Core marginal, RingTrace achieves victim-independent detection and establishes an inspectable, tamper-resilient physical evidence trail.
- **Comprehensive Physical Evaluation:** We evaluate RingTrace on a physical multi-core platform across benign, memory-intensive, and adversarial workloads. Our results demonstrate that RingTrace reliably detects stealthy flush loops prior to secret leakage, preserves 100% core attribution accuracy under severe concurrent memory stress, and incurs negligible application runtime overhead.

II. Background, Related Work, and Threat Model

A. Microarchitectural Telemetry and Related Defenses

In an F+R attack, the adversary repeatedly evicts a shared cache line, waits for a victim access, and measures the subsequent reload latency [1]. Although a standalone flush loop does not observe victim accesses, its repeated evictions still perturb the memory hierarchy. In several transient covert channels, including those used by Spectre, Meltdown, and Foreshadow, an F+R receiver converts secret-dependent cache state into observable timing differences [4]–[6]. RingTrace targets this recurring receiver activity, which we refer to as *victim-independent flush activity*.

These microarchitectural operations manifest across two complementary hardware views. Core PMUs track events local to logical processors; we monitor per-core LLC load misses, which reflect memory reloads resulting from cache line evictions. Concurrently, modern Intel multi-core processors physically partition the shared LLC into distributed slices to alleviate access bottlenecks. Each slice is paired with a dedicated uncore controller known as a CBox [14]. Positioned along the on-chip interconnect (e.g., the ring bus), each CBox maintains cache coherence for its assigned slice, coordinates cross-core bus transactions, and integrates independent uncore PMUs. Whenever a core triggers an LLC miss, the request traverses the interconnect and is routed to a designated CBox governed by a hardware-level physical address hashing function. Capturing these model-specific CBox activity counters therefore allows us to directly observe slice-level traffic across the interconnect.

Table I summarizes representative counter-based cache-attack defenses. CloudRadar and NIGHTs-WATCH use architectural events to detect anomalous cache activity [7], [8]. Earlier work by Demme et al. established the broader feasibility of counter-based malware detection [16], while Chiappetta et al. applied counter correlation and learning to cache-attack detection [17]. CacheShield instead monitors interference from within a protected application [18]. Fight Hardware associates performance-counter activity with suspected processes to guide mitigation [10], whereas SCAGuard combines control-flow information, runtime counters, and cache-state transitions [15].

As systematized by Kosasih et al. [11], system-wide aggregation discards spatial context, whereas continuous per-process monitoring imposes prohibitive runtime overhead. RT-Sniper addresses this trade-off through a core-first hierarchy, profiling processes only after identifying a suspicious core via per-core counts [9]. RingTrace extends this hierarchical organization by establishing a mathematically coupled Core–CBox telemetry representation, enabling victim-independent screening without requiring machine learning models.

The on-chip interconnect provides an orthogonal spatial observation surface. Paccagnella et al. demonstrated that ring contention leaks fine-grained cross-core activity [12], while Bian and Cho utilized deep learning over processor-mesh traffic to classify multiple cache attacks [13]. Unlike prior works that rely on complex model training or treat uncore telemetry as an isolated signal, RingTrace analytically projects joint Core–CBox outer products into an interpretable matrix, preserving local core provenance alongside distributed slice footprints.

Interconnect topology shapes both leakage paths and monitorable signals. Paccagnella et al. characterized contention on Intel ring interconnects, whereas Dai et al. studied placement-dependent contention on server-class mesh fabrics [12], [19]. These results do not imply that ring and mesh event semantics are interchangeable. Instead, RingTrace pairs a source-local signal with a topology-specific uncore view on a common timeline. CBox activity provides this view on our evaluated ring processor, while a mesh deployment would require suitable CHA/mesh events and platform-specific calibration. Thus, the transferable idea is the joint organization of local and fabric-level evidence, not the unchanged reuse of a particular raw CBox event.

B. Threat Model and Defense Scope

We assume a local, unprivileged adversary executing a flush-based receiver loop on a multi-core host, potentially co-located with a victim. In our baseline evaluation, the attacker process is pinned to a single physical core. The defender is a privileged host monitor reading Core and CBox PMUs via Linux `perf`. The monitor operates in an address- and victim-agnostic manner: it requires no prior knowledge of target memory addresses, cryptographic keys, victim execution phases, or whether a victim is currently active. Advanced evasion scenarios—such as rapid thread migration, multi-process collusion, and adversarial PMU counterbalancing—are decoupled from this baseline study and discussed in Section V.

The trusted computing base (TCB) encompasses the host kernel, the hardware PMU interface, and the monitoring daemon. The adversary can arbitrarily control the timing and frequency of the flush loop, but cannot compromise the kernel or manipulate hardware counter registers. The primary defense objective is twofold: (1) reliably screen time windows for anomalous victim-independent flush activity, and (2) pinpoint the culprit physical core without disrupting benign workloads. Because initial screening is address-agnostic, it identifies an anomalous execution context rather than semantic program in-

TABLE I
FUNCTIONAL COMPARISON WITH REPRESENTATIVE CACHE-ATTACK DEFENSES.

Work	Telemetry / model	Evaluated attacks	Benign or stress evaluation	Attribution output	Reported runtime cost
CloudRadar [7]	HPC-based online detection	F+R, Prime+Probe	Co-located cloud workloads	Attack detection across VMs	Not directly comparable
NIGHTs-WATCH [8]	HPCs with machine learning	F+R, F+F	Workloads at varying load levels	Attack presence	Not directly comparable
RT-Sniper [9]	Per-core then per-process HPCs	F+R, Prime+Probe	SPEC2017, LMBench, PARSEC, and stress	Candidate core → process	2.05–3.76%
Fight Hardware [10]	System-wide process HPCs	Cache side channels	System-wide application execution	Suspected process and mitigation	Reported; protocol differs
SCAGuard [15]	CFG, runtime HPCs, and cache-state transitions	F+R, Prime+Probe, Spectre-like and obfuscated variants	400 diverse benign programs	Attack detection and family classification	Not a comparable online monitor
Bian and Cho [13]	Mesh traffic with deep learning	Multiple cache attacks	Concurrent workloads	Attack detection	Not reported comparably
RingTrace	**Per-core LLC misses plus per-CBox activity**	**F+R, with and without a victim**	**Idle, RSA victim, 8-core memory stress; SPEC cost**	**Core label, joint Core–CBox evidence, and process confirmation**	**2.805% mean across five SPEC benchmarks**

tent. Therefore, RingTrace adopts a two-stage paradigm: low-tier screening alerts trigger a lightweight, 0.5-s process-level confirmation on the candidate core to verify event ownership (`comm, pid`) before any disruptive mitigation is enacted.

III. RINGTRACE DESIGN

A. Telemetry Processing and Joint Spatio-Temporal Attribution

Figure 1 illustrates the data path of RingTrace. The system concurrently samples core and uncore views via a common clock, cleans incomplete records, and computes a normalized joint-activity matrix per analysis window.

Let $x_k(t)$ denote LLC load misses on CPU $k \in \{0, \dots, K-1\}$, and let $y_j(t)$ denote the raw activity count at CBox $j \in \{0, \dots, J-1\}$ (with $K = J = 8$ on our evaluated platform). A single `perf stat` invocation polls both telemetry streams every $\Delta = 50$ ms, ensuring a synchronized timeline. The parser validates telemetry integrity: it rejects non-finite values and enforces that each retained timestamp contains a complete set of K core and J CBox samples. Incomplete interior timestamps are discarded rather than artificially imputed to prevent manufacturing false spatial shares; partial terminal intervals are truncated. Across 1,300 benchmark traces, all interior samples remained intact, with each 3-s horizon containing 57–59 usable intervals.

Because raw event counts fluctuate drastically with dynamic CPU frequency scaling (DVFS) and system load, RingTrace measures each source's relative activity share within each sampling interval. Let $X(t) = \sum_{a=0}^{K-1} x_a(t)$ and $Y(t) = \sum_{b=0}^{J-1} y_b(t)$. The normalized shares are defined as

$$p_k(t) = \begin{cases} x_k(t)/X(t), & X(t) > 0, \\ 0, & X(t) = 0, \end{cases} \qquad q_j(t) = \begin{cases} y_j(t)/Y(t), & Y(t) > 0, \\ 0, & Y(t) = 0. \end{cases} \tag{1}$$

A genuine zero total thus produces a zero share vector. Whenever a stream contains activity, its shares sum exactly to one, suppressing common-mode amplitude shifts while preserving spatial concentration.

For a recorded window containing T valid synchronized intervals, RingTrace computes the Joint Spatio-Temporal Attribution (JSTA) matrix by averaging per-interval outer products:

$$R_{k,j} = \frac{1}{T} \sum_{t=1}^{T} p_k(t) q_j(t). \tag{2}$$

Each entry $R_{k,j} \in [0, 1]$ measures the joint activity between Core k and CBox j. Its marginals and temporal decomposition satisfy

$$\begin{aligned} \sum_j R_{k,j} &= \frac{1}{T} \sum_{t=1}^{T} p_k(t) \mathbb{I}\{Y(t) > 0\}, \\ \sum_k R_{k,j} &= \frac{1}{T} \sum_{t=1}^{T} q_j(t) \mathbb{I}\{X(t) > 0\}, \\ R_{k,j} &= \bar{p}_k \bar{q}_j + \mathrm{Cov}_t(p_k, q_j), \end{aligned} \tag{3}$$

Here, $\mathbb{I}\{\cdot\}$ denotes the indicator function. For $Z \in \{X, Y\}$, it is defined as

$$\mathbb{I}\{Z(t) > 0\} = \begin{cases} 1, & Z(t) > 0, \\ 0, & Z(t) = 0. \end{cases}$$

Moreover, $\bar{p}_k = T^{-1} \sum_t p_k(t)$, $\bar{q}_j = T^{-1} \sum_t q_j(t)$, and Cov_t uses the population normalization $1/T$. When both streams have positive totals in every retained interval, the row and column sums recover $\bar{p}_k$ and $\bar{q}_j$ exactly. The matrix retains spatial concentration and same-interval co-activity in a directly inspectable representation.

Maintaining the full matrix provides a cross-domain evidence chain that is more difficult to perturb consistently across core and uncore observations. A Core-only monitor omits the uncore footprint and may therefore be affected by manipulation of core-local events, such as counterbalancing [20]. Conversely, a CBox-only monitor loses core provenance. Although $R_{k,j}$ does not represent the direct routing of requests from a particular core to a particular CBox, it captures their

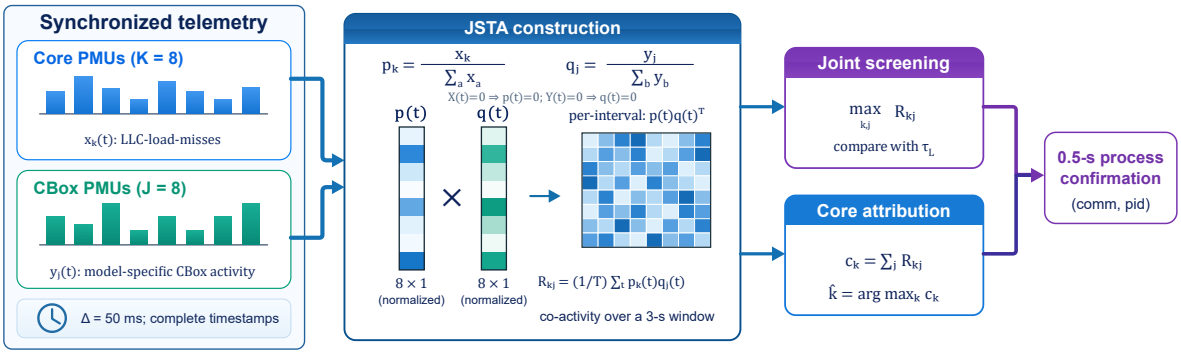

Fig. 1. Overview of RingTrace. A single `perf` invocation collects synchronized Core LLC-load-miss and CBox activity measurements. After alignment and normalization, their outer products are averaged to form the JSTA matrix. The maximum entry is evaluated against the screening threshold; row-wise aggregation then attributes the event to a candidate core for process-level confirmation.

same-interval physical co-activity. Reproducing a benign joint pattern would therefore require an attacker to shape both core-local LLC misses and their distribution across LLC slices, whose mapping is governed by physical-address hashing [14]. Normalization also establishes an intuitive baseline: when both streams have positive totals and uniform shares, each cell contributes $1/(KJ) = 1/64 \approx 0.0156$. Deviations from this baseline indicate spatial concentration.

B. Hierarchical Detection and Online Sliding Response

RingTrace enforces a hierarchical decision rule that decouples anomaly screening from core attribution. Summing across columns yields a score for each core:

$$c_k = \sum_{j=0}^{J-1} R_{k,j}, \qquad \hat{k} = \arg\max_k c_k, \qquad c^* = c_{\hat{k}}. \tag{4}$$

When every retained interval has positive CBox activity, $c_k = \bar{p}_k$ exactly. The detector raises a screening alert when the joint matrix maximum $s_J^* = \max_{k,j} R_{k,j}$ exceeds a calibrated threshold τ; $\hat{k}$ then labels the candidate attacker core. Thus, the joint Core–CBox score performs screening, while the row sums perform attribution. We establish $\tau_L = 0.05$ (3.2 times the uniform baseline $1/64$) as the sensitive screening point to capture victim-absent flush activity, while Section IV-C discusses the high-confidence tier τ_H.

To support low-latency online deployment, RingTrace maintains an incremental sliding buffer of $M = W/\Delta = 60$ samples (for a horizon $W = 3$ s and stride $\Delta = 50$ ms). Rather than recomputing the entire trace, the running matrix updates incrementally:

$$R^{(n)} = R^{(n-1)} + \frac{p(n)q(n)^\mathsf{T} - p(n-M)q(n-M)^\mathsf{T}}{M}. \tag{5}$$

This rank-one update requires constant time in horizon length and only $O(KJ)$ operations (64 cell touches), eliminating recomputation overhead.

The update stride $\Delta = 50$ ms defines the decision refresh rate rather than a rigid detection latency. The empirical time-to-first-alert is modeled as $T_{\text{alert}} = m\Delta + T_{\text{update}}$, where m is the number of post-onset samples required to cross τ_L, and T_{update} is the negligible update time. Let $\hat{j} = \arg\max_j R_{\hat{k},j}$ be the dominant CBox coordinate for candidate core $\hat{k}$. If cell $(\hat{k}, \hat{j})$ contributes $r \approx p_{\hat{k}} q_{\hat{j}}$ per sample, the sliding score reaches τ_L once $m = \lceil \tau_L M / r \rceil$. For an aggressive flush loop where $r \approx 1$, this yields a rapid theoretical lower bound of $m = 3$ samples (≈ 150 ms) to trigger an alert. Screened candidate cores then proceed to lightweight 0.5-s process profiling for definitive confirmation before mitigation.

IV. Evaluation

A. Prototype and Experimental Setup

We implement RingTrace as a lightweight monitor using a single system-wide Linux `perf` process to concurrently sample per-core LLC misses and raw uncore activity across all eight CBoxes. The collection pipeline polls both streams every $\Delta = 50$ ms and computes normalized JSTA matrices via Eqs. (1)–(2). Upon triggering an alert, the prototype initiates an optional 0.5-s `perf record` burst pinned to the candidate core to confirm process identity. First-stage hardware detection is evaluated independently of this confirmation step.

Experiments are conducted on an Intel Core i7-11700K processor featuring eight physical cores and a 16-MB shared LLC. Simultaneous multithreading (SMT) is disabled in firmware, leaving eight online logical CPUs (0–7) that map directly to physical cores. Table II summarizes the platform configuration. Crucially, CPU frequency scaling operates under the default `powersave` governor with Turbo Boost enabled, demonstrating that RingTrace's per-interval normalization remains resilient against dynamic DVFS fluctuations.

We evaluated five distinct system states across 1,300 benchmark windows: (1) *Idle* (50 windows without workloads); (2) *Victim-only* ($8 \times 50 = 400$ windows, sweeping the victim across cores 0–7); (3) *Attacker-only* ($8 \times 50 = 400$ windows, sweeping the flush loop across cores 0–7); (4)

TABLE II
EVALUATION PLATFORM AND CONFIGURATION.

Item	Configuration
Processor	Intel Core i7-11700K; 8 cores; 16 MB shared LLC; SMT disabled
System	Ubuntu 20.04.6; Linux 5.15.0-139; `perf` 5.15.178
Telemetry	`LLC-load-misses`; model-specific raw `uncore_cbox_N/event=0x11,umask=0x01/`
Sampling	50-ms interval; 3-s analysis horizon ($M = 60$ samples)
Protocol	50 windows per configuration; 0.2-s inter-window gap

TABLE III
FIXED-THRESHOLD JSTA WINDOW RESULTS AT $\tau_L = 0.05$.

Condition	n	Alerts	Rate	Wilson 95% ref. interval	s_J^* range
Idle	50	0	0%	[0, 7.13]%	[.0237, .0437]
Victim only	400	0	0%	[0, 0.95]%	[.0201, .0389]
Full-core stress	50	7	14.0%	[6.95, 26.19]%	[.0204, .1576]
Attacker only	400	400	100%	[99.05, 100]%	[.0839, .1367]
Attacker+victim	400	400	100%	[99.05, 100]%	[.4672, .6207]

Attacker+victim ($8 \times 50 = 400$ windows, pinning the victim to Core 4 while varying the attacker across all cores); and (5) *Full-core stress* (50 windows running concurrent memory stress on all eight cores). The victim continuously executed RSA-2048 private-key decryptions via OpenSSL 1.0.0 `libcrypto.so.1.0.0`. The adversary executed the open-source ISEC-TUGRAZ Flush+Reload probe [21]. To establish an extreme benign baseline, the stress workload deployed pinned instances of `stress-ng --vm 1 --vm-bytes 1G --vm-keep` across all CPUs 0–7 (8 GiB footprint). Additionally, we evaluated sustained adversarial interactions under concurrent memory pressure by running continuous 60-s traces with the victim on Core 4 and the attacker sweeping the remaining cores.

B. Metrics and Evaluation Protocol

For attack evaluation, a true positive (TP) occurs when the screening score exceeds threshold τ ($s_J^* > \tau$); in idle, victim-only, or benign workloads, any alert is counted as a false positive (FP). We report condition-level true-positive rates (TPRs) and false-positive rates (FPRs) accompanied by two-sided 95% Wilson score reference intervals. Attacker-core attribution is correct (Top-1) when the predicted core $\hat{k}$ matches the adversary's physical core; the core margin $c_{\hat{k}} - \max_{k \neq \hat{k}} c_k$ measures attribution confidence.

The primary screening gate is $\tau_L = 0.05$: windows above this threshold proceed to candidate-core attribution and process-level confirmation. We also report $\tau_H = 0.20$ as a high-score tier for the observed attacker+victim regime; it does not replace the primary gate or authorize mitigation without confirmation.

C. Detection Efficacy and Threshold Sensitivity

Table III and Fig. 2 present the detection results across all benchmark states. At the screening threshold $\tau_L = 0.05$,

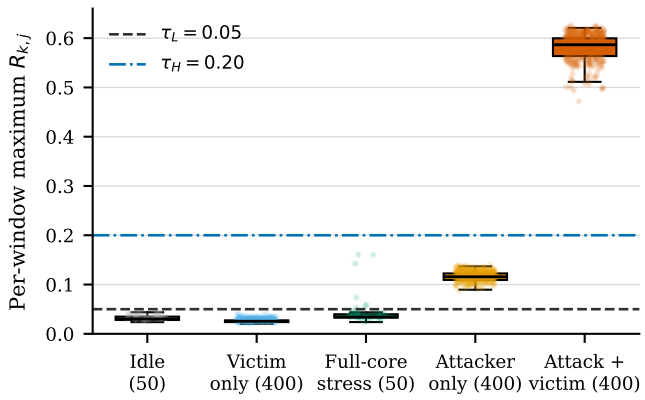

Fig. 2. Distribution of maximum JSTA scores per window across 1,300 benchmark observations. Horizontal dashed lines denote the screening threshold $\tau_L = 0.05$ and high-confidence threshold $\tau_H = 0.20$.

RingTrace achieves 100% detection across both attacker-only and attacker+victim scenarios. Crucially, all 400 victim-only windows remain strictly below τ_L, proving that normal cryptographic victim computation does not trigger false alarms. In the attacker-only scenario, all 400 windows cleanly cross τ_L, demonstrating that RingTrace reliably detects the microarchitectural perturbations of repeated flush operations without requiring active victim participation.

Under full-core 8-GiB memory stress, 43 of 50 windows remain below τ_L (mean 0.03371). Seven transient excursions cross τ_L (14.0% FPR), peaking at 0.15759. Analysis of these excursions reveals that memory stress workers occasionally experience localized scheduling bottlenecks: in runs 21–23, Core 5 monopolized 83.4–94.7% of the total LLC-miss share (versus the 12.5% uniform expectation). This spatial concentration appropriately triggers the share-based screen, passing the event to secondary process confirmation.

The wide gap between the stress maximum (0.15759) and the attacker+victim minimum (0.46725) validates the high-confidence tier: setting $\tau_H = 0.20$ inside this 0.30965 margin completely eliminates stress triggers (0/50, FPR 0%) while preserving 100% (400/400) attacker+victim detections. Sweeping τ reveals clear operating tradeoffs: at $\tau = 0.08$, stress FPR drops to 6% with 100% TPR; above $\tau = 0.137$, attacker-only detection diminishes, confirming that τ_H serves as an accelerated response shortcut for high-amplitude interactions rather than a universal detector for stealthy loops.

D. Core Attribution and Spatial Robustness under Stress

Fig. 3 illustrates the secondary joint evidence maps. Across all 800 controlled attack windows, summing across matrix columns to recover the Core marginal achieves 800/800 Top-1 core attribution accuracy. Attribution relies on the complete row marginal $c_k = \sum_j R_{k,j}$ rather than single-cell maxima, ensuring robustness when uncore CBox traffic disperses across multiple slices due to memory hashing.

To evaluate attribution resilience under adversarial conditions, Fig. 4 shows seven continuous 60-s traces where the F+R

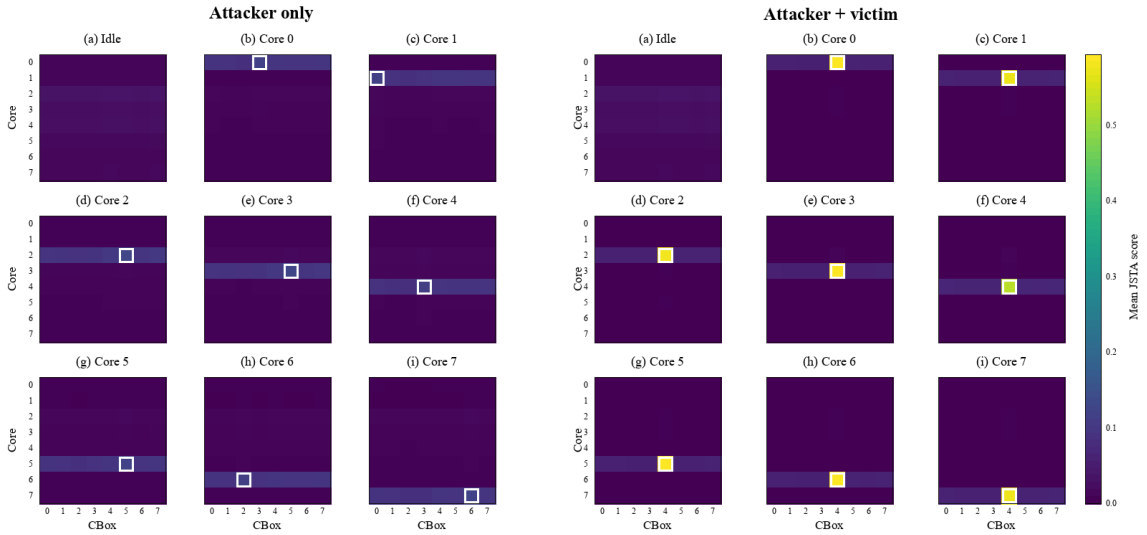

Fig. 3. Mean Core–CBox JSTA matrices for idle and attacker cores 0–7, with and without the victim. All panels share the same linear color scale. White boxes highlight the maximum cell.

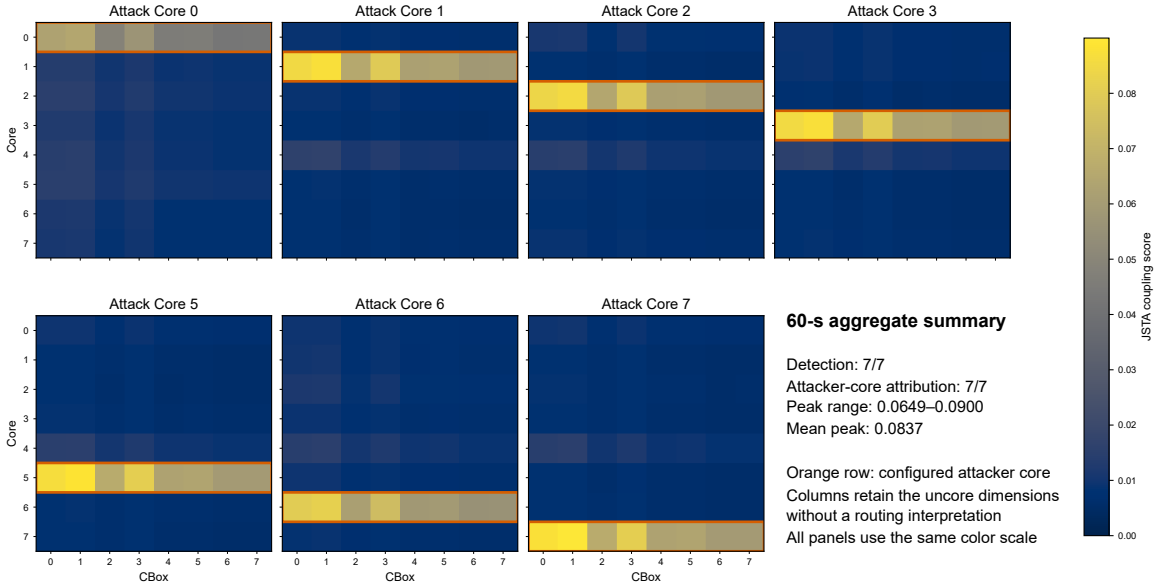

Fig. 4. Mean Core–CBox JSTA matrices from seven 60-s attacker+victim traces under concurrent full-core memory stress. Core 4 hosts the victim and is not an attacker position. Solid orange outlines identify the configured attacker row in each panel; all panels share the same color scale. First-stage screening and row-sum attribution are correct for all windows.

attacker and RSA victim execute concurrently with aggressive 8-core memory stress. Across all 140 non-overlapping 3-s windows, RingTrace maintains 140/140 detection at τ_L and 140/140 Top-1 core attribution. Although concurrent memory pressure dilutes individual cell peaks (mean peak 0.0837, preventing τ_H activation), the configured attacker row consistently commands 41.0–57.2% of the total matrix mass, leading the second-ranked core by a substantial margin of 0.317–0.483. This confirms that RingTrace's spatial signatures remain structurally discernible even under severe uncore interconnect contention.

E. Latency, Process Confirmation, and System Overhead

Backend Processing Latency. Table IV summarizes the timing benchmarks across 50 independently collected traces evaluated twice, yielding 100 timed executions. Processing a complete 3-s trace in batch mode required an average of 25.23 ms (median 21.24 ms, P90 34.65 ms). Amortized across the 60 sampling intervals, this corresponds to approximately 0.42 ms of processing per 50-ms step ($< 0.9\%$ of a single core's compute budget). Crucially, because batch evaluation finishes over $100\times$ faster than the corresponding 3-s collection horizon, the backend sustains continuous streaming analysis

TABLE IV
DETECTION AND FULL-TRACE BATCH INFERENCE FOR CORE-0 F+R TRACES.

Metric	Result
Separately collected attack traces	50
Detection / core attribution	50/50 / 50/50 (100%)
Mean maximum JSTA score	0.1126 ± 0.0074
Inference timed executions	100
Mean 3-s trace batch latency (bootstrap 95% CI)	25.23 ms [23.36, 27.31]
Median batch latency	21.24 ms
P90 / P95 batch latency	34.65 / 51.59 ms
Maximum batch latency	62.27 ms
Amortized per-interval processing time	≈ 0.42 ms

TABLE V
APPLICATION RUNTIME OVERHEAD ON SPEC CPU2017.

Benchmark	Runtime overhead	SPEC ratio loss
perlbench	4.720%	4.508%
gcc	4.361%	4.175%
mcf	1.667%	1.639%
x264	0.563%	0.560%
xz	2.712%	2.640%
Equal-weight mean	2.805%	2.704%

with substantial real-time throughput and zero queueing backlogs.

Process-Level Confirmation. To validate the second-stage response pipeline, screening triggers activate a 0.5-s `perf record` profiling burst pinned to the candidate core. Inspecting command and process identifiers (`comm`, `pid`) revealed sharp discrimination: when the adversary (`spy`) executed on Core 2, process-level aggregation attributed 99.88% of recorded LLC misses to `spy`; aggregated by binary command, `spy` accounted for 99.93% of miss events, while the most active benign background process accounted for only 0.05%. This confirms that false screening triggers induced by transient memory stress can be dismissed in sub-second time without penalizing benign workloads.

Application-Level Performance Overhead. Table V evaluates the runtime impact of RingTrace on five representative SPEC CPU2017 integer benchmarks spanning compute-intensive (`perlbench`, `gcc`), memory-latency-bound (`mcf`), multimedia streaming (`x264`), and compression workloads (`xz`). Across ten repetitions with and without monitoring, the equal-weight mean runtime overhead was 2.805% (runtime-weighted mean: 2.793%). Every benchmark exhibited less than 5% slowdown (peaking at 4.720% for `perlbench`), demonstrating that concurrent Core and CBox PMU telemetry is commercially viable for multi-tenant deployment.

V. DISCUSSION

A. Architectural Defense Value and Operational Deployment

RingTrace enforces a constructive hierarchy separating three defense roles: the JSTA peak screens joint activity, its analytical Core marginal pinpoints the culprit core, and targeted profiling confirms event ownership before mitigation. Retaining local core provenance alongside distributed uncore footprints throughout the decision path provides an architectural defense rather than passive monitoring. This joint representation prevents false accusations from asymmetric single-core bursts while maintaining high sensitivity to stealthy, victim-absent exfiltration loops.

Operationally, this tiered design informs practical response policies while suppressing alert fatigue. Scores in $(\tau_L, \tau_H]$ flag ambiguous events, triggering a 0.5-s `perf record` burst on the candidate core to isolate command and process identifiers (`comm`, `pid`), resolving transient benign bursts in sub-second time without disrupting workloads. High-amplitude signals exceeding τ_H enable expedited escalation in low-noise regimes. Production deployment should establish labeled baselines across diverse benign phases; if a suspect thread migrates or alters its PID, the monitor resets inspection rather than inheriting prior state.

Crucially, while evaluated on bare-metal Linux, this attribution paradigm translates directly to multi-tenant clouds. Modern virtualization platforms routinely enforce dedicated vCPU-to-pCPU pinning (e.g., via `libvirt` or `cpuset` cgroups) to curb noisy-neighbor interference. By pinpointing the physical core, the host hypervisor can immediately map the flagged pCPU to the offending VM or container without guest-side agents. Host-level tools (e.g., `perf kvm`) then verify the suspect vCPU thread, enabling targeted mitigation—such as vCPU throttling, core re-pinning, or migration—while shielding co-located benign tenants.

B. Limitations, Scope, and Evasion Resilience

Scope and Microarchitectural Channels. The empirical focus of this work is deliberately centered on the Flush+Reload receiver mechanism. Because modern transient-execution attacks (including Spectre, Meltdown, and Foreshadow [4]–[6]) universally employ flush-and-reload loops to extract microarchitectural secrets, monitoring this recurring exfiltration primitive provides a decisive defense boundary prior to data leakage. However, this focus does not claim coverage over non-flush side channels, such as Prime+Probe, Flush+Flush, or execution-port contention. Furthermore, while the current prototype is evaluated on a commodity 8-core ring architecture, CBox counter semantics and physical topologies vary across processor generations, necessitating platform-specific recalibration when porting across microarchitectures.

Resilience Against Adversarial Evasion. Recent studies, such as Vizard [20], demonstrate that sophisticated adversaries can evade core-level HPC detectors by injecting counterbalancing instructions or cache hits to normalize anomalous event ratios. RingTrace significantly raises the bar against such evasion: while an attacker can easily manipulate core-private counters locally, shaping a realistic Core–CBox joint footprint requires coordinating accesses across distributed uncore slices according to the processor's complex, non-linear address-to-slice hashing algorithms [14]. Even if an advanced adversary leverages reverse-engineered slice mappings or huge-page alignment to steer memory transactions, simultaneously counterbalancing both core misses and slice-specific uncore

traffic introduces non-trivial execution latency and bus contention. This operational burden inevitably throttles the flush-and-reload loop and slashes effective exfiltration bandwidth, directly undermining the attacker's primary objective. Formally quantifying the performance-stealth tradeoff for adaptive adversaries with partial slice control remains a compelling direction for future work.

VI. Conclusion

In this paper, we addressed the lack of spatial context in conventional hardware-counter detectors and the difficulty of isolating stealthy cache side channels under complex workloads. By coupling core-local cache misses with distributed uncore CBox telemetry, we introduced RingTrace, a system-level monitoring framework centered on Joint Spatio-Temporal Attribution. Through this model-free matrix formulation, we established a hierarchical defense pipeline that effectively screens for anomalous spatio-temporal activity and pinpoints the culprit core. Our approach reliably detects victim-independent flush activity prior to secret exfiltration, maintains robust spatial attribution under severe concurrent memory pressure, and achieves rapid process confirmation with negligible application runtime overhead.

Future research should explore extending this cross-domain telemetry model beyond commodity ring architectures to more complex processor topologies, such as server-grade mesh interconnects and multi-socket platforms, to adapt to diverse cloud deployment scenarios. Additionally, investigating adaptive evasion strategies particularly adversaries attempting coordinated uncore manipulation and integrating automated hardware-software mitigation mechanisms could further enhance system resilience. Addressing the remaining challenges in microarchitectural side-channel defense requires continued research and innovation.

References

[1] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[2] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Topics in Cryptology – CT-RSA 2006*, ser. Lecture Notes in Computer Science, vol. 3860. Springer, 2006, pp. 1–20.

[3] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, ser. Lecture Notes in Computer Science, vol. 9721. Springer, 2016, pp. 279–299.

[4] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.

[5] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, pp. 973–990. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[6] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, pp. 991–1008. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[7] T. Zhang, Y. Zhang, and R. B. Lee, "CloudRadar: A real-time side-channel attack detection system in clouds," in *19th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, ser. Lecture Notes in Computer Science, vol. 9854. Springer, 2016, pp. 118–140.

[8] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "NIGHTs-WATCH: A cache-based side-channel intrusion detector using hardware performance counters," in *7th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*. ACM, 2018, pp. 1–8.

[9] M. Song, J. Lee, T. Suh, and G. Koo, "RT-Sniper: A low-overhead defense mechanism pinpointing cache side-channel attacks," *Electronics*, vol. 10, no. 22, p. 2748, 2021.

[10] S. Carnà, S. Ferracci, F. Quaglia, and A. Pellegrini, "Fight hardware with hardware: System-wide detection and mitigation of side-channel attacks using performance counters," *ACM Digital Threats: Research and Practice*, vol. 4, no. 1, pp. 1–24, 2023.

[11] W. Kosasih, Y. Feng, C. Chuengsatiansup, Y. Yarom, and Z. Zhu, "SoK: Can we really detect cache side-channel attacks by monitoring performance counters?" in *19th ACM Asia Conference on Computer and Communications Security (AsiaCCS)*. ACM, 2024, pp. 172–185.

[12] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring(s): Side channel attacks on the CPU on-chip ring interconnect are practical," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 645–662. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/paccagnella

[13] X. Bian and H. Cho, "Detecting cache-based side-channel attacks by leveraging mesh interconnect traffic monitoring," in *40th ACM/SIGAPP Symposium on Applied Computing (SAC)*. ACM, 2025, pp. 1829–1838.

[14] C. Maurice, N. L. Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse engineering intel last-level cache complex addressing using performance counters," in *Research in Attacks, Intrusions, and Defenses (RAID)*, ser. Lecture Notes in Computer Science, vol. 9404. Springer, 2015, pp. 48–65.

[15] L. Wang, L. Bu, and F. Song, "SCAGuard: Detection and classification of cache side-channel attacks via attack behavior modeling and similarity comparison," in *60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[16] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. J. Stolfo, "On the feasibility of online malware detection with performance counters," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2013, pp. 559–570.

[17] M. Chiappetta, E. Savaş, and C. Yılmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016.

[18] S. Briongos, P. Malagón, G. Irazoqui, and T. Eisenbarth, "CacheShield: Detecting cache attacks through self-observation," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 2018, pp. 224–235.

[19] M. Dai, R. Paccagnella, M. Gomez-Garcia, J. McCalpin, and M. Yan, "Don't mesh around: Side-Channel attacks and mitigations on mesh interconnects," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2857–2874. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/dai

[20] M. Song, T. Suh, and G. Koo, "Vizard: Passing over profiling-based detection by manipulating performance counters," *IEEE Access*, vol. 11, pp. 48 099–48 112, 2023.

[21] ISEC, Graz University of Technology, "flush_flush: Open-source cache-attack examples," includes a Flush+Reload implementation; accessed September 2026. [Online]. Available: https://github.com/isec-tugraz/flush_flush/tree/master