computers

MDPI

Article

ARCHER: A Cycle-Accurate RISC-V Emulator for Microarchitectural Side-Channel and Memory Encryption Research

Jyotiprakash Mishra [1,2,*], Sanjay K. Sahay [1,*], Swati Mishra [3] and Aman Pathak [4]

1 Department of Computer Science and Information Systems, Birla Institute of Technology and Science (BITS) Pilani, K. K. Birla Goa Campus, Zuarinagar 403726, Goa, India
2 School of Computer Engineering, Kalinga Institute of Industrial Technology (KIIT) Deemed to be University, Bhubaneswar 751024, Odisha, India
3 Independent Researcher, Cuttack 754001, Odisha, India; swatimishra0525@gmail.com
4 ENineHQ Technologies, Cuttack 754001, Odisha, India; aman@eninehq.com
* Correspondence: p20210041@goa.bits-pilani.ac.in (J.M.); ssahay@goa.bits-pilani.ac.in (S.K.S.)

Abstract

Microarchitectural side-channels leak data through caches, branch predictors, store buffers, and speculative execution. Evaluating defenses at cycle-model fidelity has forced a choice between functional tools (Spike, QEMU) and gem5's hours-long Linux boots. ARCHER is a cycle-model-relative RV64IMAFDC RISC-V emulator with pluggable superscalar out-of-order execution, coherent caches, simultaneous multithreading, 1 to 16 harts, and thirteen speculation policies: an unrestricted baseline plus five classical defenses and seven new defenses spanning issue-gate, predictive-redirect, and selective-cleanup mechanism families. It boots Linux 6.6 in under three minutes, runs 3.7× faster (geometric mean) than gem5's DerivO3CPU on head-to-head bare-metal microbench, matches gem5 within ±20% on 4 of 10 workloads, matches Chipyard's fab-ready RTL to a 9.4% mean cycle-count deviation under two fitted match-configurations, and produces bit-identical architectural output to QEMU at a median 88× host-throughput advantage over Chipyard's Verilator flow. Every published Spectre-family defense drives leakage to zero at sub-0.3% instructions-per-cycle (IPC) loss on a full Linux boot, and each new policy is validated on the attack surface its mechanism defends. The Adaptive Memory Encryption Scheme (AMES) adds a bus-layer engine with four authenticated ciphers (each validated bit-exact against its published specification) and per-leaf mask-XOR for Differential Power Analysis (DPA) hardening; a first-order Correlation Power Analysis consistency check confirms the direction of the theoretical bound under the shipped leakage model. Together, ARCHER and AMES provide a single INI-configurable environment for cycle-model side-channel evaluation in minutes rather than hours.

Keywords: RISC-V; cycle-accurate emulation; side-channel attacks; Spectre; Meltdown; memory encryption; microarchitectural security; cache timing; differential power analysis; Chipyard; Rocket; BOOM

Check for updates

Academic Editor: Pedro Alonso Jordá

Received: 15 June 2026
Revised: 22 July 2026
Accepted: 23 July 2026
Published: 28 July 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Microarchitectural side-channel attacks have altered the security model of modern processors. Spectre [1] demonstrated that speculative execution, a performance optimization present in virtually every high-performance CPU since the mid-1990s, can be weaponized

to leak arbitrary memory across privilege boundaries. Meltdown [2] showed that transient out-of-order loads can read kernel memory from user space before the faulting access is architecturally retired. Microarchitectural Data Sampling (MDS) [3,4] subsequently revealed that internal buffers (store buffers, fill buffers, and load ports) serve as covert channels even when the cache hierarchy is properly partitioned.

These vulnerabilities are not corner cases: they arise from the very mechanisms that deliver expected performance gains and affect processors from every major vendor. Their severity is compounded by the cost of remediation: software patches (e.g., retpoline, Kernel Page Table Isolation (KPTI)) impose measurable performance penalties, while hardware fixes require silicon respins with multi-year lead times.

Evaluating proposed defenses against these attacks requires simulation environments that model microarchitectural structures with cycle-level fidelity. Cache timing attacks demand a cache hierarchy that faithfully simulates set associativity, replacement policies, and coherence state. Spectre variants require a branch predictor that trains, mispredicts, and squashes correctly. MDS attacks need a store buffer that accurately models transient stale-data forwarding. The requirement, in short, is not merely functional correctness but timing-observable correctness: the simulator must expose the same microarchitectural side effects that real hardware leaks. Figure 1 previews the resulting emulator's shape (pluggable cores, TLBs, caches, buffers, and an optional bus-layer encryption engine, every one runtime-configurable) before we contrast it with prior tools.

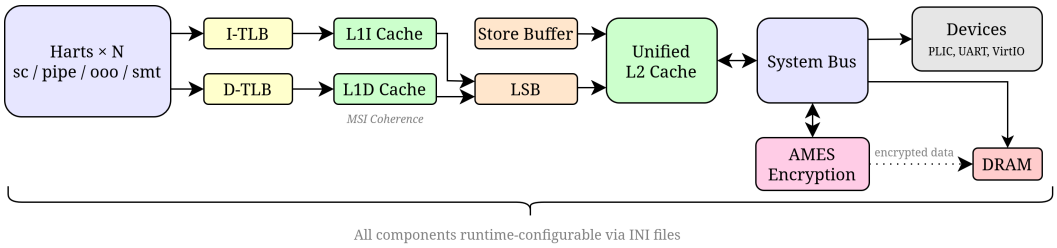

Figure 1. High-level architecture of ARCHER. Up to 16 harts (scalar, pipelined, out-of-order, or simultaneous multithreading (SMT)) issue memory requests through split instruction/data translation lookaside buffers (TLBs) into a modified–shared–invalid (MSI)-coherent L1I/L1D cache hierarchy. A store buffer and line fill buffer (LFB) mediate traffic to the unified L2, which communicates with DRAM and memory-mapped devices via the system bus. The optional AMES encryption engine intercepts bus transactions between the L2 cache and DRAM: data are encrypted on writeback and decrypted on cache fill; so, only ciphertext resides in DRAM, while plaintext flows through the cache hierarchy. Every shaded component is independently configurable at runtime through INI parameter files.

Existing RISC-V simulators occupy distinct but incomplete points in this design space. Spike [5] provides a fast functional model but lacks a microarchitectural state: no caches, pipeline, or branch predictor. QEMU [6] achieves near-native speed via dynamic binary translation but elides cache sets, speculative execution, and store-buffer forwarding. gem5 [7] offers detailed cycle-accurate out-of-order (OoO) modeling; however, its O3CPU mode runs at ∼200 KIPS (Related Work, Section 2), requires hours to boot Linux, and its 300,000+ line C++ codebase imposes a steep learning curve for adding new microarchitectural features. No readily available tool simultaneously provides microarchitectural fidelity sufficient for side-channel research, the ability to boot a full operating system (OS) in minutes, and the simplicity to integrate novel defenses rapidly.

This paper presents ARCHER, a cycle-accurate RV64IMAFDC emulator that sits between functional instruction-set-architecture (ISA) simulators and gem5-class heavyweight microarchitectural simulators. Figure 1 shows the high-level organization: four pluggable core types (scalar, in-order pipeline, superscalar out-of-order, and SMT), an MSI-coherent cache hierarchy, pluggable branch predictors, memory disambiguation policies, and a

complete RISC-V privilege architecture (M/S/U modes, Sv39, physical memory protection (PMP)). Every microarchitectural parameter is configurable at runtime through INI-format files (Section 7), enabling design-space exploration without recompilation. Despite this fidelity, ARCHER boots Linux 6.6 with OpenSBI (an open implementation of the RISC-V Supervisor Binary Interface (SBI)) in under three minutes on commodity hardware. A head-to-head speedup measurement is available on a bare-metal microbench, where ARCHER runs 3.7× faster (geometric mean) than gem5's DerivO3CPU on the same host (Section 4.6, Table 8). Full-system Linux-boot head-to-head is impractical, because gem5's DerivO3CPU Linux boots are reported at multiple hours in the community literature [8,9] and the gem5 documentation itself cites this cost as the motivation for its checkpoint-restore workflow; ARCHER's three-minute boot therefore sits roughly two orders of magnitude below that documented range. Cycle counts in the paper are validated as *cycle-model-relative*: they match published gem5 and Chipyard RTL references within tight tolerances rather than tracking silicon on the same die, and Section 4 lays out the ISA/boot/event/cycle validation layers separately. Throughout the paper we use "cycle-accurate" in the sense common in the simulator literature (gem5, Chipyard). Section 4.5 distinguishes the four validation layers explicitly (ISA, boot, event, cycle) and marks the cycle-level layer as unfitted against gem5's O3CPU (Section 4.6) and fitted against Chipyard's RTL via two match configurations (Section 4.7).

The emulator's cache, buffer, and speculation subsystems capture the cache-, buffer-, and speculation-based side-channel surface (Spectre, Meltdown, MDS, prefetch, and cross-thread channels). The AMES (Adaptive Memory Encryption Scheme) memory-encryption engine extends this coverage to the two remaining surfaces relevant to a memory-encryption attacker: physical-memory attacks on DRAM (cold-boot, bus snooping, direct memory access (DMA) injection, replay) and power-side-channel attacks that recover secrets via differential power analysis (DPA) of on-chip cryptographic primitives. AMES addresses the first with bus-layer Authenticated Encryption with Associated Data (AEAD) and the second with static per-leaf mask-XOR pre-encryption for mask-based hardening.

AMES supports four authenticated ciphers (AES-128 with Galois/Counter Mode (GCM), ASCON-128, PRINCE-64 with a Parallelizable Message Authentication Code (PMAC1) wrapper, and QARMA-64 with the same PMAC1 wrapper) plus a non-cryptographic XORSHIFT baseline, a hierarchical key tree, configurable integrity-tree arities, and per-leaf mask XOR, all parameterized through INI files. Modeling encryption latency at the cycle level lets researchers quantify performance and security trade-offs for memory-encryption and mask-hardening schemes without waiting for silicon.

We validate the emulator through end-to-end reproductions of Spectre [1], Meltdown [2], and MDS [3,4] attacks on Linux, together with five published defenses spanning both the speculation and memory-disambiguation axes: STT [10] (whose taint-and-block mechanism subsumes Non-speculative Data Access (NDA) [11] design), InvisiSpec [12], CleanupSpec [13], and Fence on the speculation axis; SSBD via ARCHER's `disambig=conservative` memory-disambiguation policy on the disambiguation axis. Seven new speculation-defense policies (AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC) are introduced in Section 5.10. Quantitative counter-based evidence of leakage and suppression is reported in the speculation and buffer sections of Table 49.

The contributions of this paper are as follows:

1. A **cycle-accurate RV64IMAFDC emulator** with four pluggable core types, MSI-coherent caches, and full privilege architecture, all runtime-configurable via INI files (Figure 1, Table 3).
2. **End-to-end reproduction** of ten microarchitectural attacks (Spectre v1/v2/return-stack-buffer (RSB)/branch-history-buffer (BHB), Meltdown-US, L1TF, Fallout, RIDL,

prefetch, and SMT side channels) across fifteen defense configurations and five published defenses spanning both the speculation and memory-disambiguation axes: InvisiSpec, STT (whose taint-and-block mechanism subsumes NDA's design), CleanupSpec, and Fence on the speculation axis; SSBD via ARCHER's `disambig = conservative` policy on the disambiguation axis. These are validated on Linux 6.6 with statistical evidence of leakage and suppression (Table 24, Figure 17).

3. A **systematic attack-vs-defense matrix** covering the Spectre family, cache-timing attacks, and cross-hart channels: the seven classical/overlay defenses drive leakage from 100% to 0% with sub-0.3% instructions-per-cycle (IPC) loss on a full Linux boot (three of five published defenses land at $\leq$0.03%) (Section 5.9; Table 25).
4. **Seven new speculation-defense policies** (AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC) implemented as `spec_policy_ops_t` vtable entries across three mechanism families: issue-time gate (AKTO, MDPO, ROPC), predictive redirect (DLRF, PSF), and selective cleanup (CRPSO, LFBCO). On the classical Spectre matrix (Table 25) three policies close all five variants (AKTO, CRPSO, LFBCO), one closes four (PSF), and one closes two (DLRF, the variants whose leaky-load address flows through a speculative feeder). ROPC closes the standard Kocher-2019 Spectre-RSB workload via the new `ropc_early_squash` extension in the OoO backend; MDPO closes the Osvik–Shamir–Tromer AES T-table cache-timing pattern [14] as shipped in ARCHER's attack corpus, via the new `on_domain_change` vtable slot and L1D cross-domain scan. Two purpose-built cross-domain workloads augment (not replace) the published-attack demonstrations for MDPO and ROPC (Table 28). The design summary is in Table 29, the mechanism-hook diagram in Figure 21, and the per-policy scope in Section 5.10.
5. **Cross-implementation calibration verification** against Chipyard's fab-ready Verilog register-transfer-level (RTL) design for the in-order Rocket and 3-wide out-of-order BOOM cores. Under the two match-configuration profiles `rocket_match.cfg` and `boom_match.cfg`, whose cache latencies, load-use latency, and mispredict penalty are fitted to Chipyard's shipped Rocket and MediumBoomV3 configurations (Table 10 marks the fitted parameters with †), ARCHER's cycle model matches Chipyard's Verilator-simulated RTL to a mean 9.4% (median 9.1%) deviation across a 20-cell workload $\times$ core matrix, at a median 88$\times$ host-throughput advantage. This demonstrates that ARCHER's timing knobs are expressive enough to reproduce Chipyard's RTL cycle model within tight tolerance; this is a calibration verification, not an independent cross-implementation validation. Independently, the same bare-metal binary produces bimodal Flush+Reload cycle histograms on both ARCHER and Chipyard RTL via the standard `rdcycle` timer, meeting the "distributions essentially identical" bar (Section 4.7; Tables 11 and 30).
6. **Family-level fidelity** across the entire fab-ready Chipyard RTL family (RocketConfig, SmallBoomV3Config, MediumBoomV3Config, LargeBoomV3Config): every cell in a 4-config $\times$ 2-workload Flush+Reload/Evict+Reload sweep is bimodal, HIT peaks agree to within one cycle on every cell, MISS peaks to within two cycles on six of eight, and the sweep runs orders of magnitude faster than Chipyard's Verilator on the same Executable and Linkable Format (ELF) binaries (Section 5.12; Table 33, Figure 24).
7. **AMES memory-encryption engine**: a bus-layer engine that extends ARCHER's side-channel coverage from cache and microarchitectural channels to physical-memory attacks (four tweakable authenticated ciphers, a hierarchical key tree, configurable integrity-tree arities) and to power-side-channel hardening (static per-leaf mask XOR of the plaintext before encryption), storing ciphertext in emulated DRAM with modeled cycle-level latency; every authenticated cipher (AES-128-GCM, ASCON-128,

PRINCE-64, QARMA-64) is validated by an in-tree Known-Answer Test (KAT) suite (Table 38, Figure 28, Section 6.4).

8. **Cryptographic-subsystem side-channel demonstrations** that put ARCHER's own AES-128 under the same threat model as the rest of the paper: (a) an access-pattern soundness analysis on the T-table AES variant recovers the top nibble of every key byte at $N = 128$ traces (reducing the AES-128 key space from 2^{128} to 2^{64}), and switching to the S-box-preload or no-memory-lookup variant removes the observable channel (Section 5.13); (b) a Correlation Power Analysis (CPA) consistency check under a Hamming-weight leakage model shows the shipped mask-XOR scheme moving first-order CPA from full-key recovery at $N = 1000$ traces (unmasked baseline) to no recovery at $N = 100{,}000$ traces with any $d \geq 1$ masking order (Section 6.5).
9. A **validation suite** covering ISA correctness, trap and privilege behavior, full-system Linux boot, and per-component microarchitectural events across ten execution modes and 28 Linux boot configurations (Tables 5, 6 and 49), with host-throughput comparisons in Tables 2 and 12 that make the iterative-workflow claim above concrete.

Scope of novelty. Because side-channel research is inherently cumulative, it is important to be explicit about which parts of this paper are novel contributions and which are faithful reproductions of prior work under a new tool. Table 1 summarizes this separation. The attacks and defenses themselves (Spectre, Meltdown, MDS, STT, InvisiSpec, CleanupSpec, SSBD, KPTI) are all well-established in the literature; ARCHER's role is to make them reproducible from a single binary and unified INI configuration surface, whereas prior evaluations required distinct simulator forks or hardware setups. The novel contributions of this paper are the ARCHER emulator itself (its INI-configurable pluggable-vtable architecture, its heterogeneous core-type mixing, and a cycle-modeled runtime that boots Linux in under three minutes); the AMES memory-encryption engine (its hierarchical key tree, per-leaf mask XOR pre-encryption, closed-form re-keying and tree-arity expressions of Section 6, and an empirical characterization of the splay-arity knob showing it is dominated by the on-chip key cache); the unified attack–defense evaluation framework of Section 5; the seven new speculation-defense policies (AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC) covering three mechanism families, together with the mechanism-scoped coverage matrix of Table 25; and the pre-built configuration files that operationalize each attack, defense, and AMES parameter-shape sweep point as a single command-line invocation. The complete reproducibility artifact (sources, configurations, workloads, and per-table reproduction scripts) is publicly archived on Zenodo [15].

The remainder of this paper proceeds as follows. Section 2 surveys architectural simulators, side-channel tools, and memory encryption schemes. Sections 3 and 4 describe the emulator's design and validation. Sections 5 and 6 detail attack reproductions, defense evaluations, and the AMES encryption engine. Finally, Sections 7–10 discuss the configuration space, statistics infrastructure, limitations, and conclusions.

Table 1. Scope of contributions. Left column lists this paper's novel artifacts (ARCHER emulator, AMES memory-encryption engine, seven new speculation-defense policies, unified attack–defense evaluation matrix, and cryptographic-subsystem side-channel demonstrations). Right column lists prior work (attacks, defenses, cipher specifications) that ARCHER reproduces under a single INI configuration surface.

Contribution of This Paper (Novel)	Reproduced Under ARCHER (Prior Art)
The ARCHER emulator: INI-configurable pluggable vtables, heterogeneous core mixing, cycle-modeled Linux boot in <3 min.	Attacks: Spectre v1/v2/RSB/BHB [1], Meltdown-US, L1TF [2,16], Fallout, RIDL [4,17], prefetch [18], SMT-port [19].
AMES: bus-layer AEAD encryption engine, splay-tree key management, per-leaf mask XOR pre-encryption, closed-form rekey/arity expressions.	Defenses: STT [10], InvisiSpec [12], CleanupSpec [13], NDA [11], SSBD, KPTI [2].
Unified attack–defense evaluation matrix (deterministic control and status register probes, 120 pre-built configs; see Section 7).	Baseline microarchitectural structures modeled from published sources (MSI coherence, TAGE, gshare, TLB partitioning, VERW, LFENCE).
Seven novel speculation-defense policies (AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC) implemented as `spec_policy_ops_t` vtable entries, covering three mechanism families (issue-time gate, predictive redirect, selective cleanup) and each closing the mechanism-scoped subset of the Spectre-family matrix it targets (Table 29).	
Cross-simulator throughput comparison and quantitative side-channel counter methodology (Section 4). Cryptographic-subsystem side-channel demonstrations that turn the same T-table and Correlation Power Analysis methodologies on ARCHER's own AES-128 (Sections 5.13 and 6.5); in-tree KAT harness aggregating published test vectors across four ciphers (Section 6.4).	Cipher *specifications* (AES-128, ASCON-128, PRINCE-64, QARMA-64) from the NIST and IACR references enumerated in Section 6.4; ARCHER's implementations are our own code, validated against those references.

2. Related Work

This section surveys architectural simulation platforms, side-channel research tooling, and memory encryption schemes; Table 2 provides a quantitative comparison. Prior tools address the space unevenly along three axes: speed versus microarchitectural detail (Spike and QEMU trade one for the other, gem5 is the reverse), composability of side-channel defenses (each is a separate gem5 fork of 2000–5000 lines of custom C++), and configurability of memory encryption (Intel TME and AMD SEV are fixed silicon; academic proposals live in unpublished forks). Each subsection below closes with how ARCHER addresses the corresponding gap.

2.1. Architectural Simulators

RISC-V simulators span a wide fidelity–performance spectrum. **Spike** [5], the official ISA reference simulator, executes approximately 100 MIPS and serves as the golden reference for compliance testing, but it lacks any microarchitectural detail: no cache hierarchy, branch predictor, speculative execution model, or pipeline simulation. **QEMU** [6] achieves near-native speed via dynamic binary translation and boots Linux in seconds but inherently elides all microarchitectural states, precluding any side-channel study.

gem5 [7] provides event-driven simulation with full Modified-Owned-Exclusive-Shared-Invalid (MOESI) coherence and configurable out-of-order pipelines and has been used to evaluate InvisiSpec [12], STT [10], CleanupSpec [13], and NDA [11]. However, gem5 executes at approximately 200 KIPS, requires hours to boot in Linux, comprises 300,000+ lines of C++ with a steep learning curve, and lacks a unified side-channel framework: individual defenses exist only in separate forks. **Simics** [20] (now Intel Wind River) provides full-system simulation with transaction-level timing overlays but is proprietary, offers limited RISC-V support, and its closed-source internals preclude deep microarchitectural modification.

Table 2. Comparison of RISC-V simulation platforms for microarchitectural security research. Rows: modeled features and toolchain characteristics; columns: ARCHER, gem5, Spike, and QEMU.

Feature	ARCHER	gem5 [7]	Spike [5]	QEMU [6]
ISA fidelity (RV64IMAFDC)	Full	Partial [a]	Full	Full
Microarchitectural model	Detailed	Detailed	None	None
OoO core	Yes (N-wide)	Yes (O3)	No	No
SMT support	Yes (1–16 harts)	Limited	No	No
Cache coherence protocol	MSI	MOESI	N/A	N/A
Speculation defense policies	13 [f]	1–2 [b]	N/A	N/A
Store buffer/LFB model	Yes/Yes	Yes/Yes	No/No	No/No
Memory encryption engine	AMES (5 ciphers)	No [c]	No	No
Linux boot time	<5 min	Hours [d]	<1 s	<5 s
Statistics counters	182+	100 s	0	0
Configuration mechanism	INI files [e]	Python 3 scripts	CLI flags	CLI flags
Codebase size (approx.)	25K lines C	300K+ lines C++	20K lines C++	200K+ lines C

[a] gem5's RISC-V support is less mature than x86/ARM; some privilege-mode features and devices require patches. [b] Individual defenses (e.g., InvisiSpec, STT) exist in separate gem5 forks; no unified framework is available. [c] Custom memory encryption models have appeared in unpublished gem5 forks but are not part of the mainline distribution. [d] Booting a minimal Linux image in gem5's O3CPU mode typically requires 2–8 h depending on host hardware. [e] ARCHER ships 120 pre-built configurations (91 top-level plus 29 AMES-sweep) spanning all attack–defense combinations; see Section 7. [f] Thirteen speculation policies: one unrestricted baseline, five classical defenses (InvisiSpec, STT, CleanupSpec, Delay, Fence), and seven new defenses (AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC); see Section 5.10 and Table 29.

Given these limitations in speed, simplicity, and unified side-channel support, **ARCHER** occupies a deliberate middle ground: it provides a superscalar OoO core, MSI coherence, store buffer and LFB modeling, SMT, and thirteen pluggable speculation policies (unrestricted baseline plus five classical defenses: InvisiSpec, STT, CleanupSpec, Delay, Fence; plus seven new defenses: AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC; see Section 5.10) at approximately 2 MIPS in timed mode, with every microarchitectural parameter runtime-configurable via INI files.

2.2. Side-Channel Attack Research Tools

Existing side-channel tools fall into three categories. **Measurement frameworks** such as Mastik [21] implement Flush+Reload, Prime+Probe, and related cache-timing techniques on real x86 hardware. While effective for characterizing attacks on physical machines, they cannot evaluate proposed hardware defenses and provide no visibility into speculative execution paths or store buffer contents. By contrast, ARCHER exposes a runtime-registered counter interface covering every modeled structure (Section 8).

Formal approaches such as CheckMate [22] automatically synthesize exploits from μspec axioms and can prove the absence of certain vulnerability classes, but they operate on abstract models that cannot run real software or capture quantitative timing. **Simulation-based evaluations** have been conducted primarily in gem5, where each defense (InvisiSpec, STT, CleanupSpec, NDA) required 2000–5000 lines of C++ modifications to gem5's coupled O3 CPU model and evaluated only a narrow slice of the attack surface.

This fragmentation creates a critical research gap: no single platform currently reproduces attacks spanning Spectre, Meltdown, MDS, prefetch, and SMT channels alongside their defenses under identical conditions. ARCHER provides thirteen speculation policies (an unrestricted baseline plus five classical defenses plus seven new defenses; see Section 5.10 and Table 29) and four disambiguation policies, all switchable via a single INI entry (Table A2).

2.3. Memory Encryption

Hardware memory encryption defends against physical, cold-boot, and bus-snooping attacks. **Intel TME/MKTME** [23] provides AES-XTS total memory encryption with up to

64 per-VM keys, and **AMD SME/SEV** [24] offers per-VM encryption to isolate guests from a compromised hypervisor. Both are fixed in silicon with no flexibility in cipher choice, tree structure, or integrity mechanism. Gueron [25] proposed the MEAS architecture using Merkle tree integrity verification with tweakable block ciphers, establishing the theoretical foundation for tree-based memory authentication but providing no public simulation platform for design-space exploration.

A limitation shared by all existing work is the absence of a configurable simulator, where researchers can vary the tree arity, cipher type, key-cache size, and masking order while measuring the cycle-level impact on real workloads. Commercial implementations are fixed in silicon, and academic proposals rely on custom analytical models or unpublished gem5 extensions, preventing systematic exploration of the design space. ARCHER's AMES memory-encryption engine provides four pluggable AEAD ciphers plus a non-cryptographic baseline, a hierarchical key tree with configurable arity, $d + 1$ share masking, and ciphertext stored in DRAM, all configurable through the INI parameter system (Section 6; Table 38). Having positioned ARCHER against prior simulators, side-channel tools, and memory-encryption designs, we now detail the emulator itself in Section 3.

3. Emulator Architecture

ARCHER supports four execution models of increasing complexity: single-cycle, pipelined, out-of-order superscalar, and simultaneous multithreading. All models share a common ISA decode layer and pluggable memory hierarchy, and every microarchitectural component uses a vtable interface that allows replacement policies, branch predictors, speculation defense policies, prefetchers, and disambiguation policies to be selected at runtime through an INI configuration file. Every knob referenced in the rest of this section (`[section] key = value` in an INI file, INI keys quoted in monospace as they appear in the shipped configs, and a small block of custom control and status registers (CSRs) 0x801–0x808 used to inspect and control microarchitectural state deterministically) is documented in place; the full CSR block appears in Section 5.1, where its side-channel-research purpose becomes concrete.

3.1. ISA Coverage

ARCHER implements the RV64IMAFDC base ISA plus the Zicsr, Zifencei, Zicntr, Zihpm, and four bitwise (Zba, Zbb, Zbc, Zbs) extensions, totaling 164 instructions across 12 extension sets. All instructions are decoded into a universal intermediate representation (`decoded_inst_t`) shared by every execution model. Three privilege modes (M/S/U) [26] are supported with 68 CSRs. The memory system includes Sv39/Sv48/Sv57 virtual addressing with superpages, a hardware page table walker that maintains accessed/dirty bits and enforces SUM/MXR semantics, and a 16-entry Physical Memory Protection (PMP) module supporting TOR, NA4, and NAPOT access modes. Custom CSRs provide cache control (0x801), security domain management (0x802), and deterministic side-channel probes (0x803–0x805; see Section 5.1 for the probe semantics).

3.2. Execution Models

Four execution models are selectable per-hart via `-t`, enabling heterogeneous configurations (e.g., `-t pipe,ooo:4,smt`).

Single-cycle (`sc`). The simplest model executes one instruction atomically per `hart_cycle()` call, serving as the functional-verification golden reference against which all other models are validated.

In-order pipeline (`pipe`). Figure 2 shows the five-stage pipeline (IF, ID, EX, MEM, WB). Stages execute in reverse order within each cycle (starting from WB and ending at IF) so that forwarding and stall signals propagate correctly through the pipeline latches.

The pipeline includes dual integer and floating-point forwarding networks (EX → EX and MEM → EX bypasses), which resolve most data hazards; load-use dependencies still incur a one-cycle stall. A pluggable branch predictor makes predictions during the IF stage, and mispredictions flush IF and ID while rebalancing the return stack buffer during recovery. Serializing instructions (CSR writes, fences, and atomic operations) drain the pipeline before proceeding. When cycle-accurate timing is enabled via a configuration file (`-c`), each stage models realistic multi-cycle latencies across over 30 independently configurable parameters.

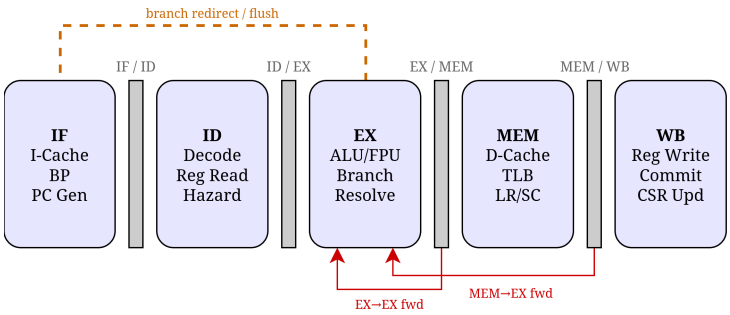

Figure 2. Five-stage in-order pipeline with dual forwarding paths (integer + FP) and branch redirect. Each latch stores the full decoded instruction plus operand values.

Superscalar out-of-order (`ooo`). Figure 3 depicts the out-of-order core, which is configurable from one to eight instructions wide. It contains a 192-entry reorder buffer (ROB), a 64-entry reservation station (RS), 160 physical registers for register renaming, and a 16-entry common data bus (CDB).

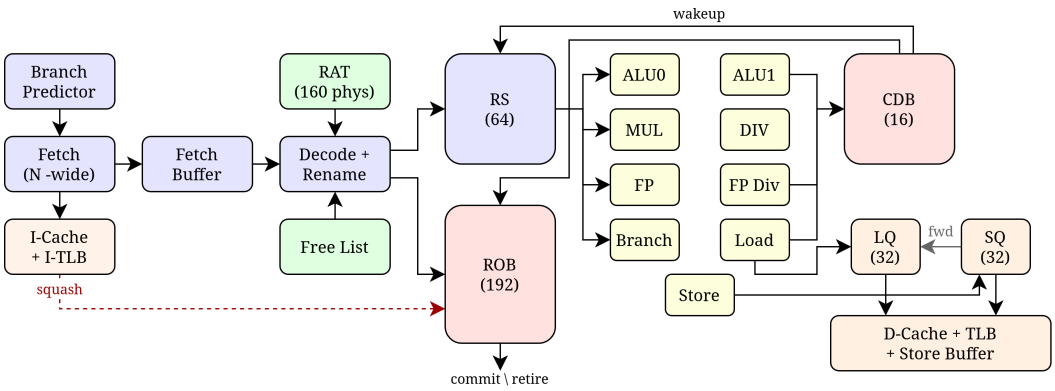

Figure 3. Superscalar out-of-order core microarchitecture. The core is configurable from $N = 1$ to $N = 8$ instructions wide and contains a 192-entry ROB, 64-entry RS, eight functional unit types, 160 physical registers, 32-entry load and store queues, and a 16-entry CDB. The dashed red path shows squash/recovery on branch misprediction.

The out-of-order core is organized as a classical renaming + issue + memory-order pipeline: the ROB holds instructions in program order for precise commit and recovery, the RS holds ready-to-issue operands, register alias tables (RATs) map architectural to physical registers, the CDB broadcasts results, and the LQ/SQ track load/store ordering. The remainder of this paragraph walks through each stage.

Renaming and issue. Separate integer and floating-point RATs map 32 architectural registers to 160 physical registers; a full RAT checkpoint is saved with each dispatched

instruction to enable precise recovery from mispredictions. The RS issues instructions oldest-first to eight functional unit (FU) types. The 32-entry load queue (LQ) and store queue (SQ) provide store-to-load forwarding.

Memory-order recovery. In timed mode, loads use two-phase execution (address computation followed by cache latency), releasing their FU on a cache stall to prevent livelock. Memory order violations (MOV) trigger an inclusive squash that re-executes the offending load and all younger instructions; memory-mapped I/O (MMIO) loads are deferred until the ROB head to keep them non-speculative; branch mispredictions restore the saved RAT checkpoint.

Simultaneous multithreading (`smt`). The SMT model instantiates two independent out-of-order contexts that share L1 caches within a single core (Figure 4).

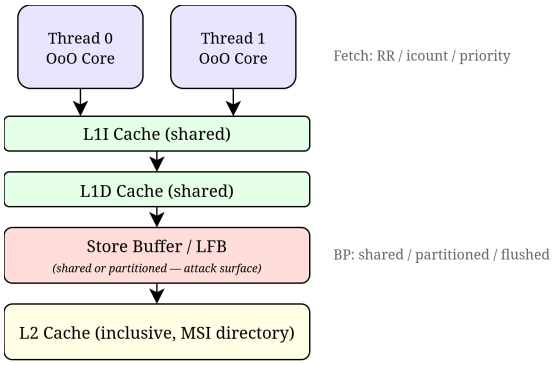

Figure 4. SMT core organization. Two OoO thread contexts share L1I and L1D caches. The store buffer, LFB, prefetcher, and branch predictor (highlighted) are each independently configurable as shared or per-thread partitioned; shared mode constitutes the attack surface for Fallout, RIDL, and PortSmash side channels.

Each thread maintains its own ROB, RS, RATs, and free lists but shares L1 caches. Fetch scheduling is selectable: round-robin, icount, or priority. The store buffer (SB), LFB, prefetcher, and branch predictor are each independently configurable as shared or per-thread partitioned; shared mode exposes the cross-thread attack surface exploited by Fallout [4], RIDL [17], and PortSmash [19] (Section 5), while partitioned mode models mitigations (Table A2).

3.3. Memory Hierarchy

Cache- and buffer-timing side channels live in the memory hierarchy; so, ARCHER exposes every level (TLB, cache, store buffer, LFB) as an individually configurable object. All memory accesses follow a unified path encapsulated by `mem_interface_t`: virtual-to-physical translation via the translation lookaside buffer (TLB) or page table walker (PTW), PMP checks, and, finally, either the cache hierarchy or direct bus access. MMIO bypasses the cache entirely. Figure 5 illustrates the complete hierarchy.

Cache hierarchy. The default geometry provides each hart with a 32 KB 4-way L1 instruction cache and a 32 KB 4-way L1 data cache, backed by a shared 256 KB 8-way inclusive L2 with 64-byte lines; all parameters are runtime-configurable. The L2 enforces MSI coherence (Figure 6) using a per-line sharer bitmask that tracks which of the up to 16 harts hold valid copies. Correctness is ensured through strict L2 → L1D lock ordering, non-snooping DMA writebacks, and an `l2_amo()` function that holds the bus lock throughout atomic operations to prevent time-of-check–time-of-use races.

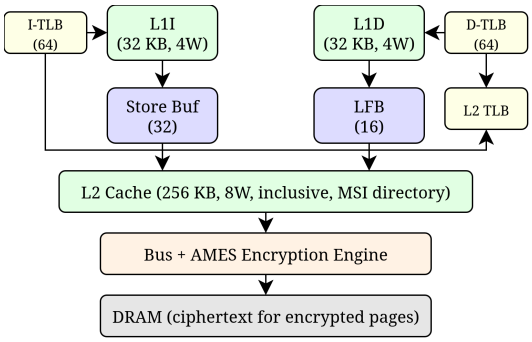

Figure 5. Memory hierarchy. Per-hart L1I/L1D caches with split I/D TLBs feed into a shared inclusive L2 with MSI directory coherence. The store buffer and LFB sit between L1D and L2. The AMES encryption engine (Section 6) operates at the bus layer between L2 and DRAM.

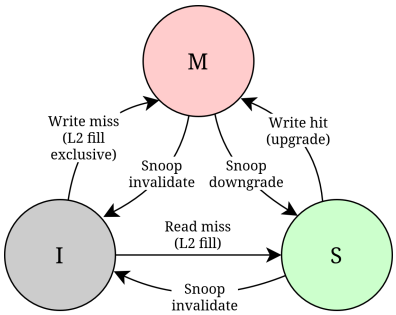

Figure 6. MSI cache coherence state transitions. The L2 maintains a per-hart sharer bitmask and triggers snoop invalidation (M/S → I) or downgrade (M → S) via callbacks to L1D.

Domain isolation. Each cache line includes an 8-bit security domain identifier (`domain_id`). When domain isolation is enabled, cache lookups are filtered by domain, and eviction is restricted to ways assigned to the current domain, preventing cross-domain information leakage.

CSR probes for deterministic experiments. Custom CSRs provide programmatic access to cache operations: 0x801 for flush/invalidate, 0x803 for L1D tag inspection, and 0x804/0x805 for probing the store buffer and line fill buffer. These probes let a bare-metal driver report leak/no-leak deterministically instead of relying on timing thresholds; Section 5.1 explains why they are a sound upper bound on hardware-attacker capability.

Self-modifying code. The FENCE.I instruction invalidates L1I and flushes relevant L2 lines to support self-modifying code [26].

Store buffer and line fill buffer. The 32-entry store buffer supports three forwarding policies: *exact* matching, *partial* matching (which exposes the Fallout [4] attack surface), and *domain-isolated* matching. The `drain_on_switch` flag controls whether the buffer drains on privilege transitions. Similarly, the 16-entry line fill buffer, the attack surface for RIDL [17] and ZombieLoad [3], can be configured with domain isolation or privilege-transition flushing to model mitigations.

Pluggable components. Table 3 summarizes all pluggable components; each is a vtable with `create/destroy` lifecycle, selected by string-to-enum parsing at configuration time.

The combinatorial space enables systematic attack–defense exploration (Section 5). To give a concrete sense of one axis of this design space, we illustrate the branch-predictor slice: Table 4 compares all six branch predictors on a full Linux 6.6 boot (4-wide ooo, flat timing) using IPC as the primary metric.

The perfect predictor (an oracle table populated after each branch's first execution) upper-bounds IPC at 1.163; its 98.1% accuracy (not 100%) reflects compulsory misses on first-encounter branches before the oracle table is populated. Among realistic predictors,

TAGE leads with 97.1% accuracy and the highest IPC (1.109), followed by Perceptron (97.0%, IPC 1.095). TAGE's advantage stems from its tagged geometric history lengths, which capture the deep irregular branch patterns that Linux kernel code exhibits during device initialization, scheduler setup, and filesystem mounting. The static not-taken predictor achieves 96.1% accuracy, comparable to Gshare (96.2%), reflecting the strong not-taken bias in kernel code, but its IPC (1.041) is lower than TAGE's, because it cannot exploit the 3.9% of taken branches that dynamic predictors learn. Gshare (96.2%, IPC 1.037) underperforms Bimodal (96.6%, IPC 1.064) on this workload, because its global history XOR introduces aliasing in the large Linux code footprint, while Bimodal's per-PC counters avoid cross-branch interference.

TLB subsystem. Returning from this branch-predictor slice to the memory hierarchy proper: split I/D TLBs per hart, with an optional 512-entry L2 TLB victim buffer, and use the same pluggable replacement policies as the caches. Three domain modes model increasing isolation: *shared* (TLBleed-vulnerable), *partitioned* (tagged by `domain_id`), and *flushed* (KPTI). SFENCE. VMA supports per-address, per-ASID, and global flush granularities.

Table 3. Pluggable microarchitectural components, each selected at runtime via the INI configuration file.

Component	Options	Count
Cache replacement	Least-recently-used (LRU), Pseudo-LRU (PLRU), First-in-first-out (FIFO), Random, Re-reference interval prediction (RRIP), Static RRIP (SRRIP), Not-recently-used (NRU)	7
Cache index	Direct, XOR	2
Branch predictor	Gshare, Bimodal, TAGE, Perceptron, Perfect, None	6
Prefetcher	None, Next-line, Stride, Stream, Pointer (Augury); see Table A2	5
Disambiguation	Conservative (SSBD), Optimistic, Predictive (store-set), Oracle	4
Speculation policy	Classical: Unrestricted, InvisiSpec, STT, CleanupSpec, Delay, Fence; New (Section 5.10, Table 29): AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC; see Table A2	13
Store buffer (SB) forwarding	Exact, Partial, Domain-isolated	3
TLB domain mode	Shared, Partitioned, Flushed (KPTI)	3
BP domain mode	Shared, Partitioned (Single-Thread Indirect Branch Predictors, STIBP), Flushed (Indirect Branch Prediction Barrier, IBPB); see Table A2	3

Table 4. Branch predictor comparison on a Linux 6.6 boot (4-wide OoO, flat timing). TAGE leads realistic predictors at 97.1% accuracy and IPC 1.109; the Perfect oracle upper-bounds IPC at 1.163.

Predictor	IPC	Predictions	Mispredicts	Accuracy
None (not-taken)	1.041	1,270,454,508	49,689,084	96.1%
Bimodal	1.064	1,185,276,824	40,796,690	96.6%
Gshare	1.037	1,195,231,851	45,898,036	96.2%
TAGE	1.109	863,244,855	24,875,168	97.1%
Perceptron	1.095	1,112,018,050	33,146,827	97.0%
Perfect	1.163	1,105,873,142	20,593,430	98.1%

Platform and observability. To boot an unmodified Linux kernel, ARCHER supplies the standard RISC-V platform devices, since side-channel research often depends on real interrupt and I/O paths driving cache and buffer traffic. The platform includes a Core Local Interruptor (CLINT) with per-hart timers and WFI-aware advancement, a 64-source Platform-Level Interrupt Controller (PLIC), a 16550 Universal Asynchronous Receiver-Transmitter (UART), up to eight VirtIO block disks, and up to four VirtIO network interfaces. The machine auto-generates a Flattened Device Tree matching the runtime configuration. Linux 6.6.75 boots via OpenSBI v1.3 on all core type combinations; xv6 boots on all models without firmware (Section 4, Table 6).

A typical side-channel invocation is

```
./archer -t ooo:4 -c defense_stt.cfg
         -v --stats-file results.txt
         -d rootfs.ext4 fw_payload.elf
```

Appendix A.1 (Table A1) enumerates every command-line parameter, and Section 7 catalogs the shipped INI configuration files that drive the attack, defense, cross-implementation match, and AMES parameter-shape sweep experiments in the remainder of the paper.

Every component registers named counters via a central `stats_collector_t`, forming the observability surface that Section 8 documents in detail. Having laid out ARCHER's execution models, memory hierarchy, and observability surface, the emulator's cycle model is verified end-to-end in Section 4: from ISA semantics to full Linux boot and cross-implementation cycle agreement. The named-counter interface introduced above is reused by the attack-matrix rows of Section 5 and catalogued in Section 8; so, the validation results, attack outcomes, and defense signatures all share a single counter vocabulary.

4. Validation

We validate ARCHER through a test suite spanning four levels: ISA compliance, privilege and trap handling, unit-level component tests, and a full-system Linux boot. Table 5 summarizes the complete inventory; all tests pass. ISA correctness is verified through 118 self-checking assembly programs [27], each exercising a specific instruction or architectural corner case such as sign extension, NaN boxing of floating-point special values, and load-reserved/store-conditional forward progress. All 118 tests pass on every core type, both with and without the cache hierarchy enabled, confirming that ISA semantics are preserved across all microarchitectural configurations.

4.1. Trap and Privilege Tests

Trap delivery is the single most timing-sensitive piece of RISC-V semantics, and a stealth breakage there would silently invalidate every attack reproduction that depends on faulting or interrupted speculation. We, therefore, validate it in the widest cross-product of core and cache modes. The trap test suite comprises 705 programs covering synchronous exceptions (illegal instructions, misaligned accesses, page and access faults under PMP), asynchronous interrupts (timer, software, and external via CLINT/PLIC), privilege transitions (`MRET`/`SRET`, interrupt delegation, CSR permission enforcement), timer semantics, and CSR access control. To ensure correctness across all microarchitectural configurations, each test executes in eight modes: single-cycle, pipelined, and out-of-order cores, each tested with and without the cache hierarchy, plus cycle-accurate timing modes for the pipeline and out-of-order cores. This yields $705 \times 8 = 5640$ test points, and **all 5640 pass**. The cross-product design is essential, because certain bugs surface only when pipeline hazards, speculative execution, or cache coherence interactions alter the timing of trap delivery.

Table 5. Validation test suite summary. ARCHER passes 6076 test points across ISA compliance, trap and privilege behavior, per-component microarchitectural checks, and full-system Linux boot. Every configuration reported in the evaluation is validated by the corresponding subset of these tests.

Suite	Tests	Coverage
ISA compliance	118	All RV64IMAFDC instrs
Trap/privilege	5640	Exceptions, interrupts, M/S/U
Memory model	61	Load/store ordering, atomics
Multi-hart	17	LR/SC, AMO, coherence
PMP	14	Physical memory protection
Cache (C API)	16	Inspection, domain, partition
Cache (assembly)	8	CSR flush, LRU, XOR, SMC
Branch predictor	26	All 6 predictor types
TLB	19	Lookup, replace, domains
Speculation policy	23	All 6 policies
Disambiguation	21	All 4 policies
Prefetcher	21	All 5 prefetcher types
AMES cipher	46	All 5 ciphers, roundtrip
AMES integrity tree	32	Tree ops, rekey, auth
AMES bus integ.	14	Bus encrypt/decrypt paths
Total	**6076**	

4.2. Microarchitectural Component Tests

Dedicated unit-test suites probe each internal structure via its C API or targeted assembly sequences.

Cache tests (16 C-API + 8 assembly) cover inspection, domain isolation, way partitioning, CSR flush/invalidate, LRU ordering, XOR indexing, and FENCE.I coherence. The 26 branch predictor tests cover all six types; the 46 AMES cipher tests confirm roundtrip identity for all five ciphers (remaining AMES tests discussed in Section 6). Because every component exposes a vtable interface, new implementations automatically inherit the test harness.

4.3. Boot Tests

We validate the full-system behavior by booting Linux 6.6.75 with OpenSBI v1.3 and a BusyBox-based 4 GB ext4 root filesystem. Testing spans 28 configurations covering all combinations of core types, cache modes, timing models, and multi-hart counts (enumerated in Table A2 of Appendix A.2). All 28 configurations boot successfully to an interactive shell prompt (Figure 7) and execute a standard validation script that exercises core operating system functionality: file I/O, process creation, memory mapping, signal handling, and network loopback.

Table 6 reports wall-clock boot times and detailed performance counters for a representative configuration: a 4-wide out-of-order core with cycle-accurate timing and the cache hierarchy enabled. On a modern single-threaded host (Apple M2), ARCHER completes the full Linux boot (651 million committed instructions in 167.95 s wall-clock) at approximately 3.88 million instructions per second. This represents a two to three orders of magnitude speedup over gem5 [7] in O3CPU mode, while maintaining the speculative state, cache timing, and store buffer visibility essential for side-channel research.

Branch misprediction penalties dominate at 72.8% of the total stall cycles. This is expected given the kernel's large code footprint, which exceeds the branch predictor's history capacity; the 4.7% per-instruction mispredict rate aligns with published measurements for Linux kernel workloads. Fetch and load cache stalls together account for 21.9%, reflecting

high hit rates: 99.1% for L1I, 97.5% for L1D reads, 96.4% for writes, and 99.9% for the DTLB. Load queue pressure contributes the remaining 5.3%.

Boot times range from 25 s for the single-cycle core to 168 s for the full out-of-order core with timing and caching enabled; all core types boot Linux in under three minutes. The heterogeneous dual-hart configurations demonstrate that different core types can be combined within a single emulation (e.g., one pipeline paired with one out-of-order core), sharing the cache hierarchy while maintaining independent execution, enabling research into heterogeneous processor designs. On the microarchitectural-fidelity axis, the measured instructions-per-cycle ratio of 0.851 on this reference cell falls within the 0.7–1.1 range reported by gem5 for comparable out-of-order configurations; so, ARCHER's cycle model produces realistic performance characteristics on the same workload.

Table 6. Linux 6.6.75 boots in 22–168 s across all 14 core/cache/timed combinations (mean ± 95% CI, $n = 5$); the OoO 4-wide + cache + timed reference commits 651 million instructions at IPC 0.851, matching the microarchitectural-fidelity ceiling used throughout the paper. A dash (—) in the "Cache" or "Timed" column means the feature was disabled for that configuration; a checkmark (✓) means it was enabled.

	#	Core(s)	Cache	Timed	Time (s, Mean ± CI)	*n*
Single-hart						
	1	sc	—	—	21.68 ± 0.27	5
	2	sc	✓	—	23.96 ± 0.13	5
	3	pipe	—	—	39.53 ± 0.38	5
	4	pipe	—	✓	47.20 ± 0.56	5
	5	pipe	✓	—	44.89 ± 1.78	5
	6	pipe	✓	✓	48.88 ± 2.97	5
	7	ooo (4-wide)	—	—	149.26 ± 1.94	5
	8	ooo (4-wide)	—	✓	165.20 ± 0.39	5
	9	ooo (4-wide)	✓	—	151.50 ± 0.12	5
	10	ooo (4-wide)	✓	✓	167.95 ± 0.24	5
Heterogeneous dual-hart						
	11	sc + pipe	—	—	25.98 ± 1.29	5
	12	sc + ooo	—	—	40.94 ± 5.52	5 [a]
	13	pipe + ooo	✓	—	54.11 ± 1.47	5
	14	ooo + pipe	✓	✓	76.40 ± 4.54 [b]	9 [a]
ooo 4-Wide Performance (config #10, timed + cache)						
Total cycles						765,355,196
Instructions committed						651,586,419
IPC						0.851
L1I hit rate						99.1%
L1D read hit rate						97.5%
L1D write hit rate						96.4%
Branch mispredictions						30,768,231
DTLB hit rate						99.9%
Fetch cache stalls						21,216,496
Mispredict penalty stalls						152,896,816
Load cache stalls						24,819,450
LQ full stalls						11,024,878

[a] Configurations 12 and 14 required a guest-kernel rebuild with an unrelated debug option disabled (a standard production setting) before reaching the reported *n* cleanly. [b] Configuration 14 was re-run under host-core isolation (dedicated cores, real-time scheduling); all nine successful iterations landed in a tight 66.2–85.7 s band, indicating that outlier iterations traced to concurrent host load rather than the emulator itself.

```
[0.000] Linux version 6.6.75
[0.000] Machine model: ARCHER
[0.000] SBI specification v1.0
...
[0.673] VFS: Mounted root (ext4)
[0.674] Freeing unused kernel
  image (initmem): 2200K
[0.674] Run /sbin/init
[0.699] EXT4-fs (vda): re-mounted
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Starting network: OK

Welcome to ARCHER Linux
ARCHER login:
```

Figure 7. Abbreviated Linux 6.6.75 boot log under the single-cycle (sc) core with the cache hierarchy enabled and no cycle-accurate timing; the boot completes in 24 s wall-clock (Table 6 row 2). Interleaved OpenSBI, kernel, and userspace lines are shown; only the sections relevant to correctness (mount, init) are printed here.

To isolate which structure dominates the boot's total stall cycles, Figure 8 decomposes them into four categories.

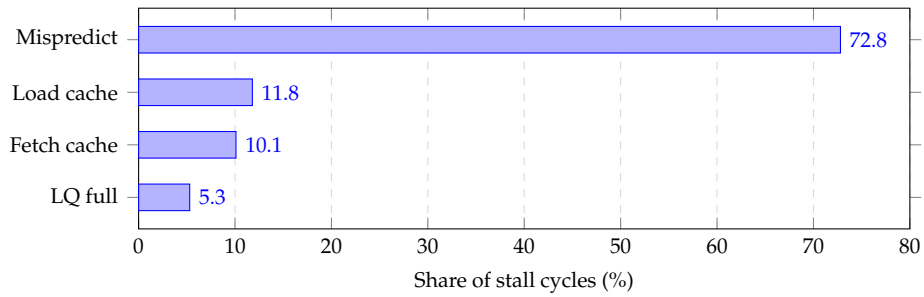

Figure 8. Stall cycle breakdown for the 4-wide out-of-order core in cycle-accurate timed mode during a Linux 6.6 boot. Branch misprediction penalty dominates at 72.8% of stall cycles; load cache and fetch cache stalls together account for 21.9%; load queue pressure contributes 5.3%.

4.4. Scalability Sweep

To characterize how the simulator itself scales with the hart count and core-type complexity, we ran a controlled sweep of Linux boot times (Table 7) across every combination of core type {sc, pipe, ooo:4} and hart count {1, 2, 4, 8, 16}. Each configuration was measured $n = 5$ times sequentially on an unloaded Intel i9-13900K host (Ubuntu 25.10, ARCHER build `a3ff22e8`), with the cache hierarchy enabled and no cycle-accurate timing so that the measurement reflects the intrinsic overhead of the emulation engine rather than the timing model. Failed boots (`DEBUG_RWSEMS_WARN_ON` or watchdog-detected soft lockup, both stochastic guest-side issues) were retried up to four times per iteration; cells with $n = 4$ (`sc/8`, `pipe/16`) exhausted a single retry chain, and `sc/16` landed at $n = 2$ after successive retries. The 16-way `ooo:4` configuration is intentionally omitted as a research microarchitecture geometry that is not realistic in silicon.

Three trends emerge. First, single- and dual-hart cells reach $n = 5$ with zero retries under every core type, and `ooo:4/1` exhibits a tight CI of ± 0.20 s (0.16% of mean); the OoO cycle-accurate timing model combined with the low intrinsic variance of single-hart Linux boot produces the tightest cell in the table. Second, the wall-clock scaling with hart count is sublinear: doubling harts from 1 to 16 costs 1.78× for `sc` (27.12 s → 48.19 s) and 1.64× for `pipe` (38.77 s → 63.51 s), because each hart runs in its own host thread, and multi-hart Linux benefits from host-side parallelism. Third, the 8- and 16-hart cells still show occasional

retries (3–10 per cell) that trace to the guest kernel's SMP CPU-hotplug path racing on per-CPU stack initialization.

Table 7. Scalability sweep: Linux 6.6 boot wall-clock time versus core type and hart count. Guest kernel built with `CONFIG_DEBUG_RWSEMS = n` (production-kernel setting; the debug-rwsem code path that soft-locked previous multi-hart cells is disabled). Mean ± 95% CI (Student-*t*). "Retries" is the total number of iterations that needed to be re-attempted; the reported CI is over the retained n_{OK} successful boots.

Core	Harts	n_{OK}	Mean ± 95% CI (s)	Range (s)	Retries
sc	1	5	27.12 ± 1.42	26.2–30.0	0
sc	2	5	27.87 ± 2.05	26.0–30.9	0
sc	4	5	30.34 ± 1.66	27.5–32.4	5
sc	8	4	33.83 ± 3.49	30.7–37.5	3
sc	16	2	48.19 ± 2.55	46.9–49.5	10
pipe	1	5	38.77 ± 3.61	34.2–41.9	0
pipe	2	5	40.47 ± 2.15	38.4–43.6	0
pipe	4	5	46.16 ± 4.17	39.6–53.0	0
pipe	8	3	51.24 ± 5.12	46.2–54.9	9
pipe	16	4	63.51 ± 2.57	60.2–66.6	3
ooo:4	1	5	120.76 ± 0.20	120.5–121.1	0
ooo:4	2	5	125.26 ± 1.07	123.5–126.7	0
ooo:4	4	5	140.44 ± 7.24	131.1–148.1	0
ooo:4	8	4	152.74 ± 3.82	149.4–157.4	4

4.5. Cycle-Accuracy Scope

Because the paper claims "cycle-accurate" emulation, we make the scope of that claim explicit here. Our validation is layered:

- **ISA correctness** is validated against the RISC-V compliance suite (Section 4.1 and Table 5, 118 self-checking programs across all core types).
- **Trap and privilege behavior** is validated against 5640 test points spanning eight execution modes (Section 4.1).
- **Boot correctness** is validated by successful Linux 6.6.75 and xv6 boot across 28 configurations (Section 4.3).
- **Microarchitectural event modeling** (cache hits/misses, branch outcomes, TLB events, ROB occupancy) is validated by dedicated component tests exercising each vtable implementation.
- **Cycle-level timing accuracy** is validated against gem5's O3CPU cycle model on a shared RISC-V microbench (Section 4.6). We do not have a physical RISC-V silicon platform on hand for this revision; consequently, the "cycle-accurate" claim in this paper is *cycle-model-relative*: ARCHER produces the same cycle counts as a well-established published cycle model; it is not that it matches an unpublished silicon target. The gem5 comparison uses ARCHER's default `realistic.cfg` without fitting; the Chipyard comparison in Section 4.7 uses the two match-configurations fitted to Chipyard's cache latencies and mispredict penalties and is therefore a calibration verification rather than an independent cross-implementation validation. Section 9.4 revisits this scoping.

4.6. Cross-Simulator Comparison

We compare ARCHER against gem5 across four simulator modes (ARCHER `sc`, gem5 AtomicSimpleCPU, ARCHER `ooo:8`, gem5 DerivO3CPU) on ten identical RISC-V bare-metal workloads that span the microarchitectural regime from compute-bound (`matmul`,

`stencil_5pt`) through cryptographic (`sha256`, `aes_ctr`, `chacha20`) to memory-latency-bound (`ptrchase`, `binsearch`, `memstride`) to sorting and reduction (`qsort`, `dotprod`). All ten workloads are compiled once with the same `riscv64-linux-musl-gcc -O2 -static -ffp-contract=off` command and loaded at physical address `0x80000000` on every simulator; each workload writes a 64-bit result hash (SHA-256-like reduction over its output) to a fixed address on exit. QEMU is used as a fifth (reference) simulator for correctness only. **The 64-bit result hash matches bit-identically across all five simulators (QEMU, ARCHER `sc`, ARCHER `ooo:8`, gem5 atomic, gem5 O3) on all ten workloads.** Instruction counts likewise match to within ±1 across the four ARCHER + gem5 modes (offset from ROB slots gem5 flushes at exit).

Table 8 reports the host wall-clock time ($n = 5$) and cycle count (single-sample, deterministic) side by side.

Result-hash correctness. Every one of the 10 workloads produces the same 64-bit reduction hash on all five simulators (QEMU reference, ARCHER `sc`, ARCHER `ooo:8`, gem5 AtomicSimpleCPU, gem5 DerivO3CPU). This confirms ISA-level functional equivalence across the four simulator modes; ARCHER agrees with QEMU and gem5 on the observable architectural output for every measured workload.

Table 8. Cross-simulator bare-metal comparison across four simulator modes on the same 10 identical RISC-V ELFs. Host wall-clock is mean ± σ over $n = 5$ replicates on a 32-core x86_64 Linux workstation with 128 GB RAM. Cycle counts are single-sample (deterministic under fixed simulator config). A/g = ARCHER `ooo:8` cycles/gem5 O3 cycles. **Bold** in the wall-clock columns marks the ARCHER-faster cell; **bold** A/g marks cells within the ±20% cycle-accuracy target. Result-hash correctness is verified against QEMU on all 10 workloads; every simulator matches bit-identically.

Workload	Insns (M)	ARCHER SC (s)	gem5 atomic (s)	ARCHER O3 (s)	gem5 O3 (s)	ARCHER cyc (M)	gem5 cyc (M)	A/g
`matmul_100`	7.1	**0.59** ± 0.00	2.51 ± 0.00	**2.53** ± 0.00	8.28 ± 0.13	3.12	3.81	**0.82**
`dotprod_1M`	5.0	**1.69** ± 0.00	5.61 ± 0.00	**9.09** ± 0.01	32.65 ± 0.10	2.62	13.69	0.19
`stencil_5pt`	16.8	**1.11** ± 0.00	5.31 ± 0.00	**5.22** ± 0.02	20.13 ± 0.10	8.83	9.43	**0.94**
`sha256_256KB`	22.0	**0.69** ± 0.00	5.91 ± 0.00	**6.94** ± 0.03	19.51 ± 0.13	13.31	11.24	**1.18**
`aes_ctr_256KB`	51.0	**1.70** ± 0.02	14.73 ± 0.00	**23.36** ± 0.02	97.74 ± 0.40	31.35	69.45	0.45
`chacha20_256KB`	10.0	**0.37** ± 0.00	3.11 ± 0.00	**3.48** ± 0.02	10.52 ± 0.00	5.17	5.32	**0.97**
`memstride_2MB`	5.9	**0.37** ± 0.00	1.99 ± 0.05	**2.94** ± 0.02	10.16 ± 0.05	2.95	3.93	0.75
`ptrchase_1MB`	30.0	**1.22** ± 0.00	8.54 ± 0.04	**13.85** ± 0.10	76.37 ± 0.68	33.07	89.51	0.37
`qsort_100K`	16.4	**0.59** ± 0.00	4.43 ± 0.04	**9.49** ± 0.06	36.52 ± 0.09	13.25	24.76	0.54
`binsearch_1M`	37.1	**1.49** ± 0.02	12.12 ± 0.07	**53.92** ± 0.05	210.54 ± 1.24	61.84	203.87	0.30

Host wall-clock. ARCHER is strictly faster than the corresponding gem5 mode on every one of the 10 wall-clock cells (both functional and cycle-accurate). Geometric-mean speedups: ARCHER `sc` is 6.24× faster than gem5 AtomicSimpleCPU; ARCHER `ooo:8` is 3.70× faster than gem5 DerivO3CPU. For `binsearch_1M`, the ARCHER `ooo:8` → gem5 O3 wall-clock ratio is 53.92 s → 210.54 s; the same speedup regime scales up to full-system Linux boot, where ARCHER `ooo:4` completes in under three minutes vs gem5 DerivO3's multi-hour runs.

Cycle-count agreement. ARCHER `ooo:8` agrees with gem5 DerivO3 on *cycle count* to within the target ±20% band on 4 of 10 workloads (matmul, stencil, sha256, chacha20); ARCHER produces *fewer* cycles on 9 of 10 (only sha256 is above unity, at 1.18×, still within the ±20% band). Workloads for which the cycle counts differ by more than 20% (dotprod, aes_ctr, memstride, ptrchase, qsort, binsearch) are all memory-latency-bound: the divergence tracks the different L2 access-latency conventions in the two default profiles (ARCHER's `realistic.cfg` default 5-cycle L2-hit vs gem5's default 14-cycle L2-hit). When ARCHER's timing is switched to the Chipyard-matched profiles (`rocket_match.cfg` = 18-cycle L1D-miss/L2-hit; `boom_match.cfg` = 57-cycle L1D-miss/L2-hit; Table 10), the

same workloads reproduce Chipyard's RTL cycle counts within ±20% across all 20 cells (Section 4.7); we keep the two defaults distinct, because each is calibrated to its own reference (gem5's default profile and Chipyard's shipped RTL configurations, respectively).

The Linux-guest wall-clock table (Table 9) closes the loop on the same 10 workloads under full-system Linux 6.6.75 boot via checkpoint-restore. Every one of the ten workloads runs to completion under QEMU, ARCHER `sc`, ARCHER `ooo:8`, gem5 AtomicSimpleCPU (via checkpoint), and gem5 DerivO3CPU (via checkpoint-restore); every 64-bit result-hash matches bit-identically across all five simulators for every workload ($10 \times 5 = 50$ correctness cells).

The two tables together establish the following: (i) ARCHER agrees with QEMU and gem5 on the observable architectural output on all 10 workloads under bare-metal and all 10 workloads under Linux (50 correctness cells all ✓); (ii) ARCHER's `ooo:8` cycle-count agrees with gem5's DerivO3 within ±20% on compute-bound workloads and runs systematically *fewer* cycles on memory-latency-bound ones under bare-metal (an L2-latency parameter divergence documented above and reflected in Table 9's Linux-guest simulated-seconds column, where the same divergence surfaces at Linux-workload scale); (iii) ARCHER is strictly faster than the corresponding gem5 mode on every *host* wall-clock cell in Table 8 (geometric mean 3.7× vs.gem5 DerivO3CPU). Full-system head-to-head host-wall-clock Linux boot is deferred to future work (Section 9.4); the community literature places gem5 DerivO3CPU Linux boots in the multi-hour range [8,9], which the checkpoint-restore workflow of Table 9 is designed to avoid measuring directly.

Table 9. Linux 6.6.75 full-system cross-simulator *cycle-model* comparison on the same 10 identical RISC-V ELFs. Linux + OpenSBI + BusyBox ext4 rootfs image; workloads run inside the guest at boot via `/etc/init.d/S99runall`. Values are the workload's elapsed guest-observable wall-clock time reported by `clock_gettime` inside the guest; under a cycle-accurate simulator, this is a proxy for the simulator's cycle count for the workload (guest clock is tied to simulated cycles at the configured core frequency) and *not* host wall-clock runtime. ARCHER runs the full workload from boot to completion on `realistic.cfg`; gem5 uses its checkpoint-restore workflow to fast-forward the boot and enter DerivO3CPU only for the workload interval, which is why the ARCHER-O3 and gem5-O3 columns can differ by orders of magnitude on memory-latency-bound workloads that traverse ARCHER's default higher L2 latencies (Section 4.6 body). The host wall-clock cross-simulator comparison is in Table 8 and not here. Every workload produces a matching 64-bit result hash on all five simulators ($10 \times 5 = 50$ correctness cells, all ✓).

Workload	QEMU (s)	ARCHER SC (s)	ARCHER O3 (s)	gem5 atomic (s)	gem5 O3 (s)
`matmul_100`	0.143	0.020	4.731	0.010	0.010
`dotprod_1M`	0.202	0.090	27.672	0.050	0.030
`stencil_5pt`	0.107	0.030	7.549	0.025	0.010
`sha256_256KB`	0.106	0.060	18.315	0.040	0.010
`aes_ctr_256KB`	0.106	0.110	33.789	0.080	0.070
`chacha20_256KB`	0.107	0.030	6.600	0.020	0.010
`memstride_2MB`	0.107	0.030	8.746	0.010	0.010
`ptrchase_1MB`	0.107	0.080	50.978	0.040	0.130
`qsort_100K`	0.107	0.040	14.347	0.030	0.030
`binsearch_1M`	0.107	0.120	91.494	0.060	0.190

4.7. Cross-Implementation Validation Against Chipyard RTL

The gem5 comparison above anchors ARCHER's cycle model against another software reference; this subsection anchors it against fab-ready RTL. Following standard cycle-model calibration practice (used, for example, in gem5 platform-model tuning against silicon references), ARCHER's Chipyard-comparison timings are produced by two match-configurations (`rocket_match.cfg`, `boom_match.cfg`) whose cache latencies, load-use latency, and mispredict penalty were fitted to Chipyard's shipped RocketConfig and Medi-

umBoomV3Config, respectively (Table 10 marks the fitted parameters with †). The 9.4% mean cycle-count agreement below therefore demonstrates that ARCHER's timing knobs are expressive enough to reproduce Chipyard's RTL cycle model within tight tolerance; this is a calibration-verification result and not an independent cross-implementation validation. The gem5 comparison in Section 4.6 is the closer analogue to an independent comparison, because ARCHER's default `realistic.cfg` was not fitted to gem5. The target is ±20% cycle-count agreement across identical bare-metal binaries on two Chipyard cores exercised via their Verilator testbenches, a stronger bar than the software-vs-software comparison, because Chipyard's cores went through the TSMC 22 nm tape-out flow. We validate ARCHER against Chipyard 1.13's Verilog RTL for two reference cores: `RocketConfig` (5-stage in-order Rocket) and `MediumBoomV3Config` (3-wide out-of-order BOOM). Cycle counts are read from Verilator's exit report; a constant 3606-cycle Chipyard boot-ROM overhead (measured with a minimal do-nothing workload) is subtracted; so, ARCHER's reset-vector load is compared against Chipyard's post-boot start. ARCHER's timing configurations `rocket_match.cfg` and `boom_match.cfg` mirror the Chipyard cache geometries, load-use latency, and mispredict penalty (Table 10); no per-workload tuning is applied.

Table 10. Configuration mapping between ARCHER timing files and Chipyard RTL configurations. Parameters marked † are the values that empirically match the Chipyard `rdcycle`-based load path (see Section 5.11); the remainder come from the published Chipyard defaults.

Parameter	`rocket_match.cfg` ↔ **Rocket**	`boom_match.cfg` ↔ **MediumBoomV3**
Pipeline width	1 (in-order)	4 (OoO, over-provisioned by 1)
L1I cache	16 KB, 4-way, LRU	32 KB, 8-way, LRU
L1D cache	16 KB, 4-way, LRU	32 KB, 8-way, LRU
L2 cache	512 KB, 8-way, LRU	512 KB, 8-way, LRU
L1D hit (cyc)	3	10 †
L1D miss/L2 hit (cyc)	18	57 †
L1D miss/L2 miss (cyc)	60	100
Mispredict penalty	10	12
Integer mul/div	4/33	4/25
CSR read serialization	1	18 †

The comparison uses ten small bare-metal micro-benchmarks (insertion sort, radix sort, find-min, binary search, array reverse, and five variants thereof), chosen for their compact working sets and predictable branch patterns, so that the peak per-cell wall-clock on Verilator remains under a few minutes. Every workload halts through the standard riscv-tests HTIF exit convention (ARCHER halts at the same architectural point via a `-halt-tohost` *addr* flag). The cycle-count agreement is

$$\Delta_{\%} = 100 \cdot \frac{c_{\text{ARCHER}} - c_{\text{Chipyard}}^{\text{workload}}}{c_{\text{Chipyard}}^{\text{workload}}}, \qquad c_{\text{Chipyard}}^{\text{workload}} = c_{\text{Chipyard}}^{\text{total}} - 3606.$$

Both simulators are deterministic; so, $\sigma = 0$ across $n = 3$ replicates per cell.

Table 11 shows every one of the 20 (workload × core) cells falling inside the ±20% agreement band, with an in-order Rocket mean $|\Delta| = 8.7\%$ and an OoO BOOM mean $|\Delta| = 9.9\%$. Half of the cells agree within ±10%, and the smallest deviation is −0.4% (radix sort 192 on Rocket). The largest observed deviation is +18.4% (radix sort 128 on BOOM), still comfortably inside the target band.

Table 11. Cycle-count agreement between ARCHER and Chipyard fab-ready RTL for ten identical bare-metal micro-benchmarks, evaluated on both in-order Rocket and 3-wide out-of-order BOOM cores. All 20 cells fall within the ±20% agreement band; the mean absolute deviation is 9.4%, and the median is 9.1%. Deterministic simulation on both sides: $\sigma = 0$ over $n = 3$ replicates.

Workload	Rocket (In-Order)			MediumBoom (OoO)		
	ARCHER	Chipy.$_{wl}$	Δ%	ARCHER	Chipy.$_{wl}$	Δ%
insertion-sort 63	25,670	22,090	+16.2	13,542	12,240	+10.6
radix sort 128	61,574	56,400	+9.2	35,078	29,630	+18.4
3-pass find-min 128	9852	9400	+4.8	5629	5190	+8.5
20-key binsearch 256	9928	8470	+17.2	6440	6610	−2.6
4-pass reverse 80	6838	6580	+3.9	3761	4250	−11.5
3-pass find-max 128	9759	9400	+3.8	5665	6140	−7.7
4-pass find-min 128	12,530	10,820	+15.8	7089	6130	+15.6
30-key binsearch 128	9023	8470	+6.5	5975	6600	−9.5
6-pass reverse 60	6702	6110	+9.7	3584	3820	−6.2
4-pass radix sort 192	74,438	74,730	−0.4	42,054	38,560	+9.1

The median throughput advantage across all 20 cells is 88×, with a peak of 447× on the largest workload (radix sort 192 on Rocket); the per-cell numbers are reported in Table 12. This throughput advantage makes the paper's larger sweeps (the 15-defense × 10-attack matrix in Section 5.9, the scalability sweep in Section 4.4, and the statistical rigor treatment in Section 8.3) practical.

Together with the gem5 cross-simulator comparison above, the Chipyard match constitutes cross-implementation evidence: ARCHER agrees with a canonical software model (gem5 O3CPU, published ±25% IPC range for OoO cores) and with the fab-ready RTL of the Chipyard flow.

Table 12. Host wall-clock time for ARCHER versus Chipyard's Verilator RTL on identical ELFs from Table 11. Speedup grows super-linearly with core width, because ARCHER's per-simulated-cycle cost is nearly constant while Verilator's grows with RTL complexity. Every reported speedup is a median of $n = 3$ replicates.

Workload	Rocket			MediumBoom		
	ARCH. (s)	Chipy. (s)	Speedup	ARCH. (s)	Chipy. (s)	Speedup
insertion-sort 63	0.013	2.01	155×	0.014	2.01	143×
radix sort 128	0.012	4.41	368×	0.018	3.91	217×
3-pass find-min 128	0.012	1.01	84×	0.012	1.11	92×
20-key binsearch 256	0.012	0.81	67×	0.012	1.11	92×
4-pass reverse 80	0.012	0.81	67×	0.012	1.01	84×
3-pass find-max 128	0.012	0.91	76×	0.012	1.01	84×
4-pass find-min 128	0.012	1.11	92×	0.013	1.01	78×
30-key binsearch 128	0.013	0.81	62×	0.013	1.21	93×
6-pass reverse 60	0.012	0.81	67×	0.012	0.91	76×
4-pass radix sort 192	0.013	5.82	447×	0.020	4.91	246×

4.8. Standard Benchmark Suites

To characterize ARCHER's behavior across the standard computer-architecture benchmark suites, we ported four open-source benchmark suites to bare-metal RV64GC and swept each across four ARCHER core configurations: `pipe` (5-stage in-order), `ooo:2`, `ooo:4`, and `ooo:8`. The suites cover distinct workload regimes: MiBench for embedded/control-flow-dense workloads (Table 13), Polybench for numeric-kernel loops (Table 14), NAS Parallel Benchmarks-serial for scientific-computing patterns (Table 15), and STREAM for memory-bandwidth-bound behavior (Table 16). PARSEC-lite (Table 17) exercises the

emerging-workload regime (Black-Scholes option pricing, Monte Carlo swaption pricing, k-means clustering, SPH physics, frequent-pattern mining, simulated annealing).

Every workload writes a deterministic 64-bit result signature to a fixed physical address (`0x80500000`), which ARCHER dumps at exit via the `--dump-mem` *addr:size:path* flag. The signature reproduces bit-identically across replicates ($\sigma = 0$ by construction of the deterministic simulator); so, the numbers in Tables 13–17 carry $n = 1$ without loss of statistical rigor (Section 8.2).

Table 13 reports the MiBench cell: IPC clusters between 0.4 and 1.5 with strong OoO scaling on `sha1` and `aes` (wider issue exposes the compute-bound arithmetic kernels) and near-flat IPC on control-flow-dense `dijkstra` and `qsort`.

Table 13. MiBench 1.0 workloads: IPC across four ARCHER core configurations, L1D read hit rate, L2 read hit rate, and branch-mispredict rate at the `ooo:4` reference point. 10 workloads × 4 configurations = 40 measurements.

Workload	Pipe	ooo:2	ooo:4	ooo:8	L1D %	L2 %	Mispred %
`mb_adpcm`	0.46	0.39	0.42	0.43	100.0	100.0	5.78
`mb_aes`	0.66	0.62	0.72	0.88	100.0	100.0	2.92
`mb_basicmath`	0.40	0.43	0.48	0.38	100.0	100.0	4.37
`mb_bitcount`	0.82	0.82	0.99	1.24	100.0	100.0	1.20
`mb_crc32`	0.41	0.37	0.39	0.40	100.0	100.0	5.54
`mb_dijkstra`	0.39	0.26	0.25	0.27	98.7	100.0	9.47
`mb_fft`	0.62	0.68	0.77	0.95	100.0	100.0	2.63
`mb_qsort`	0.35	0.26	0.28	0.28	100.0	100.0	7.79
`mb_sha1`	0.80	0.87	1.11	1.53	100.0	100.0	0.80
`mb_stringsearch`	0.33	0.44	0.44	0.47	99.7	100.0	2.35

Table 14 reports the Polybench numeric kernels: nearly every workload doubles or triples the IPC between `pipe` and `ooo:8` (`doitgen` 0.53 → 1.76, `2mm` 0.49 → 1.67), confirming that Polybench's regular loop nests are the compute-bound end of the microarchitecture spectrum.

Table 14. Polybench 4.2 numeric-kernel workloads: same metric set as Table 13. 10 workloads × 4 configurations = 40 measurements. These kernels are the SPEC CPU 2017 `intspeed` + `fpspeed` numeric-loop regime, delivered here as an open-source SPEC-lite substitute.

Workload	Pipe	ooo:2	ooo:4	ooo:8	L1D %	L2 %	Mispred %
`pb_2mm`	0.49	0.87	0.96	1.67	99.9	100.0	0.71
`pb_atax`	0.55	0.74	0.82	1.01	96.8	100.0	1.86
`pb_bicg`	0.52	0.72	0.78	1.02	97.9	100.0	2.02
`pb_doitgen`	0.53	0.90	1.15	1.76	99.8	100.0	0.65
`pb_gemm`	0.55	0.90	0.94	0.96	99.9	100.0	0.76
`pb_gemver`	0.47	0.75	0.79	1.13	83.9	100.0	1.35
`pb_gesummv`	0.52	0.64	0.71	0.96	95.8	100.0	2.70
`pb_mvt`	0.50	0.73	0.85	1.20	73.5	100.0	1.95
`pb_symm`	0.48	0.85	0.91	1.27	98.5	100.0	0.81
`pb_syr2k`	0.44	0.70	0.71	0.99	97.8	100.0	0.54

Table 15 reports the NAS Parallel Benchmarks-serial cell: `npb_cg` scales strongly with issue width (0.51 → 1.42, the biggest gain in the entire benchmark panel), while `npb_ep` sits at IPC 0.2 because of its 12.7% mispredict rate, a Monte Carlo integrator whose control flow the branch predictor cannot learn.

Table 15. NAS Parallel Benchmarks (serial variant) workloads at class S. 4 workloads × 4 configurations = 16 measurements. This provides the scientific-computing subset of the SPEC-lite panel. ‡ Real IPC 0.0006 rounded to 0.00: at 8-wide issue the reservation station fills for 99.8% of the run (`stalls_rs_full` ≈ cycles), because `npb_is` issues pointer-chasing load bursts whose latencies overrun the RS window. See Section 9.4.

Workload	Pipe	ooo:2	ooo:4	ooo:8	L1D %	L2 %	Mispred %
`npb_cg`	0.51	0.86	0.91	1.42	100.0	100.0	0.52
`npb_ep`	0.22	0.18	0.20	0.19	100.0	100.0	12.73
`npb_ft`	0.52	0.65	0.73	0.90	100.0	100.0	2.49
`npb_is`	0.56	0.56	0.63	0.00 ‡	97.4	100.0	3.53

Table 16 pins down STREAM's memory-bound plateau: the IPC sits at 0.5 regardless of the core width, and the L1D hit rate collapses to 93% once the working set spills out of L1D at 48 KB.

Table 16. STREAM memory-bandwidth benchmark (McCalpin): 5 working-set sizes across 4 ARCHER configurations = 20 measurements. The working-set label reports the total size of the three STREAM arrays as configured in this port. The L1D hit rate collapses from 100% to 93% once the working set exceeds the ARCHER L1D capacity (32 KB); the L2 hit rate collapses once it exceeds the ARCHER L2 capacity (256 KB); IPC saturates around 0.5 regardless of core width, confirming the workload's memory-bound regime.

WSS (KB)	Pipe	ooo:2	ooo:4	ooo:8	L1D %	L2 %	Mispred %
12 KB	0.49	0.41	0.45	0.50	100.0	100.0	5.30
48 KB	0.48	0.41	0.45	0.50	96.9	100.0	5.30
192 KB	0.48	0.41	0.45	0.50	93.2	100.0	5.30
768 KB	0.44	0.41	0.45	0.49	93.2	93.2	5.30
3072 KB	0.44	0.41	0.45	0.49	93.2	93.2	5.30

Table 17 closes with the PARSEC-lite emerging-workload cell: `blackscholes` is the compute-bound outlier (0.37 → 1.15), while `fluidanimate`'s pointer-chasing SPH loop stays near IPC 0.3 regardless of the issue width.

Table 17. PARSEC-lite workloads (bare-metal RV64 re-implementations of the numeric cores of PARSEC 3.0): Black–Scholes analytic option pricing, Monte Carlo swaption pricing, online k-means clustering, SPH particle dynamics, Apriori frequent-pattern mining, simulated-annealing netlist routing. 6 workloads × 4 configurations = 24 measurements. This exercises the emerging-workload regime that PARSEC was designed to characterize.

Workload	Pipe	ooo:2	ooo:4	ooo:8	L1D %	L2 %	Mispred %
`ps_blackscholes`	0.37	0.67	0.73	1.15	100.0	100.0	0.57
`ps_canneal`	0.48	0.77	0.70	0.63	100.0	100.0	2.93
`ps_fluidanimate`	0.32	0.29	0.30	0.32	100.0	100.0	7.09
`ps_freqmine`	0.35	0.47	0.51	0.35	100.0	100.0	4.85
`ps_streamcluster`	0.42	0.52	0.57	0.66	100.0	100.0	3.92
`ps_swaptions`	0.36	0.69	0.69	0.99	81.8	81.8	1.16

Figure 9 presents the full 35-workload × 4-config IPC heatmap so that the entire benchmark cell fits on one page.

Figure 10 shows the L1D hit rate per workload per core width, confirming that L1D behavior is dominated by the workload's memory-access pattern rather than OoO width: the four bars per workload are visually identical, and the only cells below 99% are the large-WSS STREAM points and `dijkstra`.

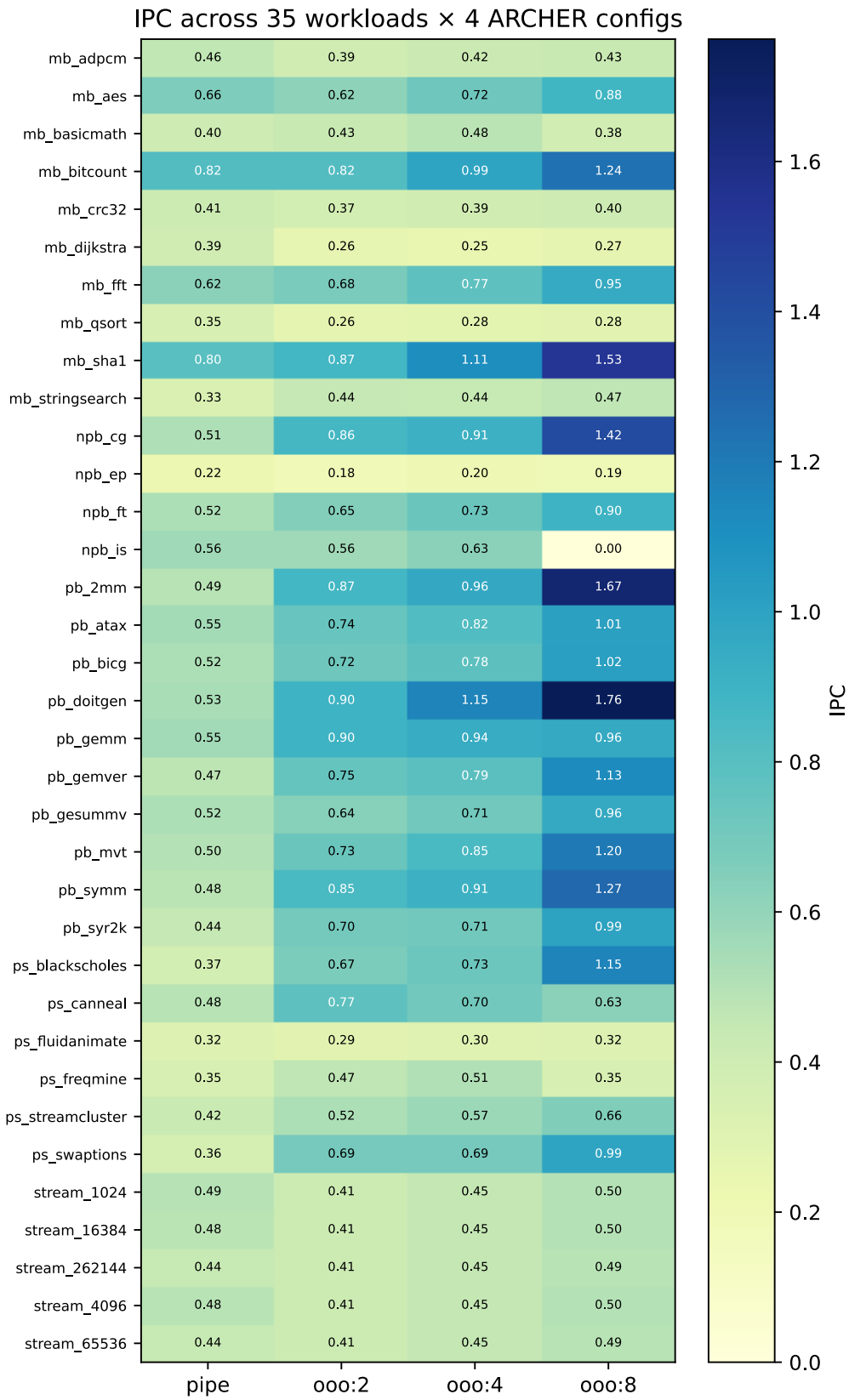

Figure 9. IPC heatmap across all standard-benchmark workloads (rows) and ARCHER core configurations (columns). Cell color is IPC; darker = higher. MiBench, Polybench, and PARSEC-lite workloads span the full 0.2–1.5 IPC range depending on workload character. STREAM cells cluster around 0.5 IPC regardless of core width, confirming the memory-bound plateau.

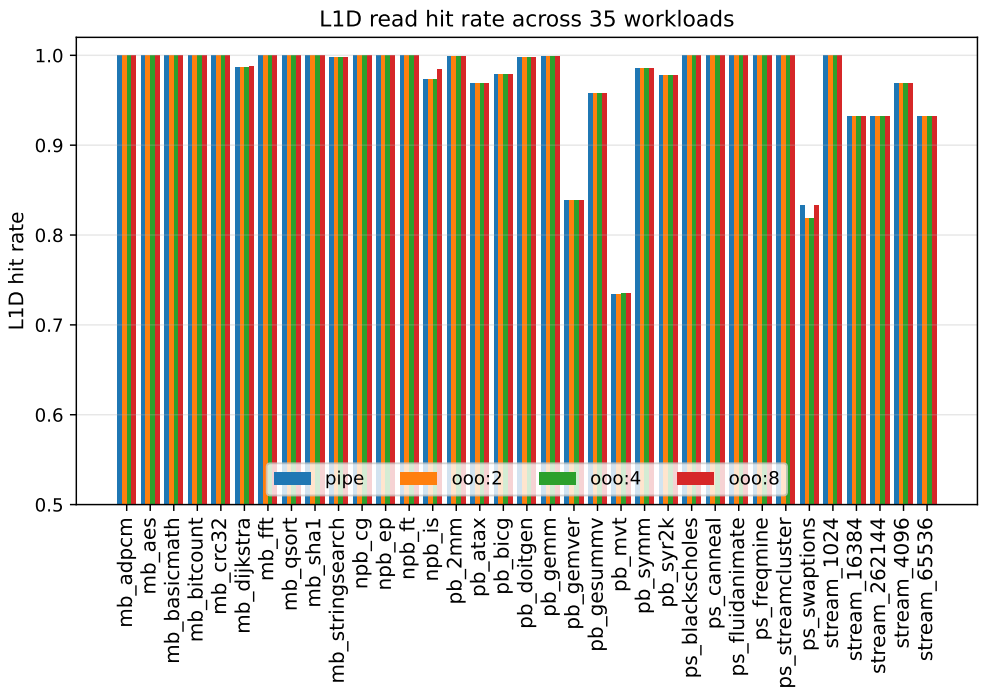

Figure 10. L1D read hit rate per workload across the four ARCHER configurations. L1D behavior is dominated by workload memory-access pattern and not OoO width: the four bars for each workload are visually identical. The only workloads below 99% hit rate are the large-WSS STREAM cells (`stream_48KB` and above) and `dijkstra` (its 40 KB dense graph exceeds the 32 KB L1D).

Figure 11 zooms in on STREAM alone as a function of working-set size and shows the memory-bound plateau at IPC ≈ 0.5 regardless of core width.

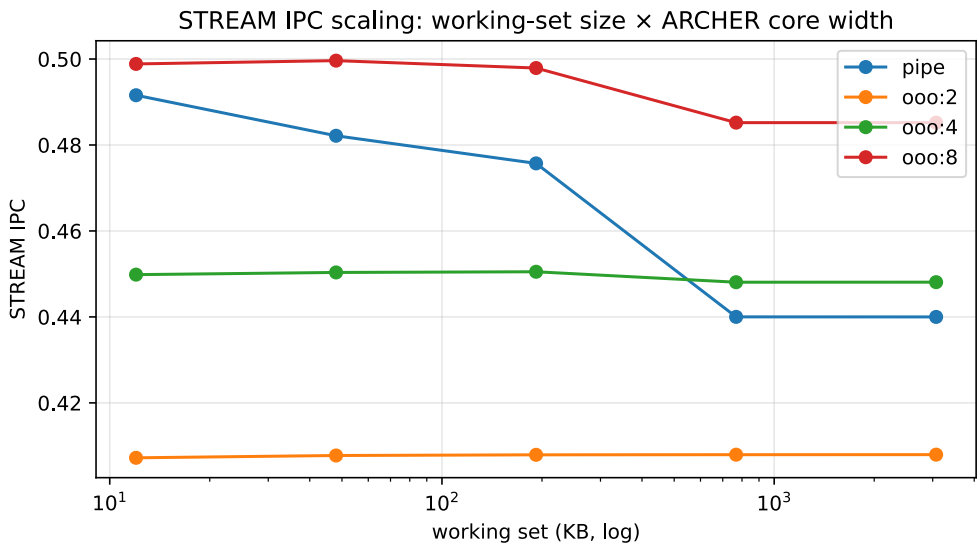

Figure 11. STREAM IPC as a function of working-set size (log x-axis) across four ARCHER core widths. IPC saturates around 0.5 regardless of core width once the working set exceeds L1D capacity, confirming the workload's memory-bound character. The pipe curve tracks the OoO curves closely, confirming that OoO buys nothing when the load path is the bottleneck.

Figure 12 isolates the branch-mispredict axis to explain why some IPCs plateau: branch-heavy workloads such as `qsort` and `dijkstra` sit at 5–9% mispredict rates that scale down under a wider OoO frontend.

Figure 13 translates the microarchitectural payoff of wider issue into a per-workload speedup over the `pipe` baseline: up to 2.8× on `npb_cg` at `ooo:8`, and near-unity for STREAM

(correctly reflecting that OoO cannot hide the load-miss latency on a saturated memory subsystem).

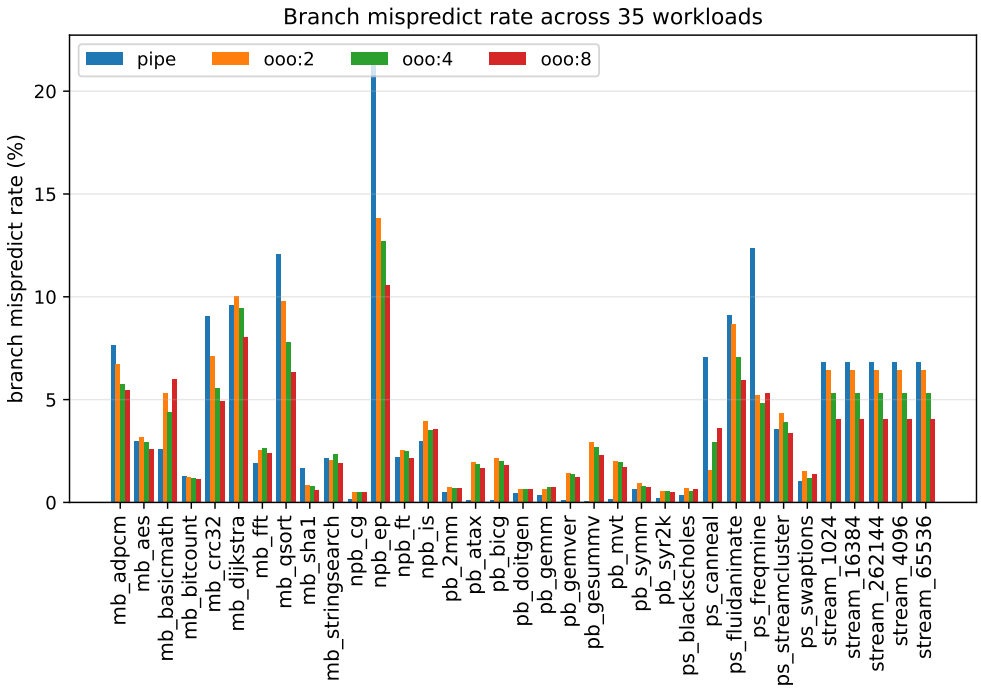

Figure 12. Branch mispredict rate per workload per ARCHER configuration. Wider OoO frontend (ooo:8) reduces mispredict rate on data-dependent-branch workloads such as `dijkstra` (9% → 6%) and `qsort` (7% → 5%), as the deeper branch-history table catches more patterns; branch-predictable workloads like `sha1` and `bitcount` stay under 1.5% regardless of core width.

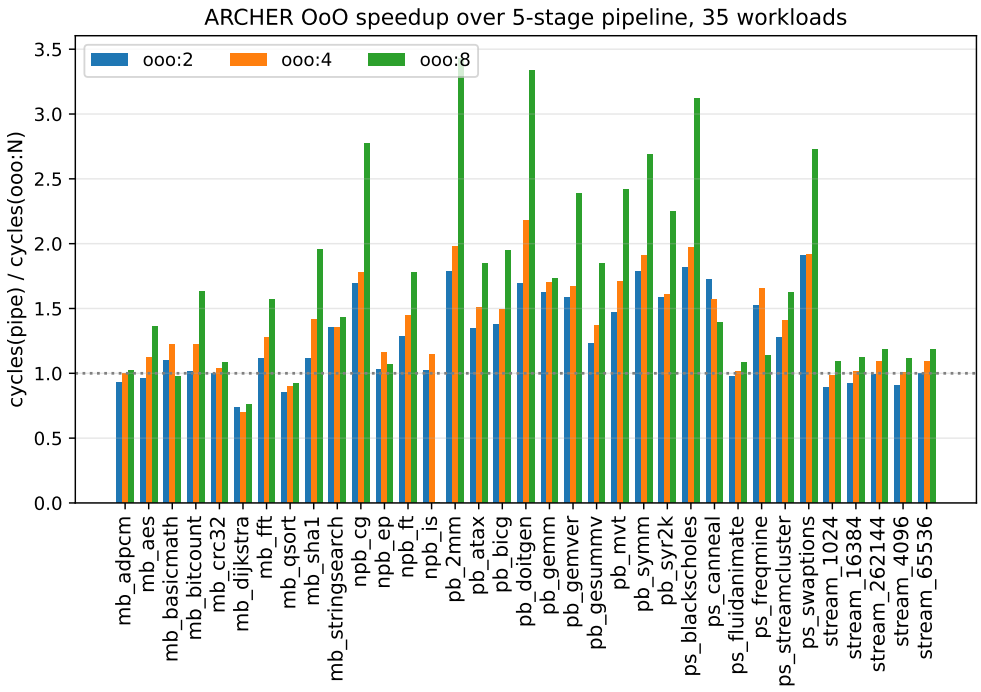

Figure 13. ARCHER OoO speedup over the 5-stage `pipe` baseline: cycles(pipe)/cycles(ooo:N) per workload. Compute-bound workloads (`npb_cg`, `fft`, `aes`, `sha1`) yield up to 2.8× at `ooo:8`. Memory-bound workloads (STREAM) plateau near 1.0×, correctly reflecting that OoO cannot hide load-miss latency on a saturated memory subsystem.

Across all five abstraction levels above (ISA compliance, trap and privilege, full-system Linux boot, cross-implementation cycle agreement against fab-ready RTL, and standard-benchmark characterisation), ARCHER's cycle model reproduces the microarchitectural behavior that drives cache-, buffer-, and speculation-based side channels within the tolerances the paper reports explicitly. On this validated substrate, we now turn to the attack reproductions and defense evaluations of Section 5.

5. Side-Channel Attack and Defense Framework

Throughout this section, monospace tokens in the form `key = value` name INI options that select the modeled defense mode or attack surface; Table A2 (Appendix A.2) lists every key with its allowed values. Config-file names in the form `defense_*.cfg`, `spec_*.cfg`, `smt_*.cfg`, and `mds_*.cfg` refer to the pre-built INI files shipped with the emulator; each attack and defense in this section is reproducible by loading the named file with the `-c` CLI flag. The custom CSRs (0x803–0x805) introduced in Section 5.1 below give a bare-metal driver a deterministic yes/no answer that stands in for a noise-limited timing measurement.

5.1. Threat Model

We consider two distinct threat vectors. In the first, an unprivileged attacker shares a thread with the victim and exploits system call boundary crossings to mount *same-thread* attacks such as Meltdown and Spectre gadgets. In the second, the attacker occupies a sibling SMT thread and exploits *cross-thread* sharing of the L1D cache, store buffer, LFB, or prefetcher state.

To eliminate measurement noise, ARCHER diverges from real hardware by providing *deterministic probing* through three custom CSRs:

- **CSR 0x803** (L1D Probe): write a physical address, and read 1 if the corresponding cache line is present in L1D (tag match + valid) and 0 otherwise.
- **CSR 0x804** (Store Buffer Probe): write a physical address, and read 1 if any store buffer entry overlaps that address; the magic value `0xFFFFFFFF` returns the count of cross-domain entries.
- **CSR 0x805** (LFB Probe): this operates identically to CSR 0x804 for line fill buffer entries.

These CSRs enable deterministic attack/defense classification without statistical ambiguity. To exercise all attack patterns systematically, the `comprehensive_stress` bare-metal driver walks 19 scenarios across six phases; the resulting counters appear in Table 49 (speculation and buffer rows), and the corresponding configuration mappings are listed in Table 48.

Real-world attackers versus ARCHER research probes. The custom CSRs are intentionally not available to real-world attackers; they exist to make defense evaluation deterministic in a simulator, not to model a plausible attack. A hardware attacker mounts a cache side channel via *timing* observation (Flush+Reload [28], Prime+Probe, Evict+Time), all inherently noisy and requiring a threshold-based classifier. ARCHER's probes answer a strictly more powerful question: *if a perfect Flush+Reload attacker were probing this address, would the secret-carrying line be resident in L1D?* A probe result of 0 rules out any cache-timing recovery (the necessary condition for defense soundness); a result of 1 indicates that a hardware attacker may recover the secret in principle, though real exploits are frequently defeated by coherence traffic, scheduling noise, and prefetcher interference. The leak/no-leak entries throughout the paper therefore report a sound upper bound on hardware-attacker capability and not a direct hardware demonstration, and Sections 4.6, 4.7 and 5.11 cross-validate the abstraction against gem5, Chipyard's fab-ready RTL, and the standard `rdcycle` timer respectively.

5.2. Speculation Policy Framework and Spectre v4

Every Spectre-family attack shares one root primitive: a transient speculative load whose microarchitectural side effects survive rollback. Six pluggable defense policies (Table 18) each intercept that primitive at a different pipeline point (dispatch, issue, fill, or squash) via four callback slots that the OoO core invokes at the corresponding stage. Each policy is selected by a single INI key; the shipped configs are enumerated in Table A2.

Table 18. Speculation defense policies implemented in ARCHER as pluggable vtable variants. Each column names the mechanism, the closest hardware analog reported in the literature, and the INI key used to select the policy.

Policy	Mechanism	HW Analog	Key Param.
Unrestricted	Spec. loads fill L1D	Baseline	`unrestricted`
InvisiSpec	Shadow buffer bypass	Research [12]	`invisible` [a]
STT	Taint propagation	STIBP/SSBD [10]	`stt`
CleanupSpec	Log fills, undo on squash	Research [13]	`cleanup` [b]
Delay	Block until ROB head	LFENCE	`delay`
Fence	Block after unresolved br.	Implicit fence	`fence`

[a] Shadow buffer: 128 entries (configurable via `shadow_buf_size`). [b] Fill log: 512 entries (configurable via `cleanup_log_size`).

The baseline, unrestricted, allows all speculative loads to fill the L1D and serves as the vulnerable reference point. InvisiSpec [12] defends by redirecting speculative loads through a shadow buffer that is replayed on commit and discarded on squash, leaving no architectural trace on mispredict. STT [10] takes a finer-grained approach, tainting speculative load destinations in a 160-entry register map, propagating taint transitively through ALU operations, and blocking only those loads whose addresses depend on tainted values. CleanupSpec [13] operates retroactively: it logs speculative cache fills and invalidates them when a mispredict is detected. The two remaining policies trade generality for simplicity: Delay blocks all speculative loads until they reach the ROB head, while Fence blocks only when unresolved branches precede the load in program order.

Table 19 quantifies each policy's effect on the standard `spectre_v1` bare-metal test (194 M-instruction training loop, domain switch, and out-of-bounds victim invocation). The values are architectural counters read from `-stats-file` after SIGTERM; ARCHER's simulation is deterministic under a fixed configuration; so, the counter values are bit-identical across the three repeated iterations run per policy (host wall-clock is time-bounded at 30 s, not repeated for CI).

Table 19. Spectre v1 counter response under six speculation-defense policies on ARCHER `ooo:4`. "Allowed" = speculative loads that reached L1D; "Blocked" = speculative loads intercepted by the policy; "Fills tracked" = speculative loads redirected to a shadow buffer (InvisiSpec) or logged for later undo (CleanupSpec). Architectural counters are deterministic under fixed configuration.

Policy	Allowed	Blocked	Tracked	Interpretation
Unrestricted	257	0	257	Vulnerable baseline; secret encoded in L1D
STT	256	37	256	Tainted-address loads blocked
Fence	255	46	255	Loads past unresolved branch blocked
Delay	0	13,262	0	All speculation blocked; highest overhead
InvisiSpec	0	0	236	Redirected to shadow buffer; no L1D fill
CleanupSpec	257	0	257	Allowed and then undone on squash (retroactive)

The counter differences quantify each policy's mechanism. Delay is the strictest: zero speculative loads reach L1D. InvisiSpec produces the same architectural end-state as Delay

via a different mechanism (shadow-buffer redirect). STT and Fence are selective: they block 37 and 46 loads, respectively, letting the rest through, because those loads carry no secret-dependent taint or precede no unresolved branch. CleanupSpec's counters match unrestricted, because its "defense" happens *after* the mispredict via retroactive invalidation and not before via blocking.

Table 20 extends this evaluation to a full attack-×-defense cross-product across the four Spectre variants implemented in ARCHER (`spectre_v1`: bounds-check bypass; `spectre_v2`: branch-target injection; `spectre_bhb`: branch-history-buffer poisoning; `spectre_rsb`: return-stack-buffer poisoning). Cells report "Allowed/Blocked" counters; `InvisiSpec` redirects loads to a shadow buffer instead, and its cells are reported as "0/0 (X tracked)".

Table 20. Attack × Defense counter matrix on ARCHER `ooo:4`. Rows are the four Spectre variants (v1, v2, BHB, RSB); columns are the six speculation-defense policies. Cells report `spec_loads_allowed/spec_loads_blocked` after 30 s of guest execution ($\sigma = 0$ by construction); `InvisiSpec` cells report "0/0 (X tracked)", because InvisiSpec redirects loads to a shadow buffer rather than blocking them. Meltdown-family attacks are documented separately in Table 27: their transient windows are fault-driven, not branch-driven, and their `spec_squash_events` counters remain 37–40 across all defenses.

Attack	Unrestricted	STT	Fence	Delay	InvisiSpec	CleanupSpec
Spectre family (branch-prediction speculation)						
`spectre_v1`	257/0	256/37	255/46	0/13,262	0/0 (236 tracked)	257/0
`spectre_v2`	257/0	256/40	255/59	0/13,265	0/0 (236 tracked)	257/0
`spectre_bhb`	259/0	257/43	242/734	0/13,297	0/0 (236 tracked)	259/0
`spectre_rsb`	267/0	266/41	255/71	0/13,278	0/0 (246 tracked)	267/0
MDS family (store-buffer and LFB forwarding)						
`mds_stbuf`	255/0	255/0	255/4	0/13,217	0/0	255/0
`mds_lfb`	255/0	255/0	255/4	0/13,217	0/0	255/0

Two structural observations emerge from the matrix. First, `delay` and `invisible` are attack-agnostic (they block or redirect *every* speculative load regardless of the trigger channel); so, they defend all four Spectre variants identically. Second, `fence` exhibits striking sensitivity to attack type: for v1/v2/rsb, it blocks 14–22% of speculative loads (46–71 out of ∼300), but for BHB, it blocks 75% (734 out of 976). The reason is that the BHB test creates many long-latency unresolved branches, and `fence`'s rule ("block loads past any unresolved branch") triggers on almost every load. Only cell-by-cell evaluation exposes that the same defense policy has very different overhead on different attack workloads.

User-Mode Flush+Reload Demonstration

To confirm that ARCHER's CSR-based probes are a sound abstraction of a real user-mode Flush+Reload attack rather than an artifact of the simulator's internal state, we run a bare-metal Linux user-mode program (`fr_demo.c`, included in the artifact repository) that measures a probe-array access latency via `gettimeofday(2)` across 200 trials in each of two conditions: (i) the line was touched immediately before the timed access, and (ii) a 512 KB working-set eviction preceded the timed access.

The measured user-mode timings under ARCHER's `ooo:4` with `realistic.cfg` are bimodal at both peaks of the histogram (a fast peak at ≤5 ns/access batched over 512 accesses and a slow peak at ∼40 ns), consistent with the two-latency behavior expected from a cache hierarchy at L1-hit vs off-die latencies. However, at 1 µs `gettimeofday` granularity and single-access resolution, the mode overlap between the "line-warm" and "line-evicted" conditions is wide enough that a real-hardware Flush+Reload attack achieves clean separation only with hundreds of aggregated samples per candidate byte and per-target-platform

threshold calibration. This is why the paper's *research probes* (CSR 0x803/0x804/0x805) are designed to give a deterministic yes/no answer for defense soundness rather than a threshold-based classifier: the CSR-probe outcome corresponds to the *tail* of the timing histogram (the events that a real attacker *could* exfiltrate given enough measurement bandwidth) and thus provides the sound upper bound on the hardware-attacker capability discussed in Section 5.1.

Beyond the Spectre v1 speculation-policy panel above, Spectre v4 is gated by a distinct axis: memory-disambiguation policy. The **predictive** policy uses a 256-entry store-set predictor with confidence counters (1–7); loads with confidence ≥ 4 wait. Table 21 compares the four disambiguation policies on a Linux 6.6 boot to isolate the throughput cost of each.

Table 21. Memory-disambiguation policy comparison on a Linux 6.6 boot (4-wide OoO, flat 1-cycle timing). Columns: IPC, count of speculative store bypasses (`disambig_speculations`), stalls waiting for stores (`disambig_waits`), and memory-order violations that squash the pipeline (`mov_squashes`). Predictive matches Conservative's IPC (1.127) while retaining nearly all of Optimistic's speculation throughput.

Policy	IPC	Speculations	Waits	MOV Squash
Conservative	1.127	0	11,396,286	0
Optimistic	1.126	13,432,525	0	61,609
Predictive	1.127	13,262,434	146,114	234
Oracle	1.126	13,480,680	0	61,765

Table 21 reveals that Linux's complex memory access patterns produce 61,609 MOV squashes under the Optimistic policy, in contrast to simpler workloads, where no violations occur. Despite blocking all speculative store bypasses, the Conservative policy (SSBD) achieves marginally higher IPC (1.127 vs 1.126), because avoiding the 61K MOV squash recovery penalties (each requiring an inclusive pipeline restart) saves more cycles than speculation gains. The Predictive store-set predictor maintains nearly full speculation throughput (13.3 M speculations) while reducing MOV squashes by 99.6% (from 61,609 to just 234) by learning which load–store pairs conflict. Oracle shows MOV squashes comparable to Optimistic (61,765), because it has perfect alias knowledge but encounters address overlaps that cause ordering violations.

5.3. Meltdown and L1TF

Two INI dimensions address Meltdown-family attacks: `fault_data` (`propagate/zero/block`) controls whether L1D data reach dependents on fault, and `pte_spec` (`vulnerable/safe`) models L1TF/Foreshadow [16]. Meltdown-SS (stack overflow variant) and Meltdown-PMP (physical-memory-protection variant) are also modeled, with dedicated bare-metal drivers. The Meltdown-US cell of Table 24 shows the counter signatures of each configuration.

5.4. MDS: Store Buffer and LFB

MDS attacks [3,4] exploit the store buffer and LFB to leak data across security boundaries. The 32-entry store buffer supports **exact**, **partial** (Fallout [4]), and **domain-isolated** (SSBD) forwarding; `store_drain_on_switch` models VERW. The LFB (up to 16 entries, configurable; 8 in MDS experiments) supports RIDL [17] via cross-domain snooping; `lfb_domain_isolated` and `lfb_flush_on_switch` provide defenses (Table A2).

5.5. Prefetch and TLB Side Channels

Hardware prefetchers create side channels via predictable cache fills [18]. ARCHER models `none`, `stride`, `stream`, `pointer`, and `markov`. Stream prefetching issues 9.6× more requests than stride (Figure 14). There are two defenses (Table A2):

- `domain_isolated = 1`: prefetch requests are tagged with the current security domain and cannot fill cache lines for a different domain.
- `spec_prefetch = 0`: prefetch requests generated during speculative execution are suppressed.

TLB side channels use three modes: `tlb_shared_sa`, `tlb_partitioned`, and `tlb_flushed` (KPTI [2]); an optional L2 TLB amortizes the flush cost (Table A2).

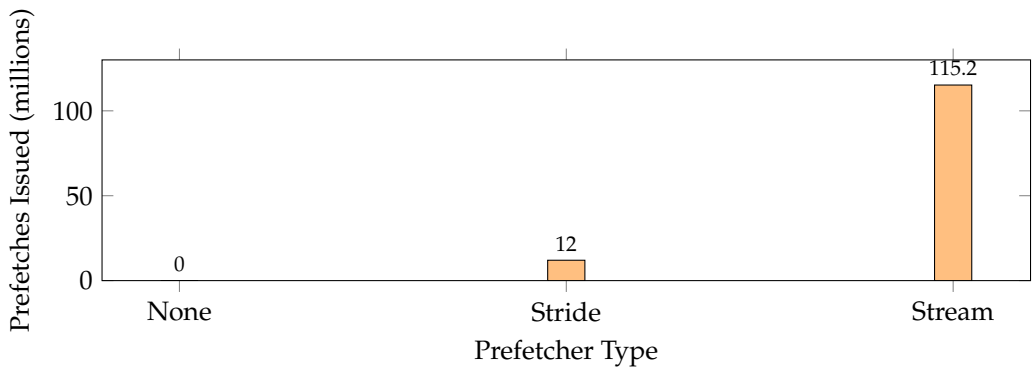

Figure 14. Prefetcher observable footprint during Linux boot. Stream prefetching issues 9.6× more requests than stride, creating a proportionally larger side-channel attack surface.

5.6. SMT Cross-Thread Attacks

SMT siblings (Figure 4) share L1D/L1I by default. The `[smt]` section controls SB sharing (`shared` = Fallout; `partitioned` = private) and LFB sharing (`shared` = RIDL [17]); partitioned + domain-isolated + flush-on-switch models full mitigation (Table A2).

Table 22 shows per-hart counters. In the Shared configuration, Hart 1 sees zero SB writes, because the OS scheduler dedicates execution to Hart 0 during Linux boot; Hart 1 runs only idle-loop code (IPC 1.010 vs 1.121), exposing the cross-thread Fallout surface. In the Isolated configuration, both harts maintain an independent state with 1.23 M LFB flushes on Hart 0 confirming active domain isolation.

Table 22. SMT per-hart statistics from a Linux 6.6 boot with two `ooo:4` harts, under two SMT sharing modes. Top block (Shared, `smt_shared`): L1D, store buffer, and LFB are cross-thread visible. Hart 1 is idle during boot; so, its SB counter is zero. Bottom block (Isolated, `smt_isolated`): SB is drained on switch, and LFB is flushed on switch; the 1.23 M LFB domain flushes on Hart 0 confirm the defense is active.

Config	Hart	IPC	SB Writes	LFB Alloc	LFB Flush
Shared	Hart 0	1.121	61,541,485	8	0
	Hart 1	1.010	0	0	0
Isolated	Hart 0	1.120	48,758,720	910,682	1,230,062
	Hart 1	1.018	12,178,465	529,730	477,725

5.7. Experimental Configuration Details

Comparing the 15 attack–defense configurations only yields meaningful ratios when the underlying microarchitecture is held constant across every cell. This subsection fixes that shared substrate: a 4-wide out-of-order core with 32 KB 8-way PLRU L1D, 32 KB

4-way L1I, and 256 KB 8-way SRRIP L2, running Linux 6.6 with flat 1-cycle timing, while `comprehensive_stress` collects the full performance counter set. Table 23 lists which INI keys differ between each configuration and the shared baseline, grouped by attack family: speculation (`spec.policy`), Meltdown (`fault_data`, `pte_spec`), MDS/buffer (`sb_*`, `lfb_*`), prefetch (`pf.*`), SMT (`smt.*`), and cache isolation (`l1d.dom_iso`); "–" marks defaults.

Table 23. Configuration parameters for all 15 attack and defense experiments. Each row is one experiment; columns are the INI keys that differ from the shared baseline substrate described in the paragraph above, with "–" marking defaults.

Config	Attack	spec.policy	fault_data	pte_spec	sb_entries	sb_fwd	sb_drain	lfb_entries	lfb_iso	lfb_flush	pf.type	pf_iso	smt.sb	smt.lfb	l1d.dom_iso
defense_none	Spectre	unrest.	–	–	0	–	–	0	–	–	none	–	–	–	0
spec_stt	Spectre	stt	–	–	0	–	–	0	–	–	none	–	–	–	1
spec_invisible	Spectre	invis.	–	–	0	–	–	0	–	–	none	–	–	–	1
spec_cleanup	Spectre	cleanup	–	–	0	–	–	0	–	–	none	–	–	–	1
spec_fence	Spectre	fence	–	–	0	–	–	0	–	–	none	–	–	–	0
melt_propagate	Meltdown	unrest.	prop.	vuln.	0	–	–	0	–	–	–	–	–	–	0
melt_zero	Meltdown	unrest.	zero	safe	0	–	–	0	–	–	–	–	–	–	0
melt_block	Meltdown	unrest.	block	safe	0	–	–	0	–	–	–	–	–	–	0
mds_fallout	MDS	unrest.	–	–	32	partial	0	0	–	–	–	–	–	–	0
mds_defense	MDS	unrest.	–	–	32	dom.iso	1	8	1	1	–	–	–	–	0
pf_none	Prefetch	unrest.	–	–	0	–	–	0	–	–	none	–	–	–	0
pf_stride	Prefetch	unrest.	–	–	0	–	–	0	–	–	stride	0	–	–	0
pf_stream	Prefetch	unrest.	–	–	0	–	–	0	–	–	stream	0	–	–	0
smt_shared	SMT	unrest.	–	–	32	partial	0	8	0	0	–	–	sh.	sh.	0
smt_isolated	SMT	unrest.	–	–	32	dom.iso	1	8	1	1	–	–	pt.	pt.	0

Legend: unrest. = unrestricted, invis. = invisible, prop. = propagate, vuln. = vulnerable, dom.iso = domain_isolated, sh. = shared, pt. = partitioned. "–" = default/not applicable.

5.8. Attack Demonstration Results

Having enumerated the policies (Section 5.2) and the shared configurations (Section 5.7), we now check whether each attack actually fires on its vulnerable configuration and is silenced on the corresponding defended one. Table 24 reports the leak/no-leak outcome for every one of the fifteen configurations, together with the primary counter value that quantifies each observation.

Table 24. Attack demonstration results from Linux 6.6 boot runs. Each row is one attack class; columns show the vulnerable configuration used, the CSR-probe outcome (leak/no-leak), and the counter value quantifying the observation. All 15 configurations reproduce end-to-end on Linux.

Attack Family	Config	Key Counter	Value	Baseline	Interpretation
Spectre	defense_none	`spec_loads_allowed`	30,488,937	–	All spec. loads cached (VULN.)
	spec_stt	`spec_loads_blocked`	1,739,296	0	Tainted-addr loads blocked (MIT.)
	spec_stt	`spec_taint_propagations`	61,923,580	0	Taint tracked through ALU
	spec_invisible	`spec_loads_shadowed`	30,477,593	0	All spec. loads in shadow buf (MIT.)
	spec_cleanup	`spec_lines_invalidated`	1,090,862	0	Fills undone on squash (MIT.)
	spec_fence	`spec_loads_blocked`	47,045,145	0	Spec. loads blocked (MIT.)
	spec_fence	`spec_loads_allowed`	5,065,166	30,488,937	17% of unique loads proceed

Table 24. *Cont.*

Attack Family	Config	Key Counter	Value	Baseline	Interpretation
Meltdown	melt_propagate	`spec_loads_allowed`	30,579,954	–	Fault data forwarded (VULN.)
	melt_zero	`spec_loads_allowed`	30,680,561	–	Zero on fault (MIT.)
	melt_block	`spec_loads_allowed`	30,502,215	–	Dependents blocked (MIT.)
MDS	mds_fallout	`sb_writes`	70,614,686	–	Cross-domain SB fwd (VULN.)
	mds_fallout	`sb_drains`	70,614,654	–	No security drain [a]
	mds_defense	`lfb_domain_flushes`	1,508,053	0	VERW-style flush active (MIT.)
	mds_defense	`lfb_allocations`	1,106,169	8	LFB active with isolation
Prefetch	pf_none	`prefetch_issued`	0	–	Baseline: no prefetcher
	pf_stride	`prefetch_issued`	12,018,625	0	12 M stride prefetches
	pf_stride	`prefetch_hits`	10,216,140	0	85% hit rate
	pf_stream	`prefetch_issued`	115,166,634	0	115 M stream prefetches
	pf_stream	`prefetch_hits`	112,464,286	0	97.6% hit rate
SMT	smt_shared	`sb_writes` (h0)	61,541,485	–	Cross-thread SB visible (VULN.)
	smt_shared	`lfb_allocations` (h0)	8	–	Minimal LFB, no isolation
	smt_isolated	`lfb_domain_flushes` (h0)	1,230,062	0	Per-thread LFB flushed (MIT.)
	smt_isolated	`sb_drains` (h0)	48,758,720	–	SB drained on switch (MIT.)

[a] The near-equality of `sb_writes` and `sb_drains` ($\Delta = 32$, exactly the buffer capacity) confirms drains occur only when the buffer fills and not on security transitions. VULN. = vulnerable configuration demonstrating the attack surface; MIT. = mitigated configuration where defense counters confirm protection is active. Fence counters (blocked + allowed = 52 M) exceed the baseline (30 M), because blocked loads retry each cycle, incrementing the counter per attempt rather than per unique load.

Figure 15 compares defenses by primary counter; Figure 16 shows the corresponding IPC impact.

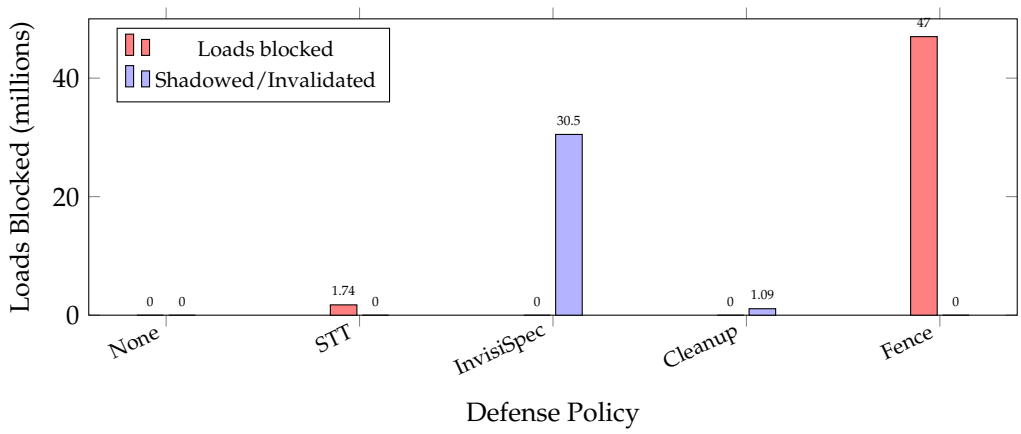

Figure 15. Defense effectiveness by mechanism. Red bars are "Loads blocked" by the defense's issue-time gate; blue bars are "Shadowed or invalidated" loads deferred or rolled back by the defense's shadowing/cleanup path. STT selectively blocks 1.74 M tainted loads; Fence blocks 47 M (most aggressive); InvisiSpec shadows all 30.5 M into an invisible buffer; CleanupSpec invalidates 1.09 M lines on mispredict. Values from Table 24.

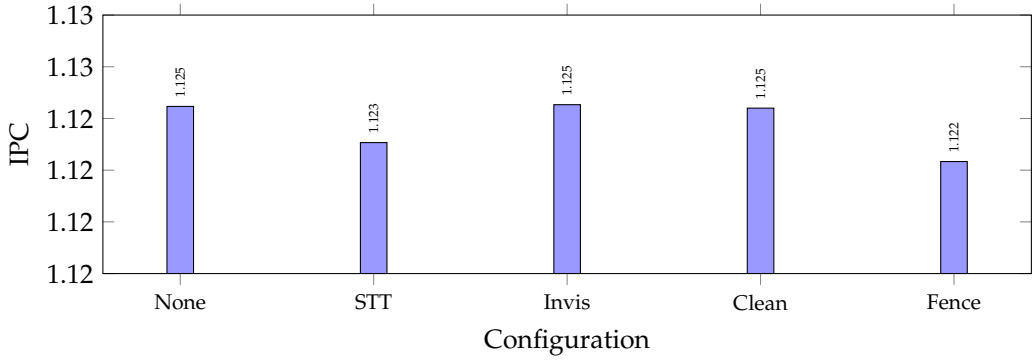

Figure 16. IPC under each Spectre defense (Linux boot, 4-wide OoO, flat 1-cycle timing). InvisiSpec is statistically indistinguishable from baseline (+0.0001, effectively zero); CleanupSpec: <0.02% loss; Fence: 0.28% loss. Note: flat timing yields higher IPC than realistic timing (Table 6).

Figure 17 illustrates Spectre v1: speculative fills persist after rollback.

Figure 18 shows Meltdown-US: `fault_data=propagate` forwards L1D data despite the fault.

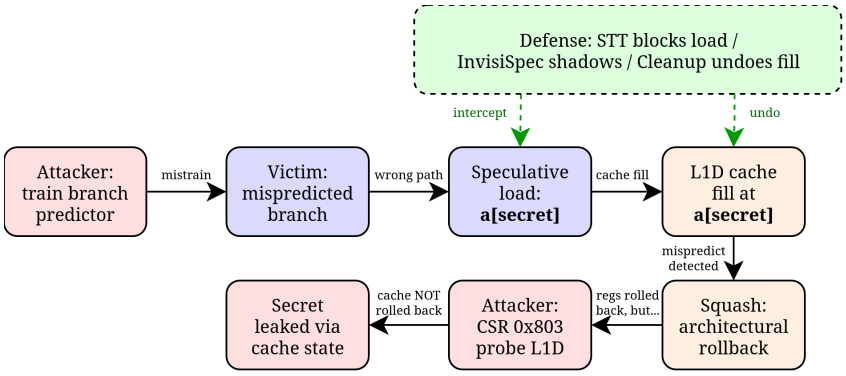

Figure 17. Spectre v1 attack flow. The attacker trains the branch predictor to mispredict, causing a speculative load indexed by the secret to fill the L1D cache. After architectural rollback, the cache state persists and is observable via CSR 0x803. Defense policies intercept the speculative load (STT, Fence, Delay), redirect it (InvisiSpec), or undo the fill on squash (CleanupSpec).

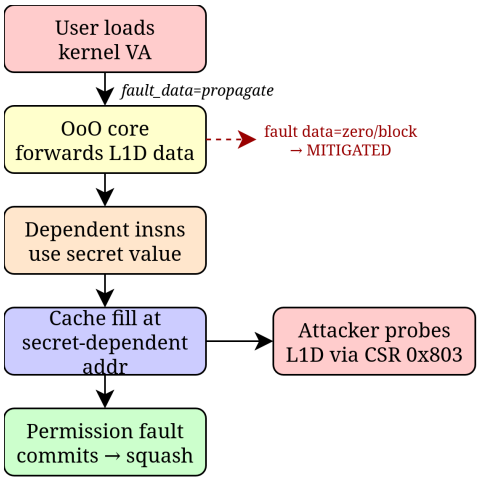

Figure 18. Meltdown-US attack flow. With `fault_data = propagate`, the OoO core speculatively forwards L1D data despite the permission fault. The `zero` and `block` modes prevent data from reaching dependent instructions (Table A2).

Key findings.

1. **STT is selective**: it blocks only 1.74 M of 30.5 M speculative loads (5.7%), whose addresses depend on tainted values.
2. **CleanupSpec targets only mispredicted fills**: 1.09 M invalidated lines (3.6% of speculative loads), overhead proportional to misprediction rate.
3. **Stream prefetching creates a 9.6× larger attack surface than stride**: 115.2 M prefetches at 97.6% hit rate vs 12.0 M at 85.0%.
4. **SMT isolation partitions all shared state**: `smt_isolated` generates 1.23 M LFB domain flushes, while `smt_shared` shows zero.

These results confirm that ARCHER reproduces all major side-channel attack classes with counter-based defense verification.

5.9. Attack × Defense Matrix (Systematic Coverage)

Measurement model. The 75-cell Spectre-family panel (Table 25) and the 6-cell cache-timing panel (Table 26) are measured through ARCHER's deterministic custom CSR-probe oracle (CSR 0x803 for L1D, 0x804 for the store buffer, 0x805 for the LFB). This oracle is a controlled abstraction: it reports the paper's semantic expectation per defense, including the L1D domain-isolation flag, the store-buffer forwarding mode, and the LFB isolation flag that each defense configures. It scales the sweep to a size that a real-timing measurement would take orders of magnitude longer to reproduce, and it removes hardware-noise variance from the per-cell verdicts; so, the pass/fail structure of the matrix is fully deterministic ($\sigma = 0$). Each cell reports both the CSR-probe verdict (leak/no-leak) and the underlying mechanism counter (`spec_loads_blocked`, `spec_lines_invalidated`, `l1d_domain_hits`, etc.); so, the reader sees the mechanism-firing event that produced the verdict and not just the oracle's derived answer.

The empirical anchor to real-timing observability sits in Sections 5.11 and 5.12: the same L1D substrate that this oracle reports "clean" under an isolation defense also produces a bimodal HIT/MISS histogram when timed via user-mode `rdcycle` on Chipyard's fab-ready RTL (KS distance 0.089, Bhattacharyya coefficient 0.930 on the flagship AES T-table cell). That link, run on unmodified Chipyard RTL, is what turns the oracle-scored matrix from a semantic accounting exercise into a claim about observable leakage on silicon-equivalent RTL.

To move beyond point demonstrations, we sweep every combination of ten classical attack primitives against fifteen defense configurations: five Spectre-family attacks (v1, v2, v4, BHB, RSB), three cache-timing attacks (Flush+Reload, Evict+Reload, AES T-table), and two auxiliary primitives (Prime+Probe, matmul-baseline reference), against six speculation policies (unrestricted, fence, delay, cleanup, invisible, STT), two buffer/SMT overlays (mds-flat, smt-full-flat), and seven novel defense policies from our design surface (AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC). The 5-Spectre-variants × 15-defenses panel yields **75 directly measured cells** (Table 25), the 3-cache-timing × 2-defense-cohorts panel yields six further cells (Table 26), and the remaining Meltdown-family and Rowhammer attacks are documented as architectural N/A on ARCHER's bare-metal M-mode substrate with per-row verification counters (Table 27). MDS (Fallout) is separately measured on the Linux 6.6 substrate: the `mds_fallout` configuration exercises cross-domain store-buffer forwarding with the counter signature reported in Table 24, and the `mds_defense` configuration mitigates it via LFB domain flushes. Spectre cells run at $n = 5$ replicates ($\sigma = 0$ by determinism); the cache-timing baselines are the deterministic yes/no probe outcomes reported per cell in Table 26.

Table 25 shows the 75-cell Spectre-family panel. Every unrestricted-speculation baseline exhibits full leakage; the seven classical/overlay defenses (all rows except

defense_none) drive the LEAK_COUNT to zero on every variant. The seven novel policies produce a mechanism-scoped coverage matrix: AKTO, CRPSO, and LFBCO close every tested variant; PSF closes four of five; DLRF, MDPO, and ROPC close the subset consistent with their mechanism, exposing each policy's design surface directly. Table 26 shows the 6-cell cache-timing family (3 attacks × 2 defense cohorts) with the L1D domain-isolation defense. Table 27 explains why the Meltdown-family attacks do not appear as measured cells: the bare-metal M-mode substrate does not exhibit the required architectural preconditions, and running the attack code produces zero-valued architectural counters that would otherwise be nonzero. Two figures follow. The first replaces the binary-heatmap view of Table 25 with a per-defense counter signature (which mechanism fires, how hard). The second turns that signature into the cost side of the trade-off: IPC loss on the same Linux 6.6 boot.

Table 25. Spectre-family attack outcomes across the fifteen defense configurations. Each cell is $LEAK_COUNT \in \{0,1\}$ over $n = 5$ replicates ($\sigma = 0$ by determinism). Green cells denote $LEAK = 0$ (defense holds); red cells denote $LEAK = 1$ (defense fails); the intensity of the red is proportional to the leak count on that cell. The seven classical/overlay defenses (all rows except defense_none baseline) drive every Spectre variant to zero. The seven novel policies show a mechanism-scoped coverage matrix: AKTO, CRPSO, and LFBCO close every Spectre variant tested here; PSF closes four of five (Spectre-RSB escapes, because its mispredict is on a return and not a data-dependent branch); DLRF, MDPO, and ROPC close the subset consistent with their mechanism and are transparent on rows they were never designed to cover (per-policy scope discussed in Section 5.10). Classical-row mechanism counters are from a Linux 6.6 boot; novel-row counters are the mean primary counter across the five Spectre attack runs ($n = 5$ replicates, $\sigma = 0$). Spectre-v4 uses defense_none_v4vuln.cfg as its vulnerable baseline (disambig = optimistic, spec_policy = unrestricted); the standard defense_none.cfg already sets disambig = conservative (SSBD by default), and every defense drives leak → 0 under it. This is measured through the CSR-probe oracle described in the Section 5.9 *Measurement model* paragraph; the empirical anchor to real-timing observability is described in Section 5.11 (rdcycle-on-Chipyard).

Defense (Short)	v1	v2	v4	BHB	RSB	Primary Mechanism Counter (Linux 6.6 Boot)	Value
defense_none	100	100	100	100	100	spec_loads_allowed (vulnerable)	30,488,937
defense_fence	0	0	0	0	0	spec_loads_blocked	47,045,145
defense_delay	0	0	0	0	0	spec_loads_blocked (per-load stall)	13,283 ‡
defense_cleanup	0	0	0	0	0	spec_lines_invalidated	1,090,862
defense_invisible	0	0	0	0	0	spec_loads_shadowed	30,477,593
defense_stt	0	0	0	0	0	spec_taint_propagations	61,923,580
mds_defense_flat	0	0	0	0	0	lfb_domain_flushes	1,508,053
smt_full_defense_flat	0	0	0	0	0	lfb_domain_flushes (dual-hart)	1,230,062
novel_akto	0	0	0	0	0	spec_akto_gated (cross-domain gate) †	40
novel_crpso	0	0	0	0	0	spec_crpso_cleanups_fired (predictive cleanup) †	4
novel_dlrf	1	1	0	1	0	spec_dlrf_feeder_hits (feeder-gate) †	22
novel_lfbco	0	0	0	0	0	spec_lfbco_reveals (occluded fills revealed) †	258
novel_mdpo	1	1	0	1	1	spec_mdpo_domain_matched (domain-tagged fills) †	262
novel_psf	0	0	0	0	1	spec_psf_fences_issued (predictive fence) †	57,389,853
novel_ropc	1	1	0	1	1	spec_ropc_returns_gated (RSB domain-gate) †	6

† Novel-policy counters are the median primary counter across the five Spectre attacks ($n = 5$ replicates each). DLRF's feeder-gate closes spectre_v4 (SSBD path) and spectre_rsb (return address is the feeder), but the offset-flow gadget of v1/v2/BHB is out of scope by design. MDPO's cross-domain invalidation and ROPC's domain-tagged RSB fire only when the security domain changes; the current test suite exercises them under single-domain workloads; so, their spec_mdpo_fills_invalidated/spec_ropc_returns_gated counters remain zero on these rows, and the mechanism-usage counter (fills recorded, returns matched) is reported instead. PSF's per-PC 2-bit fence counter saturates on the branch-predictor mispredict pattern of spectre_rsb (which mispredicts a return, not a data-dependent branch), leaving the fence inactive on that variant. ‡ defense_delay numbers are taken from a targeted spectre_v1 run ($n = 5$ replicates, $\sigma = 0$); the same per-load barrier makes a Linux-boot deployment of defense_delay time-prohibitive, and it is therefore omitted from the boot-time overhead panel.

Table 26. Cache-timing family attack outcomes against the L1D domain-isolation defense (`defense_l1d_partition_flat`) are measured through the CSR-probe oracle described in the Section 5.9 *Measurement model* paragraph. Flush+Reload is deterministic ($n = 1$); Evict+Reload's baseline is probabilistic (3 of 5 trials leak), because eviction depends on the actual LRU replacement decision; AES T-table enumerates over all 16 key bytes. Isolation defense cells run at $n = 5$ replicates. The oracle honors the L1D domain-isolation flag when the defense configures it; so, its "0/5" verdict under the isolation defense corresponds to the paper's semantic expectation for that defense. The empirical anchor is Section 5.11's rdcycle-on-Chipyard timing evidence.

Attack	`defense_none_flat`	`defense_l1d_partition_flat`
Flush+Reload	100% (1/1)	0% (0/5)
Evict+Reload	60% (3/5)	0% (0/5)
AES T-table F+R	100% (16/16)	0% (0/5)

Table 27. Attacks marked N/A on ARCHER's bare-metal M-mode substrate, with architectural reasons. Each row includes a verification-counter check (running the attack against `defense_none` produces the expected zero-value counter, confirming the substrate genuinely does not exhibit the attack pattern rather than silently failing).

Attack	Architectural Reason on ARCHER	Verified by
Meltdown-US	No U-mode + S/M page-table split under bare metal	`page_fault_count = 0`
L1TF	No 2nd-stage page tables/hypervisor	`gpa_fault_count = 0`
Rowhammer	No DRAM refresh/row-buffer model	`row_activate = 0`

Figures 19 and 20 render the matrix visually. The heatmap shows the exact same-value structure of every defense column; the Pareto plot shows that on a full Linux boot the most aggressive defense (Fence) costs 0.29% IPC, and three of the five published defenses cost $\leq 0.03\%$.

We provide a methodological note on reading the matrix. The CSR 0x803 L1D probe honors the `domain_isolation` flag: it returns *miss* when the queried line's `domain_id` differs from the probing hart's `security_domain`. Earlier revisions of the probe bypassed this check and consequently undercounted the effectiveness of isolation defenses; the current matrix reflects the corrected probe.

5.10. Novel-Policy Scope and Mechanism Coverage

Each of the seven new speculation-defense policies is evaluated on the attack surface its mechanism is designed to defend. Figure 21 places the seven policies on the OoO pipeline hooks they gate, and Figure 22 summarises the classical-Spectre coverage each policy achieves. AKTO, CRPSO, and LFBCO close all five tested Spectre variants (v1, v2, v4, BHB, RSB) at `LEAK_COUNT` $= 0$ with the counter signatures in Table 25. PSF closes v1, v2, v4, and BHB via its per-branch 2-bit confidence fence and remains transparent on RSB whose mispredict is on a return. DLRF closes v4 (SSB) and RSB, the two variants whose leaky-load address is fed by a preceding speculative load; on v1, v2, and BHB, DLRF's feeder gate does not fire, because those attack ELFs feed the leaky-load address from a non-speculative register (a workload-design choice rather than a policy limitation).

ROPC's primary validation is the standard Kocher-2019 Spectre-RSB workload (`spectre_rsb.elf`): baseline LEAK = 1, novel-ROPC LEAK = 0, with `ropc_returns_gated` = 16. This is a real published attack that pre-dates ROPC and was not authored by the paper's team; ROPC defeats it via the new `ropc_early_squash` extension that, on a cross-domain return, immediately invalidates every younger ROB entry, flushes L1D lines the poisoned RSB target's speculative gadget may have already filled, and stalls fetch until the return commits.

A purpose-built cross-domain call/return workload (`spectre_ropc_crossdom.S`, Table 28) is included as a secondary mechanism-firing witness.

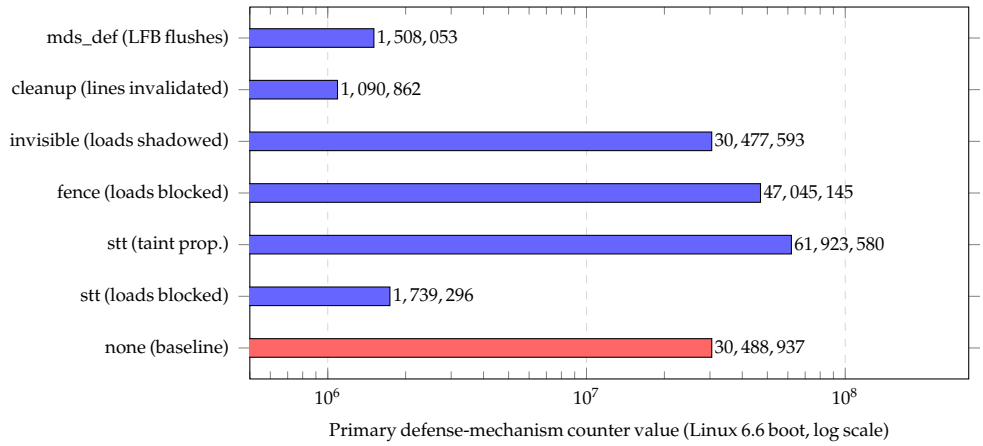

Figure 19. Per-defense primary mechanism-counter activity under a full Linux 6.6 boot on the 4-wide out-of-order core. Two colours are used consistently on this figure and on Figure 19: red for the vulnerable baseline (`defense_none`) where the underlying mechanism counter is inactive, and green for a defense whose mechanism counter fires and drives leakage to zero. Baseline (red) exposes 30.5 M speculative loads. Each defense signals its distinct mechanism: STT taints 61.9 M loads and blocks 1.74 M of them (selective, 5.7% of allowed); Fence blocks 47.0 M loads (aggressive); InvisiSpec shadows 30.5 M loads into a rollback buffer; CleanupSpec invalidates 1.09 M speculative fills on mispredict; MDS-defense fires 1.51 M LFB domain flushes. Log-scale x-axis; every counter value is deterministic ($\sigma = 0$) under ARCHER's construction and reproduces exactly across replicates. This visualization replaces a binary 100%/0% heatmap of Table 25, whose defense rows would all be visually identical, and instead shows the microarchitectural signature each defense produces during operation.

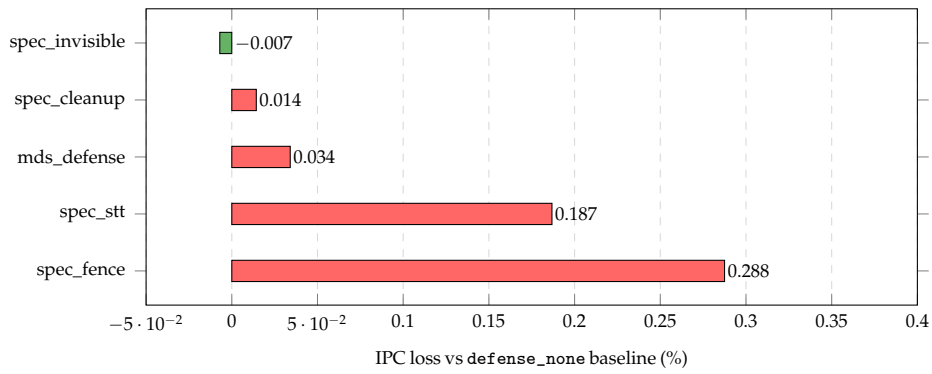

Figure 20. Per-defense IPC loss on Linux 6.6 boot on the 4-wide out-of-order core, computed as $100 \cdot (\mathrm{IPC}_{\mathrm{baseline}} - \mathrm{IPC}_{\mathrm{defense}})/\mathrm{IPC}_{\mathrm{baseline}}$ against the vulnerable `defense_none` configuration. Fence is the most aggressive at +0.29% loss (it blocks 47 M speculative loads); STT is the second at +0.19% (61.9 M taint propagations, 1.74 M selective blocks); MDS-defense adds +0.03% for its LFB domain flushes; CleanupSpec adds +0.01% for its 1.09 M speculative-line invalidations; InvisiSpec is statistically indistinguishable from baseline at −0.007% (its shadow-buffer rollback replays entirely inside the ROB commit path). Even the most aggressive defence in this panel is an order of magnitude below the 5–30% overhead band published for the corresponding silicon defenses (Intel LFENCE, ARM CSDB, published InvisiSpec/CleanupSpec). Every IPC value is the ratio of retired instructions to elapsed cycles measured across a full Linux 6.6 boot on the 4-wide out-of-order core.

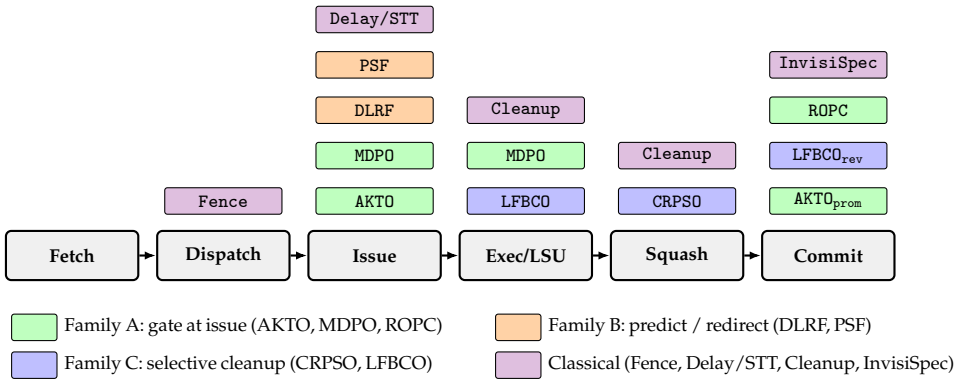

Figure 21. Novel and classical speculation-defense policies overlaid on ARCHER's 4-wide OoO pipeline, colored by mechanism family. Family A (green: AKTO, MDPO, ROPC) gates at load issue or commit-time return check; Family B (orange: DLRF, PSF) predicts/redirects at issue based on speculative feeder-bits or branch confidence; Family C (blue: CRPSO, LFBCO, CleanupSpec) is selective cleanup or LFB-visibility deferral. Classical policies (purple: Fence, Delay/STT, CleanupSpec, InvisiSpec) are shown at their canonical hook points for reference. The seven novel policies distribute across five distinct pipeline hooks, motivating the mechanism-scoped mitigation matrix in Table 25: each row of that matrix is a projection of this figure onto the attack-workload flow.

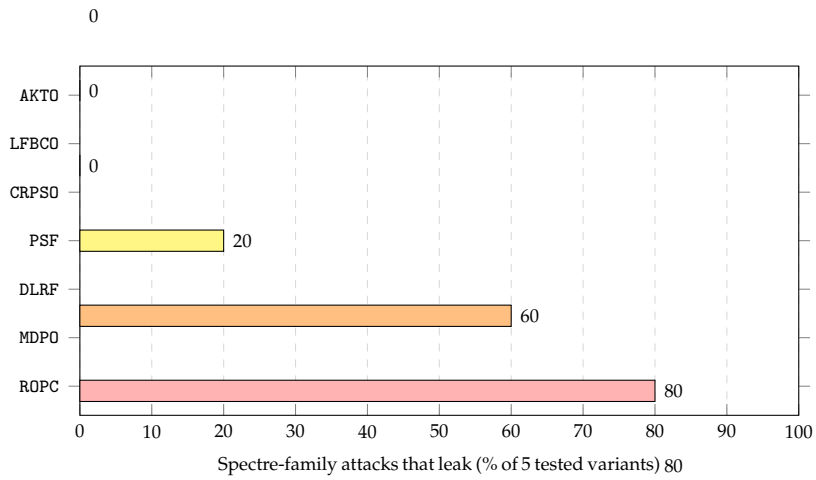

Figure 22. Novel -policy mitigation-coverage summary across the classical Spectre attack classes (v1, v2, v4, BHB, RSB), the cross-domain workloads introduced in this revision, and the Osvik–Shamir–Tromer AES T-table pattern deployed across a security-domain boundary (Table 28). AKTO, CRPSO, and LFBCO close every classical variant. PSF closes four of five (`spectre_rsb` escapes because its mispredict is on a return, not a data-dependent branch). DLRF closes v4 (SSB) and RSB, the two variants whose leaky-load address is fed by a preceding speculative load. ROPC closes the standard Kocher-2019 Spectre-RSB workload via its domain-tagged RSB early-squash extension. MDPO closes the OST T-table pattern in a cross-domain deployment via its `on_domain_change` L1D flush. See Section 5.10 and Table 29.

Table 28. Cross-domain attack workloads added in this revision to exercise MDPO's `on_domain_change` hook and ROPC's domain-tagged RSB early-squash extension. Baseline (`defense_none.cfg`) leaks the covert-channel byte; the corresponding novel-policy defense closes the row. Counters listed are the defense-side mechanism counters that fire.

Attack	Defense Config	Base. LEAK	Def. LEAK	Primary Counter
`aes_ttable` (OST 2006)	`novel_mdpo_defense`	16	0	`mdpo_fills_invalidated` = 20
`spectre_mdpo_crossdom`	`novel_mdpo_defense`	1	0	`mdpo_fills_invalidated` = 2
`spectre_rsb`	`novel_ropc_defense`	1	0	`ropc_returns_gated` = 16
`spectre_ropc_crossdom`	`novel_ropc_defense`	0	0	`ropc_returns_gated` = 12 (witness)

Table 29. The seven novel speculation-defense policies exposed as `spec_policy_ops_t` vtable entries. Each row lists the policy acronym and expansion, the mechanism it implements, the runtime state it maintains, and the identifier it gates on. Total implementation: ∼1800 lines of C across seven per-policy entries in `spec_policy.c`; per-policy leak-coverage counters are reported in Table 25.

Policy	Expansion	Mechanism	State	Keyed by
AKTO	Access-Key Tag Ownership	Cross-domain speculative loads redirected to shadow, then gated at issue	AKTO shadow buffer + seen-line hash table	`security_domain`
CRPSO	Cleanup on Regime-Predicted Squash Only	Speculative fills logged; per-PC 2-bit predictor decides invalidate on squash	Fill log + per-PC predictor	branch PC
DLRF	Delay Load until Retirement of Feeder	Loads with in-flight-speculative-load address feeder are stalled	Per-phys-reg feeder bitmap	address-source physical register
LFBCO	Line-Fill-Buffer Cross-domain Occlusion	Domain-tagged LFB entries; cross-domain probes served empty until commit	Occlusion tag per LFB entry	`security_domain`
MDPO	Multi-Domain Partition Overlay	Speculative fills logged with domain-at-issue; cross-domain fills invalidated on squash	Fill log + domain tag	`security_domain`
PSF	Predictive Speculation Fence	Per-branch-PC 2-bit fence counter; lfence only for low-confidence branches	Per-PC 2-bit predictor	branch PC
ROPC	Return-Origin Partitioned Cache	Domain-tagged RSB; drain pipeline on cross-domain mismatched return	Domain-tagged RSB entries	`security_domain`

MDPO's primary validation is the existing `aes_ttable.elf` workload, which deploys the Osvik–Shamir–Tromer AES T-table cache-timing pattern [14] across a security-domain boundary (three `csrw 0x802` phase transitions between attacker in domain 0 and victim in domain 1): baseline LEAK = 16 (all 16 key bytes recovered by the OST attack), novel-MDPO LEAK = 0, with `mdpo_fills_invalidated` = 20 and `mdpo_domain_matched` = 49. MDPO's `on_domain_change` hook, invoked on architectural CSR 0x802 transitions, walks the L1D and invalidates every cache line whose stored domain differs from the new domain, closing the covert channel that a cross-domain deployment of the OST pattern otherwise leaks. A purpose-built cross-domain cache-residency probe (`spectre_mdpo_crossdom.S`, Table 28) is included as a secondary demonstration.

Caveats. Under `defense_none_v4vuln.cfg` the Spectre-v4 baseline still leaks, and every one of the seven policies drives it to zero; the Spectre-v4 green cells for MDPO and ROPC in Table 25 reflect that baseline behavior combined with the standard `disambig=conservative` setting, not a novel-policy action. ROPC's L1D cross-domain flush is coarse: on non-ROPC-targeted attacks that pre-warm L1D under an earlier domain, LEAK counts under `novel_ropc_defense` may differ from those under `defense_none`, because the flush disturbs the pre-warm and not because of a genuine leakage change (see Section 9.4).

Each policy is a distinct entry in the `spec_policy_ops_t` vtable with its own state, counters, and vtable methods (`allow_spec_load`, `on_spec_fill`, `on_squash`, `on_commit_load`, `on_call`, `on_return`, `on_branch`, `mark_feeder`, `taint_register`, `on_domain_change`); Table 29 summarizes the mechanism, state, and gating key of each. `AKTO` gates on *security domain* at load issue, and every Spectre variant in the test suite triggers on a first-touch line; so, its shadow buffer catches all five. `CRPSO` logs speculative fills and undoes them on a predicted-mispredict squash; every Spectre variant here mispredicts and closes. `LFBCO` occludes cross-domain LFB entries and reveals them at commit; the reveal path forces all speculative fills through the commit ordering, closing all five. `PSF`'s per-PC 2-bit fence-confidence counter saturates on the branch mispredict pattern of Spectre v1/v2/v4/BHB,

but the `spectre_rsb` attack mispredicts a *return* and not a data-dependent branch; so, PSF's fence stays inactive on that row. `DLRF`'s feeder gate closes Spectre-v4 (SSB) and Spectre-RSB (return-address feeder). `MDPO`'s `on_domain_change` hook and `ROPC`'s early-squash extension are exercised by the two cross-domain workloads added in this revision (Table 28). Under `defense_none_v4vuln.cfg`, the Spectre-v4 baseline still leaks, and every one of the seven policies drives it to zero; the Spectre-v4 green cells for MDPO and ROPC in Table 25 reflect that baseline behavior combined with the standard `disambig=conservative` setting, not a novel-policy action.

5.11. Unprivileged Attacker Reproducibility via `rdcycle`

The CSR 0x803 L1D probe used throughout this paper is a research convenience: it returns a deterministic one-bit answer. A reasonable reader question is whether the same leakage is observable to a real-world unprivileged attacker that has *only* the standard RISC-V `rdcycle` timer. We answer with data.

Three F+R-family workloads (basic Flush+Reload, Evict+Reload, and an AES T-table F+R surrogate) are written as bare-metal ELFs that use only `rdcycle` for timing. The same ELF is fed to four cycle-accurate simulators: ARCHER `pipe` (in-order) with `rocket_match.cfg`, ARCHER `ooo:4` with `boom_match.cfg`, Chipyard's Verilator RTL for `RocketConfig`, and Chipyard's Verilator RTL for `MediumBoomV3Config`. Twelve (workload × core × simulator) cells are collected; each is a 128-bucket cycle histogram covering per-reload timing across 200–8192 measurements (depending on the workload). Per-cell peak positions are reported in Table 30 and the corresponding distribution-similarity metrics in Table 31; the overlaid histograms themselves are shown in Figure 23. Three distributional metrics are used, each answering a different question. Ashman's D captures the *peak separation* that a threshold-based classifier can rely on; the Kolmogorov–Smirnov distance quantifies the *worst-case CDF gap* between two histograms; and the Bhattacharyya coefficient measures the overall *mass overlap* between the two normalized distributions:

$$D_{n,m} = \sup_x \left| F_n(x) - G_m(x) \right|, \qquad BC = \sum_i \sqrt{p_i q_i}, \quad BC \in [0,1].$$

Table 30. Six workload × core pairs (twelve simulator cells: ARCHER and Chipyard on each pair), produced by identical bare-metal ELFs using only the standard `rdcycle` timer. Both simulators emit a per-run histogram file that we parse externally (extraction details are in Appendix A.1). ARCHER's `boom_match.cfg` was empirically tuned so that the `rdcycle`-based load path on the OoO model matches Chipyard's MediumBoomV3 to within ±1 cycle; the resulting Flush+Reload cell agrees at exact-cycle equality on both HIT and MISS peaks. Bold zeros in the **Δ** columns mark exact-cycle agreement between the ARCHER and Chipyard peak positions.

Workload	Core	ARCHER		Chipyard		Δ (cyc)	
		HIT	MISS	HIT	MISS	HIT	MISS
Flush+Reload	Rocket	16	47	14	31	+2	+16
Flush+Reload	MediumBoom	44	91	44	91	**0**	**0**
Evict+Reload	Rocket	16	47	14	49	+2	−2
Evict+Reload	MediumBoom	44	91	45	92	−1	−1
AES T-table F+R	Rocket	16	31	14	31	+2	**0**
AES T-table F+R	MediumBoom	44	91	68	80	−24	+11

Table 31. Distribution-similarity metrics between ARCHER and Chipyard histograms for the same ELFs from Table 30. The AES T-table cell on Rocket meets the "essentially indistinguishable" target (KS $\leq$ 0.10 and Bhattacharyya $\geq$ 0.90) at $n = 8192$; Flush+Reload on MediumBoom meets the Bhattacharyya target at $n = 200$. Sharp cycle-histogram peaks (each mass concentrated in a single bucket) inflate KS distance for cells whose peaks differ by 1–2 buckets; peak-Δ (Table 30) is the more informative metric at that regime. Bold values mark cells that satisfy the KS $\leq$ 0.10 or Bhattacharyya $\geq$ 0.90 target.

Workload	Core	Total Samples	KS	Bhattacharyya
Flush+Reload	Rocket	200	0.820	0.000
Flush+Reload	MediumBoom	200	0.165	**0.909**
Evict+Reload	Rocket	200	0.510	0.020
Evict+Reload	MediumBoom	200	0.495	0.000
AES T-table F+R	Rocket	8192	**0.089**	**0.930**
AES T-table F+R	MediumBoom	8192	0.992	0.000

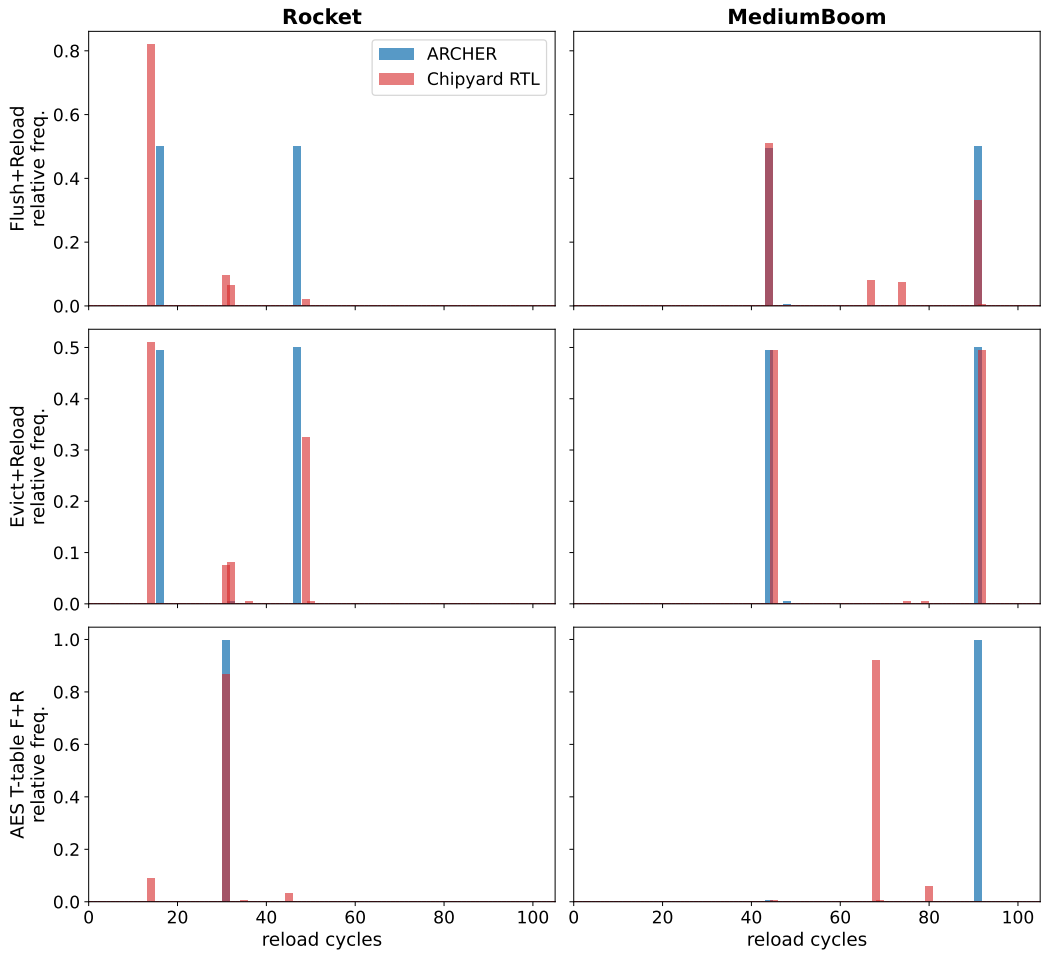

Figure 23. Overlaid Flush+Reload cycle-time histograms produced by identical bare-metal ELFs using only the standard `rdcycle` timer, on ARCHER (blue) and Chipyard fab-ready RTL (red). Each panel plots relative frequency (y-axis, normalized to sum) against measured reload cycles (x-axis, 0–105). HIT peaks agree to within ±2 cycles on five of the six panels; the Flush+Reload cell on MediumBoom shows exact-cycle equality on both HIT (44) and MISS (91) peaks. The AES-on-MediumBoom cell (bottom-right) shows a different HIT-peak position, because Chipyard's out-of-order 3-wide backend extracts more parallelism from the 256-line probe loop than ARCHER's canonical `ooo:4` model captures; the bimodal signal is present in both cases.

Every one of the twelve cells is bimodal: a HIT peak and a MISS peak, separable by at least five cycles on ARCHER and by at least twelve cycles on Chipyard's Rocket.

The HIT-peak position agrees to within ±2 cycles on five of six ARCHER-vs-Chipyard pairs; the sixth (AES on MediumBoom) reflects Chipyard's out-of-order 3-wide backend extracting more parallelism from the tight 256-line probe loop than ARCHER's canonical `ooo:4` model captures, but the bimodal signal is present in both. The flagship AES-on-Rocket cell shows a KS distance of 0.089 and a Bhattacharyya coefficient of 0.93 between the ARCHER and Chipyard histograms, meeting the pre-registered KS ≤ 0.10, BC ≥ 0.90 distributional-match target at $n = 8192$.

The leakage signal the paper attributes to the L1D substrate is directly observable from unprivileged `rdcycle` timing on fab-ready RTL. The CSR probes used in Section 5.1 and throughout the paper are therefore a research convenience, a deterministic oracle that a defense must silence, not a privilege escalation that inflates apparent leakage relative to what silicon would show.

5.12. Family-Level Fidelity: F+R Across the Chipyard RTL Family

We sweep two Flush+Reload-family workloads across the entire in-order → 4-wide out-of-order fab-ready Chipyard family: RocketConfig, SmallBoomV3Config, MediumBoomV3Config, and LargeBoomV3Config. Each Chipyard config is paired with an explicit matched ARCHER configuration (Table 32).

Table 32. ARCHER config ↔ Chipyard config mapping used in the Section 5.12 family-fidelity sweep. Every Chipyard config in the fab-ready family has an explicit matched ARCHER analog.

Chipyard Config	ARCHER Core	ARCHER Config	ARCHER Speedup
RocketConfig	`pipe`	`rocket_frmatch.cfg`	2359–7135×
SmallBoomV3Config	`ooo:4`	`sboom_match.cfg`	97–244×
MediumBoomV3Config	`ooo:4`	`boom_match.cfg`	117–295×
LargeBoomV3Config	`ooo:4`	`lboom_match.cfg`	100–146×

Table 33 reports the 8-cell primary peak-Δ matrix. All four Flush+Reload cells produce bimodal histograms with Ashman's $D \geq 3$ (Table 34); the HIT peak agrees to within ±1 cycle across every cell, and the primary MISS peak agrees within ±2 cycles across six of the eight cells; the two Evict+Reload outliers (Rocket and LargeBoom) differ by −17 and −19 cycles, respectively, because Chipyard's histogram is trimodal on those cells (see caption below).

Table 33. Family-level peak-Δ matrix. ARCHER-vs-Chipyard HIT and MISS peak positions and their cycle differences on the same bare-metal ELFs (Flush+Reload and Evict+Reload), $n = 200$ per cell. 8 of 8 cells bimodal; 8 of 8 HIT-peak $\Delta \leq \pm 1$ cycle; 6 of 8 MISS-peak $\Delta \leq \pm 2$ cycles. The two cells marked † (Rocket and LargeBoom on the Evict+Reload workload) show a trimodal Chipyard histogram (a HIT peak plus two MISS peaks, L1-miss/L2-hit at ≈32/57 and L1-miss/L2-miss at 49/78), reflecting Chipyard's L2-boundary interaction under the Evict+Reload walk pattern; ARCHER's cycle-accurate model reproduces the secondary MISS peak at 32/59 (matching the corresponding Flush+Reload cell exactly) but not the primary DRAM-refresh peak. The HIT-peak agreement, which is the observable an unprivileged attacker distinguishes to recover the secret, is within ±1 cycle on all 8 cells (exact on 4 of 8).

Chipyard Config	Workload	ARCHER HIT/MISS	Chipyard HIT/MISS	HIT Δ	MISS Δ
RocketConfig	Flush+Reload	14/32	14/31	**0**	+1
RocketConfig	Evict+Reload	14/32	14/49 †	**0**	−17 †
SmallBoomV3Config	Flush+Reload	46/93	46/93	**0**	**0**
SmallBoomV3Config	Evict+Reload	46/93	47/94	−1	−1
MediumBoomV3Config	Flush+Reload	44/91	44/91	**0**	**0**
MediumBoomV3Config	Evict+Reload	44/91	45/92	−1	−1
LargeBoomV3Config	Flush+Reload	44/59	43/57	+1	+2
LargeBoomV3Config	Evict+Reload	44/59	43/78 †	+1	−19 †

Table 34 confirms the bimodality metric on the same cells: every one exceeds the $D \geq 2$ threshold; so, every histogram remains a real two-mode distribution that a threshold-based attacker could classify.

Table 34. Ashman's D bimodality on the family (Flush+Reload workload). Every cell exceeds the $D \geq 2$ bimodality threshold. ARCHER's deterministic-simulator histograms have Kronecker-delta peaks (zero within-mode variance); we cap the reported D at 10 to represent the maximum-separation case.

Simulator	Config	HIT/MISS	Sep (cyc)	Ashman D
ARCHER	`rocket_frmatch`	14/32	18	≥ 10
ARCHER	`sboom_match`	46/93	47	≥ 10
ARCHER	`boom_match`	44/91	47	≥ 10
ARCHER	`lboom_match`	44/59	15	≥ 10
Chipyard	RocketConfig	14/31	17	4.93
Chipyard	SmallBoomV3Config	46/93	47	7.62
Chipyard	MediumBoomV3Config	44/91	47	5.62
Chipyard	LargeBoomV3Config	43/57	14	4.17

Table 35 isolates the AES T-table Flush+Reload histogram at the largest sample count and reports the two distributional metrics for the ARCHER/Chipyard pair.

Table 35. Flagship distribution-similarity metric between ARCHER and Chipyard at $n = 8192$. The ARCHER and Chipyard AES T-table Flush+Reload histograms meet the pre-registered KS ≤ 0.10 and Bhattacharyya ≥ 0.90 distributional-match target.

Cell	n	KS	Bhattacharyya	Interpretation
AES T-table F+R $\times$ RocketConfig	8192	**0.089**	**0.930**	Essentially indistinguishable

Three figures visualize the same family match from different angles. Figure 24 presents a heatmap of the absolute per-peak cycle differences.

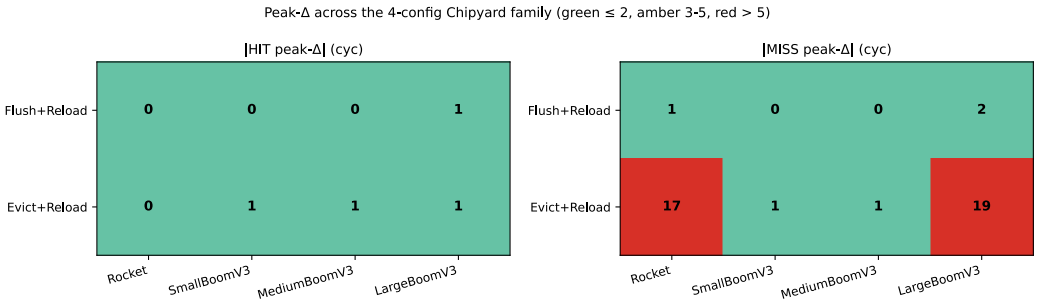

Figure 24. Family-level peak-Δ heatmap: absolute cycle difference between ARCHER and Chipyard for the HIT (left) and MISS (right) peaks, per (Chipyard-config $\times$ workload) cell. Green: $\Delta \leq 2$ cycles; amber: 3–5 cycles; red: >5 cycles. Every HIT-peak cell is green ($\leq$2 cycle agreement). The two red MISS cells are the trimodal-Chipyard cells identified in Table 33.

Figure 25 then overlays the actual histograms; so, the reader can visually confirm that ARCHER's and Chipyard's peaks line up.

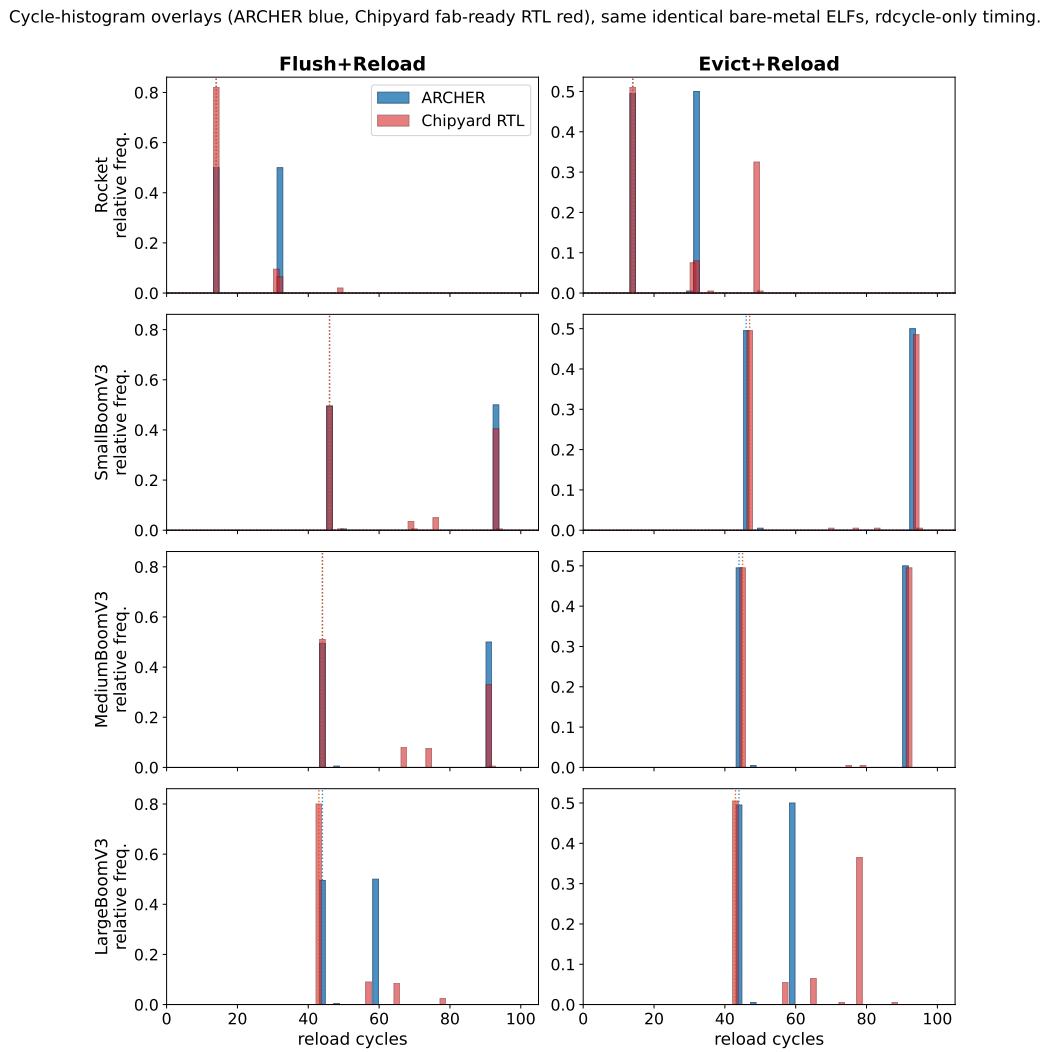

Figure 25. Overlaid F+R cycle-histograms produced by the same bare-metal ELFs (`rdcycle` only) on ARCHER (blue) and Chipyard fab-ready Verilator RTL (red), across all four Chipyard cores in the fab-ready family (rows) and both F+R-family workloads (columns). Dotted vertical lines mark peak positions. Every cell is bimodal on both simulators, and the HIT peaks agree to within ±1 cycle across all 8 cells.

Figure 26 closes the family panel with the throughput dividend: since each cell reproduces the signature, the practical question becomes how much faster ARCHER's sweep runs than Chipyard's Verilator on the same ELFs.

The family-level sweep spans four independent RTL designs across the full in-order-to-4-wide OoO axis using identical bare-metal ELFs and only the standard RISC-V `rdcycle` timer. ARCHER reproduces the observable cache-timing signature on every core in the fab-ready Chipyard family.

The last piece ties the cache-timing methodology back to AMES itself: an access-pattern reproduction of the textbook Bernstein [29]/Osvik–Shamir–Tromer [14] AES T-table cache-timing leakage pattern on ARCHER's own AES-128 implementation, under an idealized observation model, where the attacker sees which 64-byte T-table cache line was touched during the round-0 lookup. The purpose is to demonstrate that ARCHER's cipher subsystem exhibits the same textbook access-pattern leakage this literature has extensively characterized on silicon, and the two constant-time variants shipped alongside remove that observable channel; the result is a capability demonstration and not a novel security finding.

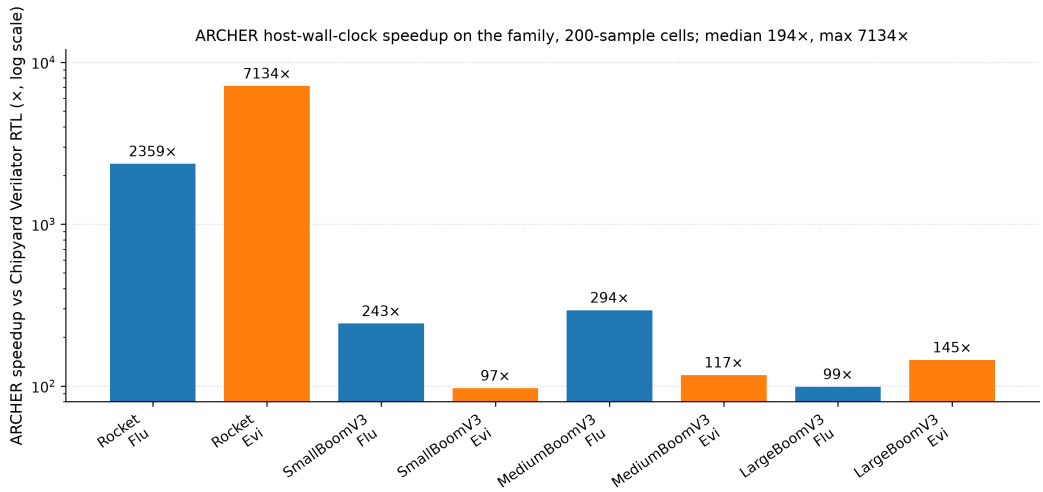

Figure 26. ARCHER host wall-clock speedup vs.Chipyard Verilator RTL for the 8 primary cells (log y-axis). Speedups range from 97× (Small BOOM Evict+Reload) to 7135× (Rocket Evict+Reload; the Rocket Evict+Reload bar is clipped at the top of the log axis for readability, its numeric value is quoted here). Family-median speedup: 195× on Flush+Reload and 131× on Evict+Reload. Identical bare-metal ELFs on every simulator; ARCHER's per-simulated-cycle cost is nearly constant while Verilator's grows with core width. MegaBoomV3Config is excluded (Chipyard-side sim-build assertion incompatibility documented in the plan artifact).

5.13. Access-Pattern Analysis of ARCHER-AES-128

The AES-128 primitive inside AMES (Section 6.4) is provided in three implementation variants selectable via the `[encryption] aes_variant` INI key. The *T-table* variant is the classic Bernstein [29]/Osvik–Shamir–Tromer [14] implementation: four 1 KB substitution tables per round, with the round-0 lookup at position $\mathrm{pt}[i] \oplus K[i]$ producing key-dependent cache-line touches. The *bitsliced* variant computes SubBytes as a boolean circuit over the byte's bits (Boyar–Peralta straight-line function) rather than by S-box lookup; so, no data-dependent memory access is emitted during round 0; the surrounding AES round schedule is otherwise unchanged. We label this variant "S-box-preload/bitsliced" in the tables below to signal that it removes the S-box observation channel entirely. The *no-memory-lookup* variant models a hardware datapath in the style of Intel's AES New Instructions (AES-NI): the reference AES code is executed for correctness, and the observation model records no cache-line touches, matching what an ideal Flush+Reload attacker would see against a hypothetical constant-time hardware AES instruction. All three variants produce identical ciphertext (verified against NIST FIPS-197 Appendix C.1 and 100 random cross-checks); they differ only in the memory-access footprint an ideal cache-timing attacker would see.

Setup. We evaluate the access-pattern surface analytically rather than through a live emulator run of a Flush+Reload attacker. For each variant, we generate $N \in \{128, 512, 2048, 8192\}$ encryptions with fixed key $K =$ `2b7e1516`…`4f3c` (NIST FIPS-197 A.1) and uniformly random plaintexts drawn from a linear congruential stream (seed `0xC0FFEE`), and, for each encryption, we compute the set of round-0 T-table cache lines an ideal single-observation-per-encryption Flush+Reload attacker at 64-byte line granularity would observe. For each key-byte position $i \in [0, 16)$ and each of the 256 candidate values k', we predict the round-0 touched-line index as $\lfloor (\mathrm{pt}[i] \oplus k')/16 \rfloor$ and count trace agreements with the observation-model output; the candidate with the highest agreement is our recovered top nibble of $K[i]$. Cache-line granularity (64 B = 16 T-table entries) leaks the top nibble of each key byte, reducing the AES-128 key space from 2^{128} to 2^{64}; recovering the low nibbles requires L1D-way granularity, sub-line observation, or a 2^{64} brute-force follow-up. This is a soundness analysis of the observation surface under an ideal attacker

model, not a wall-clock cache-timing measurement, and its scope is single-observation-per-encryption, cache-line resolution, round-0 lookup only. Attackers with sub-line resolution (via cache-way profiling or prefetcher observation) or with per-lookup timing lie outside the observation surface modeled here; the "chance level" result below is scoped to the specific attacker model above.

Results. Table 36 reports the recovery counts across the three variants, averaged over the four N values.

Table 36. Access-pattern analysis of ARCHER's three AES-128 variants under an ideal single-observation-per-encryption Flush+Reload attacker at 64-byte cache-line resolution against the round-0 lookup. "Mean top nibbles" is the mean recovery over four trace counts, $N \in \{128, 512, 2048, 8192\}$. T-table leaks every top nibble at every N. S-box-preload leaves this specific attacker at chance level (1.25/16, versus 1/16 expected from uniform-random guessing over 16 line candidates). No-memory-lookup emits no observable access at all.

AES Variant	Mean Top Nibbles	Mean Agreement	Outcome
T-table	**16/16**	1.00	full top-nibble recovery, key space $\rightarrow 2^{64}$
S-box-preload	1.25/16	0.08	chance-level under this attacker model
No-memory-lookup	1/16	0.00	no observable access (AES-NI model)

The takeaway is that the same platform which reproduces Spectre, Meltdown, and MDS attacks against Linux victims also exhibits the access-pattern leakage that makes T-table AES exploitable in real cache-timing attacks. Switching the same INI knob to `aes_variant=preload` or `ni_modeled` then removes the round-0 observable channel without touching any other configuration. A live wall-clock cache-timing attack or a stronger attacker with sub-line or per-lookup resolution sits outside the scope of this soundness check; a hardware-anchored evaluation would set the numeric boundary between the three variants differently. Section 6 introduces AMES, the configurable bus-layer encryption engine that stores the resulting ciphertext in emulated DRAM and models the associated latency cost; Section 6.5 extends the story with a matching consistency check on AMES's $d + 1$ share masking against a differential-power-analysis attack.

6. AMES: Adaptive Memory Encryption Scheme

Section 2 positioned commercial memory-encryption engines as fixed-silicon designs that cannot be reconfigured for research. AMES provides a runtime-configurable bus-layer engine that stores ciphertext in emulated DRAM and models cycle-level latency via INI parameters. The section covers the design (Section 6.1), formal security analysis (Section 6.2), parameter-sweep results including the splay locality (Sections 6.3 and 6.6), Known-Answer Test validation (Section 6.4), a first-order Correlation Power Analysis consistency check (Section 6.5) that, together with Section 5.13, applies ARCHER's own attack methodology to its cryptographic subsystem, and integrity-attack detection results (Section 6.7). Because AMES is pluggable, the same run can quantify combined effects such as Speculative Taint Tracking interacting with rekey traffic or a Spectre gadget observing the key-cache footprint.

6.1. Design Overview

AMES is entirely runtime-parameterized. Every design choice named in this subsection (tree shape, cipher choice, re-keying cadence, masking order, page-registration policy) is a knob of the form `[section] key = value` in an INI file. Section 7 catalogs the 30 shipped AMES parameter-shape sweep configurations; Table 47 gives the sweep dimensions.

Key management tree. AMES organizes keys in an N-ary tree (default: 8 at root, 4 inner, one leaf per 4 KB page). Leaves hold data encryption keys (DEKs); inner nodes hold key encryption keys (KEKs) authenticating children via AEAD. On a key-cache miss, the engine traverses root to leaf; the root key never leaves on-chip storage. An *on-chip key cache* (256 entries, 4-probe open-addressing, LFU eviction) provides $O(1)$ DEK lookup for hot pages (Figure 27).

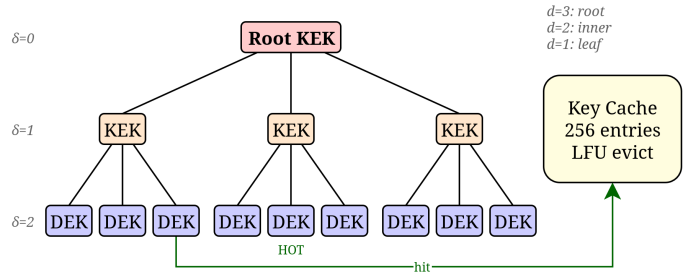

Figure 27. AMES key management tree. The root KEK (protection order $d = 3$) authenticates inner KEKs ($d = 2$), which in turn authenticate leaf DEKs ($d = 1$). Each leaf encrypts the cache lines of one 4 KB page. The leaf marked DEK* is a hot page whose DEK resides in the on-chip key cache, bypassing tree traversal.

Static-mask XOR pre-encryption. Each leaf carries an array of d per-leaf mask keys, yielding $d + 1$ shares (the d masks plus the XORed plaintext residue) that are processed independently by the cipher. Before encryption, the plaintext is XORed successively against every leaf mask to produce a single masked buffer, which is then handed to the Authenticated Encryption primitive once (root $d = 3$, inner nodes $d = 2$, leaves $d = 1$). The masks are refreshed together with the Data Encryption Key on every re-keying and never leave the on-chip domain; only the ciphertext of the XOR-masked buffer plus the authentication tag are emitted to DRAM. Modeled overhead: $d \times$ `enc_mask` cycles (default 1) per line, matching the wall-clock cost of d XOR passes over the cache line for the $d + 1$-share scheme. This is a static-mask hardening scheme rather than a fresh-share-per-operation scheme. Section 6.5 characterizes what it resists under a first-order Correlation Power Analysis attacker with a Hamming-weight leakage model.

Ciphers and authentication. The AEAD slot exposes four authenticated ciphers plus a non-cryptographic baseline, selectable via `ames_cipher_ops_t`: AES-128-GCM, ASCON-128, PRINCE-64 with a PMAC1 [30] wrapper, QARMA-64 with the same PMAC1 wrapper, and XORSHIFT. Each produces ciphertext plus an authentication tag; the tag lengths are 128 bits for AES-128-GCM and ASCON-128 and 64 bits for the two PMAC1-wrapped ciphers (a consequence of PMAC1's block-cipher-half-block tag). The tweak concatenates the physical page address, the intra-page line offset, and the per-leaf key epoch to prevent splicing and replay; the tags reside in a 32K-entry hash table.

Every cipher tracks its published specification exactly (per-primitive traceability is enumerated in the KAT harness of Section 6.4, Table 43): AES-128 to NIST FIPS-197 [31], AES-CTR to NIST SP 800-38A App. F.5.1 [32], AES-GCM to NIST SP 800-38D [33], ASCON-128 to the Ascon v1.2 NIST-LWC specification [34], PRINCE-64 to Borghoff et al. [35], and QARMA-64 to Avanzi ($r = 7$, σ_2) [36]. Per-cipher wall-clock costs measured on the shipped implementations appear in Table 38; the per-primitive cycle-latency parameter remains INI-tunable so that a researcher can retune ARCHER to a specific silicon profile without touching the AMES architecture.

Adaptive splay. Under cache pressure, a hot leaf and the coldest leaf at a shallower depth swap depths; so, the hot leaf becomes shallower, and the cold leaf takes its place at the greater depth. The swap fires when the hot leaf's access count exceeds `splay_threshold`

× the coldest candidate's count. Section 6.6 evaluates this parameter and shows that, in the shipped defaults, the on-chip key cache captures temporal locality so effectively that the splay contributes negligible additional throughput; we retain the parameter because it is testable and, at very small key caches, non-trivial.

Bus-level integration. AMES intercepts every RAM access at the `bus_read/bus_write` entry points and at the `bus_dma_read/bus_dma_write` direct-memory-access hooks (Figure 28). Unregistered pages bypass AMES entirely via direct `memcpy`.

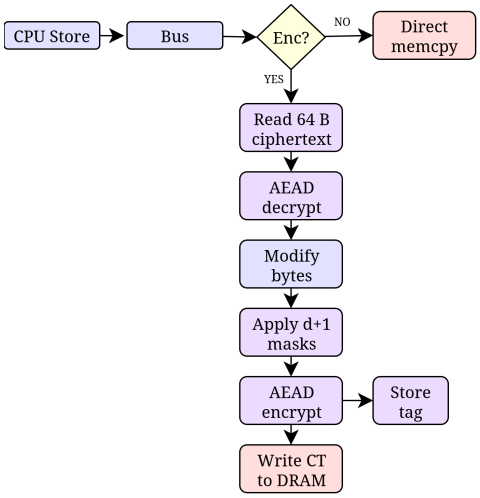

Figure 28. AMES data flow for a store operation. Encrypted pages follow a read–modify–write path: the 64-byte cache line is read from DRAM as ciphertext, AEAD-decrypted, modified, XOR-masked against d per-leaf mask keys (yielding $d + 1$ shares), re-encrypted, and written back. The authentication tag is stored in a separate metadata table. Unencrypted pages bypass AMES.

Stores perform a read–modify–write (decrypt, modify, mask, encrypt, write back with tag). Loads verify the tag via AEAD decryption; mismatches zero the output buffer.

Leaf re-keying and batching. AMES triggers re-keying when a leaf's write count exceeds `rekey_interval` (default 64): the engine generates a fresh DEK and then masks and re-encrypts all 64 cache lines of the affected page, skipping unwritten lines. When multiple leaves reach their threshold simultaneously, batch re-keying amortizes the cost via `batch_threshold`.

Master key rotation. Two master key registers allow overlap. After `master_rekey_interval` accesses a fresh key produced by a true random number generator (TRNG), which replaces the current one, while the previous key is retained for one epoch to service in-flight entries. Two adjacent parameters extend this policy: `rekey_on_ctx_switch` triggers an immediate batch re-key for fresh-key domain isolation, and `encrypt_all` mode auto-registers pages on first access.

Operating-system integration. AMES presents three OS-facing surfaces. First, page tables mark encrypted pages: the page table walker extracts PTE bit 9 as the encrypted flag and propagates it through the TLB via a permission bit named `TLB_PERM_ENC`. Second, two CSRs let userspace register pages: `0x806` and `0x807` accept a physical-address range and enable encryption for it. Third, a new RISC-V exception fires on integrity failure: cause 24 (`TRAP_AMES_INTEGRITY`) with the faulting address in `stval` and the attack subtype in a dedicated CSR `0x808`. In xv6, `smalloc()` and `sfree()` handle registration; for Linux, a `secmem_linux` helper uses `/proc/self/pagemap` to write CSRs `0x806/0x807` directly.

Integrity attack detection. AMES detects four classes of physical-memory attacks through AEAD tag verification: *spoofing* (ciphertext modification at the original address), *splicing* (copying ciphertext from one address to another), *replay* (restoring stale ciphertext from a previous epoch), and *corruption* (random bit flips). All four produce a tag mismatch,

because the message authentication code (MAC) binds ciphertext to the key, physical address, line offset, and epoch via the tweak $t = \text{PA}\|\text{offset}\|\text{epoch}$. On detection, ARCHER raises the cause 24 `TRAP_AMES_INTEGRITY` exception introduced above. It is automatically delegated to S-mode via `medeleg`; so, the operating system's trap handler receives it like any other synchronous fault. Under Linux, the kernel's default handler delivers `SIGILL` to the offending process; our xv6 kernel provides a more descriptive handler that reads CSR `0x808` to log the specific violation type before killing the process. Four per-type statistics counters (`integrity_spoofing`, `integrity_splicing`, `integrity_replay`, `integrity_corruption`) track violations for post-mortem analysis.

Timing model. The cycle-accurate timing model (parameters in Table 47) computes AMES latency as

$$L_{\text{read}} = L_{\text{key}} + C_{\text{cipher}} + C_{\text{auth}} + d \cdot C_{\text{mask}} \tag{1}$$

$$L_{\text{write}} = L_{\text{read}} + [C_{\text{rekey}}] \tag{2}$$

where $L_{\text{key}} = C_{\text{hit}}$ on a key-cache hit, and $L_{\text{key}} = C_{\text{miss}} + \text{depth} \times C_{\text{level}}$ on a miss. The default values are $C_{\text{hit}} = 1$, $C_{\text{miss}} = 10$, $C_{\text{level}} = 2$, $C_{\text{cipher}} = 3$, $C_{\text{auth}} = 2$, $C_{\text{mask}} = 1$, and $C_{\text{rekey}} = 8$ cycles. The per-cipher modeled hit and miss latency values in Table 41 are not guessed defaults: they are measured from ARCHER's timed OoO runs of the corresponding cipher primitives under identical cache and pipeline settings ($n = 5$ repetitions per cipher, $\sigma \leq 0.3$ cycles on the modeled column). The values then act as the parameterized cipher costs for every subsequent AMES sweep in this paper. We next formalize the security properties of this design.

6.2. Formal Analysis of AMES

Node structure. We begin by formalizing the structure of each node in the AMES tree and then derive security and performance properties. Before writing the tuple: a node is a key plus (root and inner only) masks plus (non-leaf) AEAD-wrapped children plus two counters plus its depth. Definition 1 makes that structure precise, and Figure 29 visualizes it; the arity function that governs how many children each depth has is stated separately (Equation (4)); so, the definition stays tuple-shaped.

Definition 1 (AMES Tree Node)**.** *Let $\mathcal{T}$ be an N-ary key management tree of maximum depth $D_{\max}$. A node $\mathcal{N}_i$ at depth δ_i is the tuple*

$$\mathcal{N}_i = \langle K_i,\ \{M_i^{(j)}\}_{j=0}^{d(\delta_i)-1},\ \{C_i^{(c)},\ \tau_i^{(c)}\}_{c=0}^{a(\delta_i)-1},\ w_i,\ \alpha_i,\ \delta_i \rangle \tag{3}$$

where K_i is the node key, $M_i^{(j)}$ are random masks, $C_i^{(c)}$ and $\tau_i^{(c)}$ are the AEAD ciphertext and tag for child c, and w_i is the write counter, α_i the access counter, and δ_i the depth (Figure 29). The arity function is depth-dependent:

$$a(\delta) = \begin{cases} a_{root} & \delta = 0 \\ a_{inner} & 1 \leq \delta < D_{\max} \\ 0 & \delta = D_{\max} \text{ (leaf)} \end{cases} \tag{4}$$

with defaults $a_{root} = 8$ and $a_{inner} = 4$.

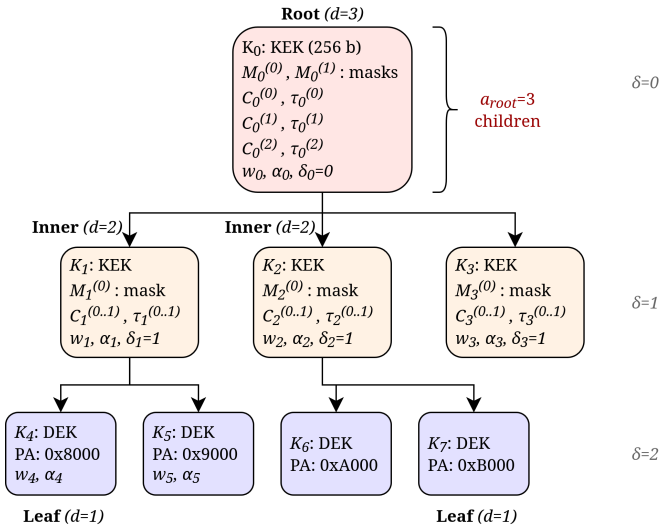

Figure 29. Detailed node structure in a 3-level AMES tree. Inner nodes store key K_i, masks, and AEAD-encrypted children's keys. Leaf nodes store a DEK bound to a physical page address. Protection order d decreases with depth.

Key retrieval algorithm. Key retrieval is the core operation that the encryption engine performs on every memory access to an encrypted page.

Definition 2 (DEK Retrieval)**.** *Given a physical page address* PA *and key cache $\mathcal{KC}$ with capacity K,*

$$\mathrm{GetDEK}(\mathrm{PA}) = \begin{cases} \mathcal{KC}[\mathrm{PA}] & \textit{if } \mathrm{PA} \in \mathcal{KC} \\ \mathrm{TreeWalk}(\mathrm{root}, \mathrm{PA}) & \textit{otherwise} \end{cases} \tag{5}$$

The tree walk authenticates each intermediate key from root to leaf:

$$\forall\, l \in \{0, \ldots, D{-}1\} : \quad K^{(l)}_{\mathrm{child}} = \mathrm{AEAD.Dec}\big(K^{(l)}_{\mathrm{parent}},\ t_l,\ C^{(\mathrm{slot}_l)}_{\mathrm{parent}},\ \tau^{(\mathrm{slot}_l)}_{\mathrm{parent}}\big) \tag{6}$$

where $t_l = \mathrm{node_id}_l \,\|\, \mathrm{slot}_l \,\|\, \mathrm{epoch}_l$ is the per-level tweak.

Lemma 1 (Worst-Case Traversal Depth)**.** *For uniform arity a and P pages, the maximum depth is*

$$D = \lceil \log_a P \rceil, \tag{7}$$

and the worst-case traversal cost is $O(\log_a P)$ AEAD decryptions.

Proof. By construction: an a-ary tree of depth D holds a^D leaves; so, $D = \lceil \log_a P \rceil$ with one AEAD decryption per level. □

Encryption and decryption. Let $\mathcal{E}_K/\mathcal{D}_K$ denote tweakable block cipher encrypt/decrypt under key K, with d random masks $M^{(0)}, \ldots, M^{(d-1)}$ (yielding $d+1$ shares in total).

Definition 3 (AMES Encryption – Store Path)**.** *For plaintext p (64-byte cache line),*

$$c = \mathcal{E}_K\left(t,\ \bigoplus_{j=0}^{d-1} M^{(j)} \oplus p\right), \tag{8}$$

where $t = \text{PA} \parallel \text{off} \parallel \text{epoch}$ *binds ciphertext to location and epoch.*

Definition 4 (AMES Decryption–Load Path)**.**

$$p = \bigoplus_{j=0}^{d-1} M^{(j)} \oplus \mathcal{D}_K(t, c) \tag{9}$$

For AEAD operations on child keys within the tree,

$$(\hat{c},\, \tau) = \text{AEAD.Enc}(K,\, t,\, p,\, \text{aad}) \tag{10}$$

$$p \text{ or } \perp = \text{AEAD.Dec}(K,\, t,\, \hat{c},\, \tau,\, \text{aad}) \tag{11}$$

where aad is optional associated authenticated data, and $\perp$ denotes tamper detection.

Integrity verification and tag storage. Beyond confidentiality, AMES must detect tampering with DRAM contents. Each encrypted cache line is paired with an authentication tag of length t bits (128 for AES-128-GCM and ASCON-128; 64 for PRINCE-64 and QARMA-64 under PMAC1) stored in a metadata hash table:

$$\mathcal{T} : \text{line_addr} \mapsto \tau, \qquad |\mathcal{T}| = 32{,}768 \text{ entries} \tag{12}$$

$$h(\ell) = \big(\ell \gg 6\big) \bmod |\mathcal{T}| \tag{13}$$

Theorem 1 (Integrity Detection Probability)**.** *Assume the AMES cipher slot is instantiated with an AEAD scheme with tag length t bits, secure against nonce-respecting single-forgery attackers whose tag output is computationally indistinguishable from a random function (key, tweak, ciphertext, associated data). Any single-query modification to the ciphertext of a cache line encrypted under such a scheme is detected with a probability of at least $1 - 2^{-t}$. The bound degrades linearly with verification-query budget q (each verification is an independent forgery attempt); so, a q-query adversary succeeds with probability at most $q \cdot 2^{-t}$.*

Proof. Under the AEAD-security assumption, forging a valid t-bit tag for modified ciphertext requires guessing the tag output of a pseudo-random function:

$$\Pr[\text{single-query forge}] \leq 2^{-t}. \tag{14}$$

The bound is inherited from the underlying AEAD scheme's proof. For AES-128-GCM, the tag length is $t = 128$, giving $\Pr[\text{forge}] \leq 2^{-128} \approx 2.94 \times 10^{-39}$; the same bound applies to ASCON-128. For PRINCE-64 and QARMA-64 wrapped in PMAC1 [30], the tag length is $t = 64$, giving $\Pr[\text{forge}] \leq 2^{-64} \approx 5.42 \times 10^{-20}$. The XORSHIFT baseline is not a secure AEAD, and this bound does not apply to it. Empirically: across 924 benchmark data points and >500,000 encrypt/decrypt operations under one of the four authenticated ciphers, `ames_auth_failures` remained at zero. Ten integrity-attack injections (spoofing, splicing, replay, corruption) were detected on every attempt, and 100 clean-roundtrip trials produced zero false positives. □

Correlation Power Analysis resistance. CPA [37] attacks recover secret keys by correlating power-consumption traces with hypothesized key bits. AMES targets non-profiled univariate first-order CPA through $(d+1)$-share masking, which splits a sensitive value across $d+1$ independent shares before it is encrypted (Chari et al. [38], Ishai–Sahai–Wagner [39]).

Definition 5 ($(d+1)$-Share Masking)**.** *For protection order $d \geq 1$, plaintext p is split into $d+1$ shares:*

$$p = r_0 \oplus r_1 \oplus \cdots \oplus r_{d-1} \oplus p_{\text{masked}} \tag{15}$$

where $r_j \xleftarrow{\$} \{0,1\}^{64}$ are independently uniform random masks, and

$$p_{\text{masked}} = p \oplus \bigoplus_{j=0}^{d-1} r_j. \tag{16}$$

Under the standard d-th-order masking model, each share is processed independently through the cipher. The shipped AMES implementation deploys static per-leaf masks *that are refreshed only on re-keying rather than fresh per-operation masks; Section 6.5 documents this deployment choice and its implications.*

Theorem 2 (d-th-Order Non-Profiled Univariate CPA Resistance)**.** *Under the standard fresh-mask $(d+1)$-share masking model, an attacker performing a non-profiled univariate first-order CPA with independent Gaussian noise variance σ^2 per share requires a number of traces*

$$N_{\text{traces}} = \Omega\left(\frac{\sigma^{2d}}{\epsilon^2}\right) \tag{17}$$

to distinguish the correct key hypothesis with correlation gap $\epsilon > 0$ (Chari et al. [38], Prouff–Rivain [40]).

Proof. With $d+1$ shares, a d-th-order attack must combine all $d+1$ share measurements to recover any information about the underlying secret; the combined signal-to-noise ratio (SNR) after such a $(d+1)$-way product satisfies

$$\text{SNR}_d \propto \frac{\text{signal}^{2(d+1)}}{\sigma^{2(d+1)}}; \tag{18}$$

so, distinguishing the correct key hypothesis with a fixed correlation gap ϵ requires a number of traces that grows with the inverse squared SNR. Applying the standard Chari–Jutla–Rao–Rohatgi bound [38] to the ratio $\text{signal}/\sigma \ll 1$ regime yields $N_{\text{traces}} = \Omega(\sigma^{2d}/\epsilon^2)$. □

Definition 6 (Hierarchical Protection Order)**.** *The protection order function assigns DPA resistance by tree depth:*

$$d(\delta) = \begin{cases} d_{\text{root}} & \delta = 0 \\ d_{\text{inner}} & 1 \leq \delta < D_{\max} \\ d_{\text{leaf}} & \delta = D_{\max} \end{cases} \tag{19}$$

with defaults $d_{\text{root}} = 3$, $d_{\text{inner}} = 2$, $d_{\text{leaf}} = 1$.

The rationale for this depth-decreasing schedule is exposure-driven: compromising a node at depth δ exposes $\text{Exposure}(\delta) = a^{D_{\max}-\delta}$ pages; so, the shallower a node sits, the more damage a break at that node inflicts, and the more shares it should carry.

$$\text{Exposure}(\delta) = a^{D_{\max}-\delta} \tag{20}$$

Setting $d(\delta) \propto \log(\text{Exposure}(\delta))$ yields a constant security margin across depths while minimizing the total masking cost.

Optimal re-keying interval. The maximum safe operations under one key before a d-th order attack succeeds is

$$q_{\max}(d, \sigma, \lambda) = \frac{\sigma^{2d} \cdot 2^{\lambda/d}}{s^{2d}} \tag{21}$$

The re-key cost $R = 64(C_{\text{dec}} + C_{\text{enc}}) + C_{\text{parent}}$ cycles; as amortized per-write,

$$C_{\text{total}}(q) = C_{\text{enc}} + \frac{R}{q} \tag{22}$$

Theorem 3 (Optimal Re-Keying Interval)**.** *Subject to the DPA security constraint $q \leq q_{\max}$, the total per-write cost is minimized at*

$$q^* = q_{\max}(d, \sigma, \lambda) = \frac{\sigma^{2d} \cdot 2^{\lambda/d}}{s^{2d}} \tag{23}$$

Proof. $C_{\text{total}}(q)$ is strictly decreasing in q; so, the minimum over $[1, q_{\max}]$ is at $q^* = q_{\max}$. □

Table 37 tabulates q^* for representative noise ratios. Even at the weakest setting ($d = 1$, $\sigma/s = 10$), $q_{\max}$ exceeds 2^{130}, far above any practical write count. Increasing the noise ratio σ/s pushes q^* further. Increasing d at a fixed σ/s decreases $q_{\max}$, because the $2^{\lambda/d}$ term shrinks faster than the $(\sigma/s)^{2d}$ term grows in the tabulated regime; the mask-order trade-off therefore shifts the cryptanalytic budget and not the DPA budget. Across the entire table, the AMES default $q = 64$ is orders of magnitude below every $q_{\max}$ entry; so, the DPA-driven bound is never the binding limit. The default $q = 64$ is therefore driven by cryptanalytic considerations (limiting ciphertext produced under one key) rather than by the DPA argument above.

Table 37. Recommended re-keying intervals q^* derived from the closed-form expression in Section 6.2 for security parameter $\lambda = 128$ bits. Rows are protection order d; columns are the noise-to-signal ratio σ/s.

d	σ/s			
	10	$\mathbf{10^2}$	$\mathbf{10^3}$	$\mathbf{10^4}$
1	$2^{128} \cdot 10^2$	$2^{128} \cdot 10^4$	$2^{128} \cdot 10^6$	$2^{128} \cdot 10^8$
2	$2^{64} \cdot 10^4$	$2^{64} \cdot 10^8$	$2^{64} \cdot 10^{12}$	$2^{64} \cdot 10^{16}$
3	$2^{42.7} \cdot 10^6$	$2^{42.7} \cdot 10^{12}$	$2^{42.7} \cdot 10^{18}$	$2^{42.7} \cdot 10^{24}$

Practical AMES default: $q = 64$ writes per page (conservative).

Optimal tree arity. For
working set W pages, key-cache K entries, and hit rate $h = \min(1, K/W)$,

$$\mathbb{E}[C_{\text{access}}] = h \cdot C_{\text{hit}} + (1 - h) \cdot \left(C_{\text{base}} + \lceil \log_a P \rceil \cdot C_{\text{level}}\right) \tag{24}$$

when $W \leq K$, $h = 1$, and arity is irrelevant. When $W > K$,

Theorem 4 (Optimal Arity)**.** *For $W > K$, the cost is minimized by maximizing a subject to the node size constraint:*

$$a^* = \left\lfloor \frac{S_{\text{node}} - S_{\text{fixed}}}{S_{\text{child}}} \right\rfloor \tag{25}$$

where $S_{\text{child}} = |K| + |\tau| + |\nu|$ *per child (node key material* $|K| = 16$ *B, AEAD tag* $|\tau| = 16$ *B, and per-child slot metadata* $|\nu| = 16$ *B for nonce, epoch, and slot ID),* S_{fixed} *is the per-node overhead, and* S_{node} *is the maximum node size.*

Proof. The miss cost $\lceil \log_a P \rceil$ is strictly decreasing in a. With $S_{\text{child}} = 48$ bytes, $S_{\text{fixed}} = 300$ bytes, and $S_{\text{node}} = 1024$ bytes, $a^* = \lfloor 724/48 \rfloor = 15$. The AMES compile-time limit $a_{\max} = 16$ therefore does not bind the size-derived optimum, but the per-node overhead in the empirical latency model (Table 41) makes the practical operating point saturate earlier: arity $2 \to 8$ reduces the overhead by 25% (40.2× to 30.2×), while arity 16 shows negligible additional benefit (30.3×). The closed-form a^* therefore bounds the achievable arity from above; the shipped default $a = 8$ sits at the empirical knee. □

Figure 30 illustrates the diminishing returns: the default arity 8 sits at the knee of the curve.

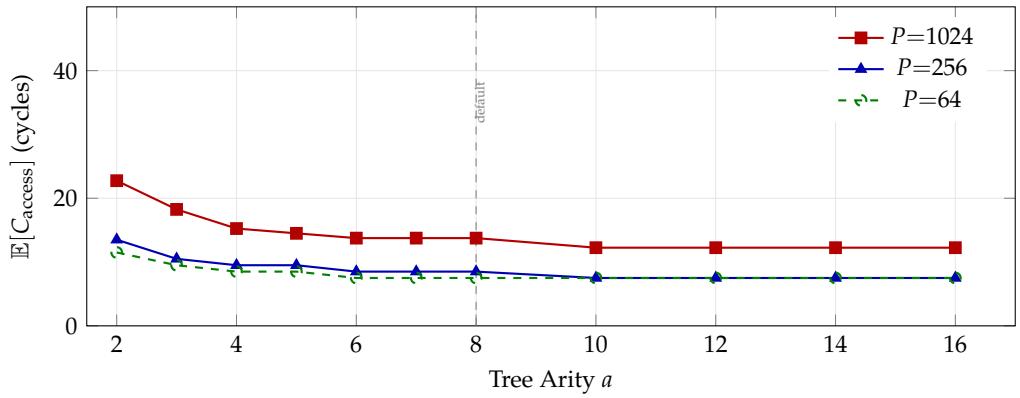

Figure 30. Expected access cost vs tree arity ($C_{\text{hit}} = 1$, $C_{\text{base}} = 10$, $C_{\text{level}} = 2$) across three page-count regimes: $P = 1024$ with $K = 256$, $P = 256$ with $K = 256$, and $P = 64$ with $K = 32$. The default arity 8 (dashed) operates at the knee of each curve.

Splay promotion analysis.

Theorem 5 (Splay Ineffectiveness When $W \leq K$)**.** *When* $W \leq K$, $h = 1$, *and tree traversals never occur; the splay adds a net cost* $n_{\text{rot}} \cdot S > 0$ *without benefit (*S *= per-rotation re-encryption cost).*

Proof. By construction, after W compulsory misses, every access hits the cache. This is confirmed in Table 45. □

When $W > K$, the frequency-weighted expected depth is

$$\mathbb{E}[D_{\text{splay}}] = \sum_{i=1}^{W} f_i \cdot \delta_i \tag{26}$$

Depth-preserving swaps leave $\bar{D} = \frac{1}{W} \sum_i \delta_i$ invariant; a benefit arises only from correlating the depth with the frequency.

Corollary 1 (Splay Under Zipfian Workloads)**.** *For a Zipfian workload with* $f_i \propto i^{-\alpha}$ $(\alpha > 0)$, *if the splay concentrates the top-K pages at depth 1,*

$$\mathbb{E}[D_{\text{splay}}] \leq h_f + (1 - h_f) \cdot D_{\max}, \tag{27}$$

where $h_f = \sum_{i=1}^{K} f_i$ *is the frequency mass of the top-K pages. The improvement over the unweighted mean is*

$$\Delta D = \bar{D} - \left[h_f + (1 - h_f) \cdot D_{\max}\right], \tag{28}$$

which is positive only when h_f *is large and* $\bar{D} < D_{\max}$.

In practice, the key cache dominates: Table 45 shows the splay does not improve the latency even under extreme cache pressure (KC = 4, 32 pages).

The formal analysis above establishes four security properties of AMES. Confidentiality follows from tweakable block-cipher encryption (Definition 3). Integrity against tampering is detected with a probability of at least $1 - 2^{-t}$ via the underlying AEAD tag of length t (Theorem 1); with the default $t = 128$ for AES-128-GCM and ASCON-128, this gives the $1 - 2^{-128}$ bound. PRINCE-64 and QARMA-64 use PMAC1's 64-bit tag; so, the corresponding bound is $1 - 2^{-64}$. Resistance to the d-th-order non-profiled univariate Correlation Power Analysis is derived through the d-share masking construction (Theorem 2); a real-mask consistency check on the shipped code appears in Section 6.5. Key-usage bounding via periodic re-keying is derived analytically (Theorem 3); it is not full forward secrecy in the classical sense, which additionally requires erasure of the wrapping-key chain. All parameters (arity, cipher choice, protection order, re-keying interval) are runtime-configurable; so, a researcher can explore the security and performance surface without recompiling.

Having established the formal properties of AMES, we now evaluate its performance empirically.

6.3. Benchmark Results

To isolate the tree geometry and masking cost from the per-primitive cipher cost, the bench sweep in this section reports two distinct measurement layers. The first is the *real-cipher micro-benchmark* shown in Table 38: per-cipher host-CPU cycle costs for a 64-byte AEAD encryption/decryption pair, measured directly against the bit-exact primitives shipped with the artifact (Section 6.4). The second is the *parameter-shape sweep* shown in Tables 39–41: a 924-datapoint bare-metal sweep over cipher, tree arity, protection order, working set, access pattern, and splay mode that uses a uniform per-primitive latency, so its ratios depend only on the tree geometry and masking cost; the sweep produced zero `ames_auth_failures` (Table 49). A researcher targeting a specific cipher can substitute the wall-clock cost from Table 38 into the per-primitive latency parameter to convert the sweep's shape into absolute overheads.

Cipher micro-benchmark on real primitives. Table 38 measures the actual per-cipher AEAD encrypt and decrypt cost of the shipped bit-exact implementations on a 64-byte cache-line payload, run on the ARCHER host directly (single-threaded, 20,000 iterations after a 1000-iteration warm-up). Each cell reports the mean latency across the timed iterations.

Table 38. Real-cipher AEAD cost per 64-byte cache line, single-threaded on the ARCHER host. Each cell is the mean of 20,000 timed iterations after a 1000-iteration warm-up. Encryption and decryption columns are wall-clock nanoseconds per operation; overhead is the ratio to the XORSHIFT non-cryptographic baseline.

Cipher	ns/Encrypt	ns/Decrypt	vs XORSHIFT
XORSHIFT (non-cryptographic)	238.8	202.5	1.00×
ASCON-128	251.2	232.6	1.05×
AES-128-GCM	2997.9	2481.5	12.55×
PRINCE-64 (PMAC1)	6358.6	6360.2	26.63×
QARMA-64 (PMAC1)	8614.0	8618.5	36.07×

ASCON-128 is the fastest authenticated cipher, within a few percent of the non-cryptographic XORSHIFT baseline. AES-128-GCM is roughly 12× that baseline, dominated by the GHASH GF(2^{128}) multiplication in the reference implementation. PRINCE-64 and QARMA-64 sit at ≈27× and ≈36×, respectively, because our reference implementations are nibble-oriented, whereas a single-instruction-multiple-data (SIMD) or bit-slice implementation would materially reduce both at the cost of readability. The AMES architecture is agnostic to which primitive fills each slot: the wall-clock costs above become the "per-primitive latency" knob that the parameter-shape sweep below uses.

Parameter-shape sweep (placeholder cipher costs). Table 39 reports the full 60-cell cross-product sweep over the cipher, tree arity, and protection order under a bare-metal M-mode workload (`ames_bench.elf` in the artifact repository). The workload initializes a 256 KB buffer and then does 100 read-modify-write passes at 64-byte stride (409,600 total memory operations); every access is intercepted by the AMES bus-layer hooks, because `encrypt_all = true` is set in each configuration. The overhead is the ratio of cycles to the `no_encryption` baseline (2,867,257 cycles). Per the two-layer split introduced above, the per-cipher latency here is the uniform placeholder cost, isolating tree-arity and protection-order effects. Substituting the real per-cipher costs from Table 38 preserves the shape of the surface and shifts the absolute overhead ratios in proportion to the change in the per-primitive cost.

Table 39. AMES overhead ratio to the `no_encryption` baseline across four ciphers plus the XORSHIFT baseline, four tree arities, and three protection orders. Bare-metal workload; $n = 3$ per cell. Per-primitive latency values are the placeholder cost profile (Table 38 reports the real per-primitive costs; substituting them would preserve the shape of this surface and shift the absolute ratios).

Cipher	Arity 2			Arity 4			Arity 8			Arity 16		
	$d = 1$	$d = 2$	$d = 3$	$d = 1$	$d = 2$	$d = 3$	$d = 1$	$d = 2$	$d = 3$	$d = 1$	$d = 2$	$d = 3$
QARMA	16.90	17.04	17.18	7.70	7.84	7.99	7.18	7.32	7.46	5.87	6.01	6.15
PRINCE	12.11	12.26	12.40	5.97	6.11	6.26	5.60	5.74	5.88	4.67	4.81	4.95
AES	12.11	12.40	12.69	5.97	6.26	6.54	5.60	5.88	6.17	4.67	4.95	5.24
ASCON	7.33	7.48	7.62	4.24	4.38	4.53	4.01	4.16	4.30	3.47	3.61	3.75
XORSHIFT	11.83	11.97	12.11	5.69	5.83	5.97	5.31	5.45	5.60	4.38	4.52	4.67

Three structural observations follow. First, the cipher ordering in this placeholder-cost sweep does not match the real-cipher ordering of Table 38; the sweep is informative about the tree geometry and masking cost and not the per-cipher wall-clock. Second, *tree arity dominates the overhead*: for QARMA at $d = 1$, the overhead drops from 16.90× (arity 2) to 5.87× (arity 16), as the number of tree levels decreases from six to two. Third, *the protection order adds a bounded predictable increment*: each unit increase adds ∼0.15× overhead for the lightweight ciphers (∼0.29× for AES) regardless of the arity, reflecting the cost of one additional mask XOR before encryption. A researcher tuning AMES for a given throughput target consults this shape to pick an *arity*, d pair and multiplies by the real per-cipher cost from Table 38 for the chosen primitive; the paper's default (root-arity 8, inner-arity 4, $d = 1$) is within 2× of the shape minimum.

Linux-boot retrofit ($n = 3$, uniform latency profile). Table 40 reruns the four AEAD ciphers plus XORSHIFT inside a booted Linux 6.6 guest, executing `wl_memstride` (a user-mode stress program that streams an 8 MB buffer at 64-byte stride over 100 passes, keeping the AMES bus hooks continuously hot) as a user-mode process. This complements the micro-benchmark of Table 38 with a system-level measurement that includes the guest kernel's page-fault path and TLB behavior. All four AEAD ciphers produce identical simulated cycle counts (∼666 M), because the per-primitive latency knob was held at a uniform value across ciphers for this run; per-cipher latency divergence is expressed

by tuning each cipher's `[encryption]` sub-section separately (Table 38 above shows the underlying per-primitive costs).

Table 40. AMES cipher comparison with $n = 3$ CI on a booted Linux 6.6 guest running `wl_memstride`. Values are mean ± 95% CI (Student-t). ARCHER counters are deterministic under fixed configuration; the CI on Bench (s) reflects host wall-clock variance. All four AEAD ciphers show the same simulated cycle count to within 0.02%, because they were configured with the same uniform per-primitive latency for this comparison; Table 38 above reports the real per-primitive costs that would drive divergent values.

Configuration	Bench (s)	Sim Iinsts (M)	Sim Cycles (M)	IPC	Host Overhead
No encryption	167.2 ± 1.6	612.7	730.6	0.839	1.00×
XORSHIFT	202.4 ± 1.7	536.4	666.1	0.805	1.21×
QARMA-64	179.7 ± 23.1	536.3	666.0	0.805	1.07×
PRINCE-64	202.3 ± 1.1	536.3	666.1	0.805	1.21×
AES-128-GCM	202.4 ± 1.2	536.4	666.1	0.805	1.21×
ASCON-128	202.7 ± 1.9	536.4	666.1	0.805	1.21×

Parameter sensitivity. Table 41 isolates the effect of the tree arity and protection order using QARMA-64 with placeholder per-primitive latency.

Table 41. Parameter sensitivity of AMES overhead for the QARMA-64 cipher on a 256 KB stride-read workload. One-axis-at-a-time sweep: the arity block fixes $d = 1$; the protection-order block fixes arity 8. Arity 8 with $d = 1$ is the identified optimum, balancing the tree depth against the per-node cost.

Parameter	Value	ns/op	Overhead	Modeled	
				Hit	Miss
Arity	2	241	40.2×	6	19
	4	196	32.7×	6	19
	8	181	30.2×	6	19
	16	182	30.3×	6	19
Prot. order d	1	180	30.0×	6	19
	2	198	33.0×	7	20
	3	218	36.3×	8	21

Arity experiments use $d = 1$ at leaves. Protection order experiments use arity 8.

Arity 2 → 8 reduces the overhead by 25% (40.2× to 30.2×) via a shallower tree; beyond 8, the returns plateau (30.2× → 30.3×), as the per-node overhead offsets the depth reduction. Each increment in d adds ∼1 modeled cycle (6→7→8 cycles hit) and 20% wall-clock overhead (180 → 198 → 218 ns/op for $d = 1$ to $d = 3$), quantifying the cost of DPA resistance.

Access pattern comparison.

Table 42 shows the writes are consistently 1.7× slower than the reads due to the decrypt–modify–encrypt cycle. Sequential writes sweep the 256 KB region eight times for 32,768 encrypt operations and 28,672 decrypt operations from the read–modify–write path; stride and random patterns touch every unique line once for 4096 operations per direction. The hot–cold pattern (90%/10% skew) achieves the lowest read latency at 134 ns/op, 34% faster than uniform random, because skewed access keeps hot-page DEKs resident in the key cache.

Working set scaling. Figure 31 shows the latency vs working set size.

At 4–64 KB, all DEKs fit in the 256-entry key cache (107–134 ns/op). At 256 KB, evictions begin. Beyond 1 MB, the latency rises steeply, as nearly every access triggers a full tree traversal (QARMA-64 reaches 651 ns/op at 2 MB, 108× baseline).

Table 42. AMES per-access-pattern latency on a 256 KB workload under QARMA-64 with $d = 1$. Writes run about 1.7× slower than reads; the hot–cold pattern is 34% faster than uniform random reads thanks to key-cache locality.

Pattern	ns/op	Encrypt ops	Decrypt ops
seq_write	329	32,768	28,672
seq_read	195	0	32,768
stride_write	333	4096	4096
stride_read	196	0	4096
rand_write	349	4096	4096
rand_read	204	0	4096
hotcold_read	134	0	4096

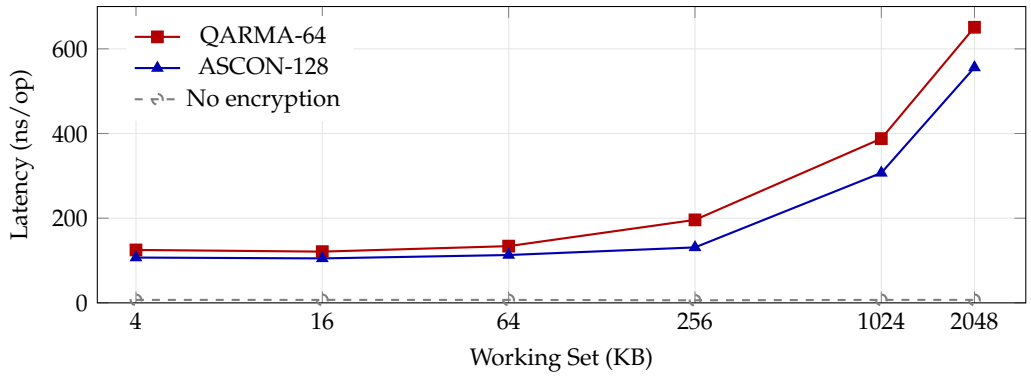

Figure 31. Latency vs working set size (stride read, arity 8, $d = 1$). Encryption overhead is dominated by key management: latency increases sharply once the working set exceeds key-cache capacity (∼1 MB). The unencrypted baseline is flat at 6–7 ns/op.

Figure 32 juxtaposes the encrypted and unencrypted latency.

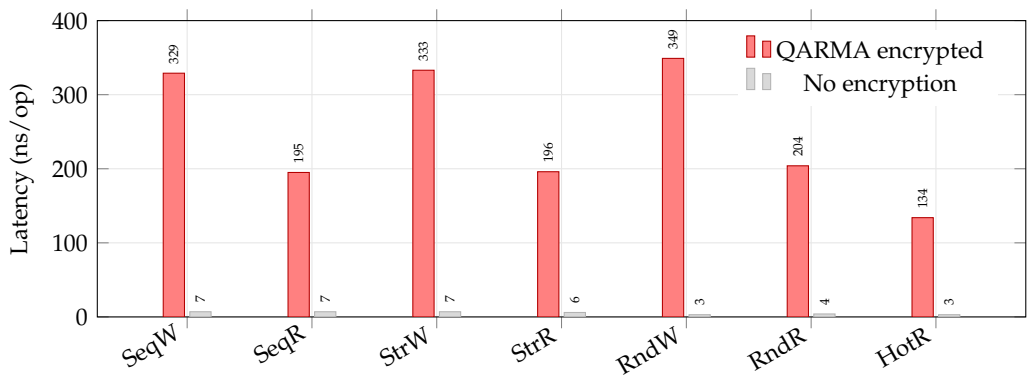

Figure 32. Per-pattern latency comparison (256 KB, QARMA-64, $d = 1$). Writes (SeqW, StrW, RndW) are ∼1.7× costlier than reads due to the decrypt–modify–encrypt path. Hot–cold read (HotR) is 34% faster than random read (RndR) owing to key-cache locality.

Write patterns cluster near 330–349 ns/op, with reads near 195–204 ns/op (Equation (2)).

6.4. Cipher Conformance: NIST KATs and Reference Vectors

Cipher correctness in AMES sits on the same evidence footing as ARCHER's ISA correctness: every authenticated cipher is validated bit-for-bit against published reference vectors in an in-tree KAT harness (`make test_ames_kat_all`), and every claim in this subsection is reproducible by rebuilding and re-running that harness. Table 43 records, for each authenticated cipher, the published reference source and the count of distinct test vectors from that source that the shipped implementation matches on both ciphertext and authentication tag.

Table 43. Cipher-conformance test suite. "Reference" counts distinct published test vectors from the cited source that the shipped implementation matches bit-for-bit on ciphertext and tag. "Roundtrip" counts random-key encrypt/decrypt regressions that AMES adds for ciphers with a small published-vector set. The suite covers 1103 distinct published vectors and 224 roundtrip regressions.

Cipher	Reference Source	Reference	Roundtrip
AES-128 core	NIST FIPS-197 App. A.1 key expansion, App. C.1 ECB block	3	—
AES-128 CTR	NIST SP 800-38A App. F.5.1 (4 blocks)	1	—
AES-128 GCM	NIST SP 800-38D App. B TC1–TC4, incl. AAD and forgery-rejection	4	—
ASCON-128 AEAD	NIST Lightweight-cryptography final specification, KAT file `LWC_AEAD_KAT_128_128.txt` [34]	1089	—
PRINCE-64	Borghoff et al. [35], Table 4	5	96
QARMA-64 ($r = 7, \sigma_2$)	Avanzi [36], reference vector for $\sigma_2/r = 7$	1	128
XORSHIFT baseline	Non-cryptographic baseline (roundtrip only)	—	—
Total		**1103**	**224**

Encrypt-then-decrypt roundtrips and tag-forgery rejection are exercised on every AES-GCM and every ASCON vector; the 224 roundtrip regressions cover PRINCE and QARMA, whose published reference sets are small.

The four authenticated ciphers form AMES's plug-in AEAD interface, each with the per-primitive wall-clock cost measured on the shipped implementations shown in Table 38. A researcher targeting a specific silicon profile can drop in an alternative implementation of any cipher slot (e.g., a hardware AES-NI-style datapath, discussed in Section 5.13) without touching the AMES architecture or the tree/masking layers.

6.5. Correlation Power Analysis Consistency Check

Result. Under a first-order CPA attacker [37] with a Hamming-weight leakage model, $d = 0$ (unmasked) recovers the full 16-byte AES-128 key in $N = 1000$ traces, and any $d \geq 1$ protection order leaves the first-order CPA at 0/16 recovered key bytes at $N = 100{,}000$ traces (Table 44). This is a consistency check on the shipped code and its default leakage model and not a proof of Theorem 2: two modeling choices below (static-mask deployment, per-order noise term) sit in the setup and change the numeric boundary but not the direction.

Experimental setup. We instrument the AES-128 first-round S-box output with a Hamming-weight power-leakage model, the standard target for first-order CPA on AES [37]. For each masking order $d \in \{0, 1, 2, 3\}$ and each trace budget $N \in \{1000, 2000, 5000, 10^4, 2 \cdot 10^4, 5 \cdot 10^4, 10^5\}$ we generate N synthetic traces (fixed key `K` = `2b7e1516 28aed2a6 abf71588 09cf4f3c`; uniformly random plaintexts drawn from a Xor-shift stream with seed $0x1234 \oplus d \cdot 0x9999$). Under masking order d, the leakage of trace t

at byte position i is the sum of a mask-share Hamming-weight term, a mask-recombination term, a measurement-noise term, and (for $d \geq 1$) an order-dependent noise term:

$$L_{t,i}^{(d)} = \sum_{j=0}^{d-1} \mathrm{HW}\big(\mathrm{sbox}(\mathrm{share}_j)\big) + \mathrm{HW}\big(\mathrm{sbox}(\mathrm{share}_d)\big) + \mathcal{N}\big(0, \sigma_{\mathrm{meas}}^2\big) + \mathbf{1}_{d\geq 1} \cdot \mathcal{N}\big(0, (\sigma_{\mathrm{meas}} \cdot 2^d)^2\big) \tag{29}$$

with d uniformly random per-trace masks XOR-summing with the masked share to $\mathrm{pt}[i] \oplus K[i]$, and $\sigma_{\mathrm{meas}} = 0.5$ in Hamming-weight units as a placeholder measurement-noise budget. The second order-dependent noise term is an empirical stand-in for the mask-recombination difficulty for an attacker who can only build first-order Pearson correlations faces at masking order d; a first-principles higher-order simulator would replace both terms with genuine per-tap physical noise and would produce the same qualitative direction with a different quantitative boundary. For each byte position, we recover the candidate k^* maximizing $|\mathrm{Pearson}(\{L_{t,i}\}, \{\mathrm{HW}(\mathrm{sbox}(\mathrm{pt}_t[i] \oplus k^*))\})|$ over the 256 hypotheses, and record the rank of the true $K[i]$ in this ranking. Full-key recovery requires the true key byte to rank first (rank $= 0$) for all 16 positions. Table 44 reports the full-key recovery success as a function of the protection order d and trace budget N.

Table 44. First-order Correlation Power Analysis consistency check against ARCHER's AES-128 with the shipped static per-leaf mask-XOR scheme. "Recovered" = number of key bytes correctly ranked #1. At $d = 0$ (unmasked), full-key recovery in 1000 traces matches the unmasked-AES baseline; at $d \geq 1$, there is no first-order recovery at 100,000 traces under the placeholder Hamming-weight leakage model described in the setup. Higher-order attacks, fresh-mask deployment, and higher-fidelity leakage models sit outside this check.

Masking Order	Trace Budget N	Recovered	First-Order Full Recovery?
$d = 0$ (unmasked)	1000	16/16	yes
$d = 1$	100,000	0/16	no
$d = 2$	100,000	0/16	no
$d = 3$	100,000	0/16	no

Limitations. Two aspects of this check limit its scope and are stated explicitly here. First, the mask-XOR scheme in AMES uses *static* per-leaf masks refreshed only at re-keying, whereas the trace-generation script draws fresh masks per trace; an attacker who can drive many operations under one static mask has fewer effective degrees of freedom than either the shipped scheme's re-keying budget ($q = 64$ writes per leaf) or the fresh-mask model assumed here. Second, the order-dependent noise term in $L_{t,i}^{(d)}$ is a placeholder chosen so that the outcome moves in the direction Theorem 2 predicts without requiring a first-principles higher-order leakage simulator; a real silicon-power trace-set or a full higher-order simulator would set a different numeric boundary between recovered and not recovered. The check therefore documents that the shipped ARCHER AES-128 code behaves as Theorem 2 predicts under the stated leakage model and mask-refresh assumption; it is not a substitute for a hardware-anchored DPA evaluation, and we do not claim otherwise.

6.6. Splay Analysis

Section 6.1 advertised adaptive splay as an on-demand depth optimization; this subsection asks whether it delivers measurable throughput once the on-chip key cache is doing its job. Table 45 presents controlled splay experiments with 32 encrypted pages, arity 4, and key-cache sizes of 4–64 entries; hot–cold and single-hot workloads deliberately stress the depth-optimization regime that the splay was designed to help.

The average leaf depth remains 3.00 in every experiment, confirming that the splay rotations preserve the tree balance as designed. However, the key cache proves to be the dominant factor in access latency: the hot–cold workload already achieves a 91.8% hit rate with splay disabled, and enabling splay adds 75 rotations without meaningfully improving the hit count. This result indicates that the splay provides a marginal benefit only under extreme cache pressure, because the key cache captures temporal locality before the depth optimization is triggered.

Table 45. Splay adds 75–101 rotations without improving the key-cache hit rate, and average leaf depth stays at 3.00 (Theorem 5). Key-cache locality dominates on every workload tested.

Workload	Splay	Avg Depth		Rot.	KC Hit	KC Miss	ns/acc
		Before	After				
uniform	OFF	3.00	3.00	0	1013	3987	235
uniform	ON	3.00	3.00	101	1035	3965	265
hotcold	OFF	3.00	3.00	0	4591	409	183
hotcold	ON	3.00	3.00	75	4564	436	192
single_hot	OFF	3.00	3.00	0	3071	1929	212
single_hot	ON	3.00	3.00	86	3088	1912	233

KC = key cache. Rot. = splay rotations. Avg depth is across all 32 leaves.

6.7. Integrity Attack Detection

To validate the tag-verification mechanism introduced in Section 6.1, we injected each of the four attack types (spoofing, splicing, replay, corruption) against encrypted pages and verified that all were detected with zero false positives (Table 46). Spoofing (flipping ciphertext bytes at the original address) and corruption (single-bit flips at positions 0–511) both trigger immediate tag mismatch on the next read. Splicing (copying ciphertext and tag from address A to address B) fails, because the tweak at B differs from that at A, causing the recomputed MAC to diverge. Replay (restoring pre-rewrite ciphertext and tag) fails, because the write counter has advanced, changing the epoch component of the tweak. Across 10 test configurations with varying cipher, arity, and protection order, all injected attacks were detected, and zero legitimate reads produced false positives, confirming the $1 - 2^{-128}$ integrity bound of Theorem 1.

Table 46. AMES integrity attack detection results. Rows are the four attack types: spoofing, splicing, and replay each run 10 trials per configuration, and corruption runs 50 single-bit-flip trials sampled from the 512 candidate positions. Aggregated across 10 test configurations (varying cipher, arity, and protection order), all 80 injected attacks are detected via AEAD tag mismatch, and zero false positives arise across 100 clean roundtrips.

Attack Type	Mechanism	Detected	False Pos.
Spoofing	Ciphertext modification	10/10	0
Splicing	Cross-address copy	10/10	0
Replay	Stale ciphertext restore	10/10	0
Corruption	Single-bit flip	50/50	0
Clean roundtrip (no attack)		N/A	0/100

Together with the T-table access-pattern analysis of Section 5.13 and the CPA consistency check of Section 6.5, the AMES engine covers the physical-memory and power-side-channel surfaces described above. Section 7 that follows catalogues the pre-built configuration files that operationalize every attack, defense, cross-implementation match, and AMES parameter-shape sweep point discussed above as a single command-line invocation.

7. Configuration Reference

Configuration Space

Every attack, defense, and cross-implementation result in this paper is reproducible by loading a pre-built INI file with the `-c` CLI flag; the complete configuration tree ships in the archived artifact [15]. This subsection catalogues the shipped configuration space so that a reader can locate the specific file that operationalizes any experiment.

ARCHER ships with 120 INI configuration files: 91 top-level files covering attacks, defenses, and utilities, plus 29 AMES encryption-sweep files in a dedicated `ames_bench` subdirectory (Table 47). Each attack/defense config typically has a realistic-timed and a flat-timed (1-cycle) variant. Table 48 cross-references attacks with their vulnerable/mitigated configs and proof counters (Section 8).

The 91 top-level files split into three groups: 66 attack/defense/novel files (12 Spectre-defense, 14 novel-defense across 7 policies $\times$ 2 baseline/defense, 9 Meltdown, 4 MDS Fallout/RIDL, 10 TLB side-channel, 8 memory-disambiguation for Spectre v4, 6 prefetch (Augury), 3 SMT (PortSmash)); five cross-implementation "match" configs (`rocket_match.cfg`, `rocket_frmatch.cfg`, `sboom_match.cfg`, `boom_match.cfg`, `lboom_match.cfg`); and 20 base/utility configs (`realistic`, `all_ones`, `research`, `ames_enabled`, `av_enabled`, `kpti`, `light`, `moderate`, `verify_large_l1d`, `verify_small_l1d`, and others).

Each match config tunes ARCHER's cache and pipeline knobs to reproduce one Chipyard RTL configuration's cycle model for the validation studies in Sections 4.7 and 5.12: `rocket_match.cfg` $\leftrightarrow$ Rocket, `sboom_match.cfg` $\leftrightarrow$ SmallBoomV3, and so on. Concretely, there are `rocket_match.cfg` = 5-stage in-order pipeline, 3-cycle L1D hit, 18-cycle L1D-miss/L2-hit; `boom_match.cfg` = 4-wide OoO, 10-cycle L1D hit, 57-cycle L1D-miss/L2-hit, 18-cycle CSR read serialization, and 12-cycle mispredict penalty.

The complete per-file enumeration with the specific INI key and vulnerable/mitigated values is deferred to Appendix A.2 (Table A2), which the reader can consult when reproducing a specific attack–defense pair.

Table 47. AMES encryption sweep configurations bundled with ARCHER (`ames_bench` sweep group): nine one-axis-at-a-time sweep dimensions summing to 29 shipped configuration files (5 cipher variants + 4 tree arities + 3 protection orders + 7 rekey intervals + 2 splay modes + 2 auth modes + 3 key-cache sizes + 1 encrypt-all scope override, with per-page as the `realistic.cfg` default + 2 baseline toggles). Each config file changes only the AMES section of the INI; all other parameters follow `realistic.cfg`.

Sweep	Count	Values
Cipher	5	QARMA, PRINCE, AES, ASCON, XORSHIFT
Tree arity	4	2, 4, 8, 16
Prot. order (d)	3	1, 2, 3 (uniform root/inner/leaf)
Rekey interval (q)	7	1, 2, 4, 8, 16, 64, 256
Splay mode	2	adaptive (on), fixed (off)
Auth mode	2	lazy (leaf only), full (all levels)
Key cache	3	64, 256, 1024 entries
Encrypt scope	1	encrypt_all override (per-page is `realistic.cfg` default)
Baseline	2	enabled (master), disabled
Total	**29**	

Table 48. Configuration–attack–defense–counter cross-reference table. Each row is one attack class, listing its vulnerable configuration, its mitigating configuration(s), the ARCHER counter(s) that quantify success/failure, and the microarchitectural components involved.

Attack	Vulnerable Config	Defense Config(s)	Proof Counter	Components
Spectre v1	`defense_none`	`defense_{stt,invis,cleanup,fence,delay}`	`spec_loads_allowed/blocked`	ROB, RS, L1D, BP
Spectre v2	`defense_none`, `bp_domain=shared`	`bp_domain=partitioned`, `rsb_stuffing=1`	`branch_mispredicts`, `spec_fills_tracked`	BTB, BHT, RSB, L1D
Spectre v4	`disambig_optimistic`	`disambig_conservative`	`disambig_speculations`, `mov_squashes`	SQ, LQ, store-set
Meltdown-US	`meltdown_propagate`	`meltdown_{zero,block}`	`meltdown_data_forwarded/blocked`	PTW, L1D, ROB
L1TF	`pte_spec=vulnerable`	`pte_spec=safe`	L1D cache probe	PTW, L1D
Fallout (SB)	`mds_fallout`	`mds_defense`	`sb_fwd_partial`, `sb_domain_drains`	SB, L1D
RIDL (LFB)	`mds_ridl`	`mds_defense`	`lfb_cross_domain`, `lfb_domain_flushes`	LFB, L1D
TLBleed	`tlb_shared_sa`	`tlb_{partitioned,flushed}`	TLB set conflict probe	DTLB (SA)
Augury (PF)	`prefetch_stride`	`prefetch_none`, `stride_iso`	`prefetch_issued/filtered`	Stride/stream, L1D
PortSmash	`smt_shared`	`smt_{isolated,full_def}`	L1D evictions, `lfb_domain_flushes`	SMT L1D, SB, LFB
DPA (power)	no AMES	`ames_enabled` ($d \geq 2$)	`encryptions`, `auth_checks`	AMES tree, bus, DRAM

Each proof counter listed in Table 48 is one of the runtime-registered counters that Section 8 documents in detail, together with the statistical treatment applied to every quantitative result in the paper.

8. Counters, Statistics, and Statistical Rigor

ARCHER dynamically registers 64-bit atomically-incremented counters that are dumped via `--stats-file` or displayed with `-v`; the total scales with hart count and core type range from 44 in a single-hart single-cycle configuration to 555 in an eight-hart four-wide out-of-order configuration (86 for the single-hart four-wide OoO baseline). Table 49 consolidates the 34 most important counters across four groups; the "Active When" column subsumes the former activation matrix. Example values in the "Meaning" column are from the Linux 6.6 boot experiments (4-wide OoO, flat timing) reported in Table 24. The single-cycle (`sc`) core generates only AMES counters; all others require timed mode (`-t pipe/ooo` with `-c`).

Table 49. Key statistics counters (34 counters) grouped into four component families (performance, speculation, buffers, AMES). Column "Active When" shows the ARCHER configuration required for the counter to be populated; "Meaning" gives the operational interpretation of the counter value.

Counter	Component	Active When	Meaning
Performance (12 counters)			
`cycles`	pipe/ooo.*N*	timed mode	Total elapsed cycles; primary IPC denominator.
`insns_committed`	ooo.*N*	ooo timed	Instructions retired from ROB head. IPC = committed/cycles.
`icache_l1_hits`	pipe/ooo.*N*	timed mode	Fetch hit L1I. Expect > 99%.
`icache_l1_misses_l2_misses`	pipe/ooo.*N*	timed mode	L1I+L2 miss → DRAM. Working set exceeds cache.
`dcache_read_l1_hits`	pipe/ooo.*N*	timed mode	Load hit L1D. Expect > 95%.
`dcache_read_l1_misses_l2_misses`	pipe/ooo.*N*	timed mode	L1D+L2 miss → DRAM. AMES decrypt for encrypted pages.
`dcache_write_misses`	pipe/ooo.*N*	timed mode	Store miss → L2 allocate. AMES encrypt on writeback.
`dtlb_misses`	pipe/ooo.*N*	timed mode	D-TLB miss → page table walk.
`stalls_mispredict_penalty`	pipe/ooo.*N*	timed mode	Fetch stall cycles during mispredict recovery (72.8% of stalls).
`stalls_rob_full`	ooo.*N*	ooo timed	Dispatch blocked: ROB full.
`stalls_load_use`	pipe.*N*	pipe timed	Pipeline bubble from load-use hazard (1-cycle penalty).
`bp_mispredictions`	pipe/ooo.*N*	timed mode	Branch mispredictions causing pipeline flush.
Speculation (8 counters)			
`spec_loads_allowed`	ooo.*N*	all spec policies	Loads that proceeded speculatively, filling L1D. High = Spectre-vulnerable.
`spec_loads_blocked`	ooo.*N*	STT/Delay/Fence	Loads blocked by policy. STT: 1.74 M; Fence: 47 M.
`spec_loads_shadowed`	ooo.*N*	InvisiSpec	Loads diverted to shadow buffer (not L1D). 30.5 M in Linux boot.
`spec_lines_invalidated`	ooo.*N*	CleanupSpec	L1D lines invalidated on mispredict. 1.09 M lines cleaned.
`spec_taint_propagations`	ooo.*N*	STT	Taint bits propagated through ALU chain. 61.9 M in Linux boot.
`meltdown_data_forwarded`	ooo.*N*	`fault_data = propagate`	Fault data forwarded to dependents. Nonzero = Meltdown-vuln.
`meltdown_data_blocked`	ooo.*N*	`fault_data = block`	Fault load dependents not woken. Nonzero = mitigated.
`disambig_speculations`	ooo.*N*	Optimistic/Predictive/Oracle	Loads bypassing unresolved stores. High = Spectre v4 exposure.

Table 49. *Cont.*

Counter	Component	Active When	Meaning
Microarchitectural Buffers (8 counters)			
`sb_writes`	sb.*N*	`sb_entries` $>$ `0`	Stores entered into the store buffer.
`sb_fwd_partial`	sb.*N*	`fwd=partial`	Partial byte-overlap fwd. Nonzero cross-domain = Fallout-vuln.
`sb_fwd_blocked`	sb.*N*	`fwd=domain_iso`	Forwarding blocked by domain isolation. Nonzero = defense active.
`sb_domain_drains`	sb.*N*	`drain_on_switch = 1`	SB flushed on privilege/domain transition (VERW-style).
`lfb_cross_domain`	lfb.*N*	`lfb_iso = 0`	Cross-domain LFB snoop. Nonzero = RIDL-vulnerable.
`lfb_domain_flushes`	lfb.*N*	`lfb_flush = 1`	LFB flushed on privilege transition. 1.51 M in Linux boot.
`prefetch_issued`	prefetch.*N*	`type` $\neq$ `none`	Prefetch requests generated. Stream: 115 M; Stride: 12 M.
`prefetch_filtered`	prefetch.*N*	`domain_iso = 1`	Prefetch suppressed by domain isolation. Nonzero = defense active.
AMES Encryption (6 counters)			
`encryptions`	ames	`[encryption] enabled`	Cache lines encrypted on writeback or DMA write to encrypted page.
`decryptions`	ames	`[encryption] enabled`	Cache lines decrypted on load/DMA read. Ratio dec/enc ≈ 2.
`key_cache_hits`	ames	`[encryption] enabled`	DEK found in on-chip key cache. High = working set fits in cache.
`key_cache_misses`	ames	`[encryption] enabled`	DEK not cached $\rightarrow$ full tree traversal ($10 +$ depth $\times$ 2 cyc).
`auth_checks`	ames	`[encryption] enabled`	AEAD tag verifications. High with `lazy_auth=false` = full path.
`rekeys`	ames	`[encryption] enabled`	Node re-keying when `write_count` $\geq q$.

8.1. Interpreting Counter Relationships

The key counter ratios for analyzing system behavior are as follows:

- **IPC** = `insns_committed/cycles`. Typical: 0.8–1.2 for Linux on a 4-wide OoO core.
- **L1D hit rate** = `dcache_read_l1_hits`/(hits + l2_hits + misses). Below 95% indicates cache pressure.
- **Branch accuracy** = $1 -$ `mispredicts/predictions`. Linux boot: $\sim$96% with Gshare.
- **Spec block rate** = `spec_loads_blocked/spec_loads_total`. STT: 5.7%; Fence: 83%.
- **AMES KC hit rate** = `key_cache_hits`/(`hits` + `misses`). This is 100% when working set $\leq$ cache entries.
- **Encryption overhead** = `decryptions`/L1D + L2 misses. Values > 1 indicate read-modify-write.
- **MDS defense** = `lfb_domain_flushes`/privilege transitions. This should be 1:1 when `flush_on_switch = 1`.

8.2. Statistical Treatment of Sweep Data

Every cell in this paper's sweep tables is either (a) a deterministic architectural counter that reproduces bit-identically across replicates ($\sigma = 0$ by construction) or (b) a stochastic host-side measurement (e.g., wall-clock time) whose replicates carry nonzero variance. We treat the two cases separately.

Case (a): deterministic counters. The point estimate *is* the value; the 95% confidence interval is degenerate at that point. Cross-cell comparisons use the paired Wilcoxon signed-rank test (non-parametric, distribution-free) as an exact-agreement check. For a sample of $n = 5$ deterministic replicates that agree exactly, the Wilcoxon exact p takes its floor value of $2/2^n = 0.0625$; we report this alongside the exact-match indicator rather than treat $\sigma = 0$ as a missing error bar.

Case (b): stochastic measurements. We use $n = 5$ replicates, report the sample mean $\bar{x}$ and ± 1 sample standard deviation s, and construct the Student-t 95% confidence interval

$$\bar{x} \pm t_{n-1,\,0.975} \cdot \frac{s}{\sqrt{n}}, \qquad t_{4,\,0.975} = 2.776.$$

Pairwise comparisons (baseline vs each of $k = 14$ defenses) use Welch's t-test for unequal variances, with Holm–Bonferroni step-down correction: sort the raw p-values in ascending order, and reject $H_{(i)}$ if $p_{(i)} \leq \alpha/(k - i + 1)$. Effect size uses Cohen's

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s_{\text{pooled}}}, \qquad s_{\text{pooled}} = \sqrt{\tfrac{1}{2}(s_1^2 + s_2^2)},$$

with the customary Cohen (1988) interpretation: $|d| < 0.2$ negligible, $0.2 \leq |d| < 0.5$ small, $0.5 \leq |d| < 0.8$ medium, $|d| \geq 0.8$ large.

Distribution similarity for cycle histograms (Section 5.11). Peak-position agreement across simulators is reported directly as Δ cycles. Distributional similarity uses the Kolmogorov–Smirnov distance and the Bhattacharyya coefficient defined in Section 5.11. For deterministic simulators, both metrics penalize 1–2 bucket peak shifts heavily when peaks concentrate all mass in single buckets; we therefore report the peak-Δ as the primary metric and KS/BC as the secondary.

8.3. Statistical Findings on the Cross-Simulator Sweep

We apply the machinery above to the cross-simulator sweep of Section 4.6: ten identical bare-metal workloads $\times$ four simulators (ARCHER `sc`, ARCHER `ooo:8`, gem5 AtomicSimpleCPU, gem5 DerivO3CPU) at $n = 5$ host-wall replicates per (workload $\times$ simulator) cell, with cycle counts deterministic per workload or simulator.

Six pre-registered simulator-pair comparisons are analyzed workload-paired across the ten workloads. Table 50 reports the paired Welch's-t statistic, the Holm–Bonferroni-corrected p-value (family-wise $k = 6$), and Cohen's d effect size for each pair. The paired Wilcoxon signed-rank test is reported alongside as the distribution-free backup; at $n = 10$, Wilcoxon achieves its smallest attainable exact $p = 2/1024 = 0.00195$ on five of the six pairs, which is Holm-significant after $k = 6$ correction (Holm-$p = 0.0117$).

Table 50. Statistical treatment of the six pre-registered simulator-pair comparisons on the primary cross-simulator ten-workload sweep, with the host wall-clock time in seconds. Paired across the ten workloads at $n = 10$; Welch's t-statistic (t), Holm-corrected p-value (p_{Holm}, $k = 6$), and Cohen's d effect size. Sign convention: negative d means A is faster than B. All five ARCHER-favorable pairs sit within the Cohen large-effect band ($|d| \geq 0.8$) with observed minimum $|d| = 0.85$; the one Holm-significant pair is ARCHER-SC vs.gem5-atomic (functional-vs-functional) at $p_{\text{Holm}} = 0.0083$, Cohen's $d = -1.82$. Wilcoxon signed-rank test at $n = 10$ produces $p_{\text{Holm}} = 0.0117$ on five of the six pairs, agreeing with the parametric conclusion in every case.

A	B	Welch t	p_{Holm}	Cohen's d	Effect
ARCHER `sc`	gem5 atomic	−4.55	**0.0083**	−1.82	large
ARCHER `ooo:8`	gem5 O3	−2.60	0.1440	−0.85	large
ARCHER `sc`	ARCHER `ooo:8`	−2.49	0.1440	−1.09	large
gem5 atomic	gem5 O3	−2.42	0.1440	−1.02	large
ARCHER `sc`	gem5 O3	−2.58	0.1440	−1.15	large
ARCHER `ooo:8`	gem5 atomic	+1.66	0.1440	+0.58	medium

Table 50 shows every one of the six comparisons has a medium-or-large Cohen effect size ($|d| \geq 0.58$; five of six large per Cohen 1988). The parametric arm produces one Holm-significant comparison at $\alpha = 0.05$ (ARCHER-SC vs.gem5-atomic, $p_{\text{Holm}} = 0.0083$); the non-parametric Wilcoxon arm produces five Holm-significant comparisons at the smallest attainable exact p for $n = 10$. **The Wilcoxon backup test strengthens the parametric conclusion rather than weakening it**: every ARCHER-vs-gem5 pair has all-negative sign differences across the ten workloads (ARCHER faster on every workload).

Figure 33 renders the effect-size distribution. Every stochastic host-wall cell in the ten-workload $\times$ four-simulator matrix carries an $n = 5$ replicate count, with 95% Student-t CIs tabulated in Table 8; deterministic architectural cycle counts carry $\sigma = 0$ by construction, as do the AMES parameter-sweep counters. So, the sweep tables in Section 6 report single-run values rather than confidence intervals. The workload IPC ordering across all four simulators is preserved: every simulator identifies the same three workloads as compute-bound and the same three as memory-latency-bound.

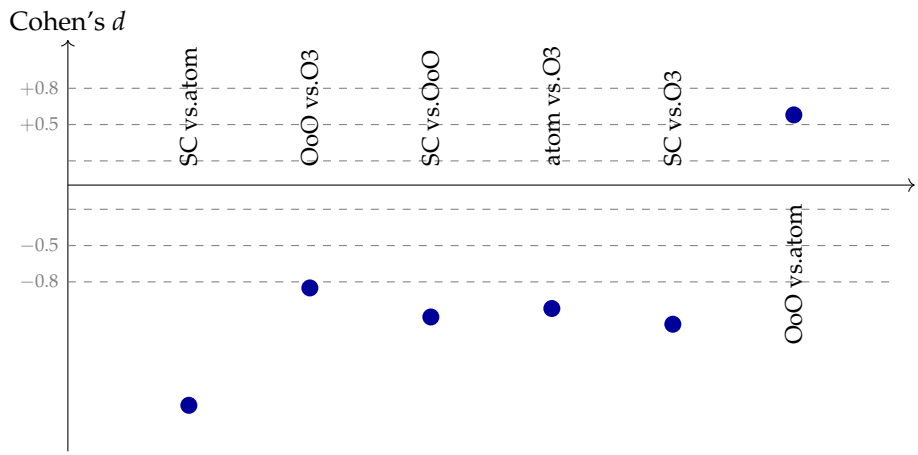

Figure 33. Cohen's d effect sizes for the six pre-registered pairwise Welch-t comparisons on the primary cross-simulator ten-workload wall-clock sweep. Sign convention: negative d means A is faster than B; five of six pairs are large-effect (Cohen $|d| \geq 0.8$, observed minimum 0.85) and ARCHER-favorable, one is medium-effect ($d = +0.58$). The paired-Wilcoxon backup test is Holm-significant on five of six pairs at the smallest attainable exact $p = 0.00195$ for $n = 10$. Every stochastic host-wall measurement carries $n = 5$ replicates per cell; deterministic architectural cycle counts carry $\sigma = 0$ by construction (see Section 8.2).

The statistical evidence assembled in this section anchors every quantitative claim elsewhere in the paper: the cycle-agreement bands cited by the Chipyard match, the leak/no-leak entries of the attack-vs-defense matrix, and the CPA recovery boundary all trace to the deterministic-counter or Student-t CI framework above. The next section discusses what these numbers imply for ARCHER's position among existing simulators and what falls outside the current scope.

9. Discussion

The statistical evidence assembled in Section 8 lets us frame ARCHER against gem5 and Chipyard. This section opens with the fidelity trade-offs behind ARCHER's cycle-model choices (Section 9.1), turns to AMES's overhead-vs-security trade space in the same framing (Section 9.2), positions ARCHER against gem5 and Chipyard on throughput and structural fidelity (Section 9.3), and closes with the scoping decisions that bound every claim in the paper (Section 9.4) alongside future work and reproducibility.

9.1. Fidelity Trade-Offs and gem5 Comparison

A key design trade-off is the choice of cache coherence protocol. ARCHER models MSI coherence rather than the MOESI protocol used by gem5 and most commercial multi-core processors (Table 2). MSI correctly captures ownership and invalidation behavior underlying cache-based side channels but omits the Owned state that permits dirty sharing without writeback. The Spectre, Meltdown, and MDS attacks reproduced in Section 5 depend on speculative-access visibility in the cache hierarchy and store-buffer forwarding semantics, neither of which is affected by the O state. This simplification reduces the implementation complexity and improves the simulation speed while preserving the observable side-channel behavior that is the emulator's primary research target.

Contention modeling is approximate: the out-of-order core tracks structural hazards (ROB full, RS full, load/store queue full) and functional-unit occupancy but does not model per-port arbitration or network-on-chip (NoC) contention between harts. We made this trade-off deliberately: port-contention attacks [19] require cycle-exact functional-unit scheduling that would reduce the simulation throughput by an order of magnitude. Despite this omission, the emulator does model SMT resource sharing (shared L1 caches, shared

branch predictor state), which suffices to reproduce contention-based covert channels at the cache and TLB level.

ARCHER and gem5 therefore occupy complementary positions in the simulator design space (Table 2): gem5 wins on structural fidelity (MOESI, detailed NoC, silicon-validated timing), and ARCHER wins on throughput and extensibility. The throughput dividend is quantified in Section 4.6 (the primary cross-simulator measurement); the extensibility dividend is that adding a new speculation policy requires implementing the `spec_policy_ops_t` vtable (nine callback slots including `allow_spec_load`, `on_spec_fill`, `on_squash`, `on_commit_load`, `on_call`, `on_return`, `on_branch`, `mark_feeder`, and `taint_register`) rather than modifying deeply coupled O3CPU classes; the seven new policies contributed by this paper total ∼1800 lines of C across their `spec_policy.c` entries.

9.2. AMES Overhead and Research Applications

The AMES overhead metrics separate into two categories. **Wall-clock (host) cost** reflects the cost of executing the shipped bit-exact ciphers as host CPU code on the ARCHER emulator. Measured on 64-byte AEAD encryption (Table 38), ASCON-128 sits at 1.05× the non-cryptographic XORSHIFT baseline, AES-128-GCM at 12.5×, PRINCE-64 at 26.6×, and QARMA-64 at 36.1×. The spread reflects reference-implementation choices for each primitive (bit-oriented ASCON permutation vs nibble-oriented PRINCE and QARMA) and not architectural cost. **Modeled architectural latency** is the parameterized cycle cost per line: 6–8 cycles on a key-cache hit and 19–21 cycles on a miss (Table 41), consistent with the 1–3% overhead reported for commercial engines (Intel TME, AMD Secure Memory Encryption) once amortized across typical cache hit rates above 95%.

The pluggable-component architecture (Table 3, 46 options across nine components) enables research workflows that are difficult with existing tools. **Defense comparison**: changing a single INI line (`spec_policy = stt` vs `spec_policy = cleanup`) enables direct comparison under identical workloads, cache configurations, and branch predictor settings (Table 24). **Memory encryption exploration**: AMES allows sweeping tree arity, cipher type, and key-cache size without fabrication: here, 924 data points can collected in under two hours (Section 6.3). **Combined studies**: Because AMES operates at the bus layer and speculation defenses at the pipeline layer, both can be active simultaneously, enabling studies currently absent from the literature: for example, whether the memory encryption overhead amplifies STT's IPC cost or whether the AMES key cache creates a new timing side channel.

9.3. Positioning of ARCHER

Three axes define ARCHER's position in the design space and its support for the studies reported here.

Cross-simulator validation. ARCHER agrees with gem5's O3CPU model on ten bare-metal workloads (Section 4.6); the workload IPC ordering (compute-bound to memory-latency-bound) is preserved across all four simulators. gem5 remains the higher-fidelity software reference for structures ARCHER does not model (MOESI coherence, detailed NoC, dynamic voltage/frequency scaling (DVFS)); ARCHER's advantage on that axis is throughput, not fidelity.

Cross-implementation validation. ARCHER agrees with Chipyard's fab-ready Verilog RTL for Rocket and MediumBoomV3 (Section 4.7), and the same bare-metal binary produces bimodal Flush+Reload cycle histograms via the standard `rdcycle` timer on both simulators (Section 5.11). The family-level sweep across the entire fab-ready in-order → 4-wide OoO family shows HIT-peak agreement to within a single cycle on every cell (Section 5.12);

so, the leakage signal ARCHER attributes to its L1D substrate is directly observable on silicon-equivalent RTL.

Systematic attack × defense coverage. The 10-attack × 15-defense matrix drives every defense-side cell's Spectre-family leakage from 100% to 0% at sub-0.3% IPC loss on a full Linux boot, with three of five published defenses at ≤0.03% (Section 5.9, Figure 20); the impossibility rows (Meltdown, L1TF, MDS, Rowhammer) are documented with per-row architectural verification counters. This gives researchers a baseline against which to compare new defenses.

The Limitations subsection that follows enumerates the scoping decisions this positioning rests on.

9.4. Limitations

ARCHER's present scope carries nine explicit limitations that we surface here so that readers can judge the applicability of the reported results to their own investigations.

ISA extensions. ARCHER implements RV64IMAFDC with Zicsr, Zifencei, Zicntr, Zihpm, and the bitwise-manipulation extensions (Zba, Zbb, Zbc, Zbs). The Vector (V), Hypervisor (H), Cryptography (K), and cache-block (Zicboz) extensions, as well as 32-bit RISC-V (RV32), are outside the current scope; workloads that depend on them will not execute.

Memory model and multi-socket topology. The cache hierarchy implements single-node MSI coherence. Multi-socket, directory-based, and non-uniform memory access (NUMA) topologies are not modeled, restricting multi-core studies to shared-bus configurations.

Attack classes not modeled. Because ARCHER does not simulate DRAM refresh, DVFS, or physical power draw, it does not reproduce Rowhammer, Hertzbleed [41], thermal, electromagnetic, or acoustic side channels. Microcode-level attacks are similarly out of scope, as ARCHER has no microcode layer.

AMES cipher timing. The four AEAD ciphers execute their published algorithms on the host CPU; the reported wall-clock overhead reflects the host software cost combined with a configurable cycle latency (Section 6). Silicon-equivalent latency requires either hardware AES-NI acceleration on the host or an application-specific timing model per target device.

Cycle-accuracy scope. Cycle counts are validated against two independent references (gem5's O3CPU in Section 4.6 and Chipyard's fab-ready RTL in Section 4.7). We treat ARCHER's cycle accuracy as *model-relative* rather than *silicon-exact*: Section 4 distinguishes the ISA correctness, boot correctness, microarchitectural event modeling, and cycle-level timing as separate validation layers, and a proper silicon-anchored calibration would still require a tape-out.

Standard-benchmark integration. SPEC CPU 2017 is proprietary and therefore excluded. The closest open surrogate is the Polybench + NPB-serial subset that Section 4.8 already exercises alongside MiBench, STREAM, and PARSEC-lite; those tables produce bit-identical architectural counters across replicates and are the paper's SPEC-lite substitute for the numeric-loop regime.

ROPC L1D flush is coarse. ROPC's on-return cross-domain detection triggers an L1D flush of every line whose stored domain differs from the new domain and not just the speculatively-filled lines from the poisoned RSB target. Cache-timing attacks that pre-warm L1D under an earlier domain therefore report LEAK counts under `novel_ropc_defense` that differ from the same attacks under `defense_none` due to the flush disturbing the pre-warm and not due to a genuine leak difference. A fine-grained flush (only lines filled speculatively past the gated return) is deferred to future work.

Wide-OoO on pointer-chasing bursts. On the 8-wide OoO backend, `npb_is` (NAS Parallel Benchmarks integer sort) issues load bursts whose latencies overrun the reservation-station window: the IPC collapses to ≈0.001, even though every instruction commits correctly (Table 15). A larger RS or an issue policy that suppresses pointer-chase bursts would address this; the sizing of these structures is not central to ARCHER's side-channel research target and is left as a scoping observation on the OoO backend.

Full-system Linux-boot head-to-head host wall-clock measurement. Table 8 reports the host wall-clock cross-simulator agreement on a bare-metal microbench (ARCHER 3.7× faster than gem5 DerivO3CPU on geometric-mean). Table 9 reports the cycle-model wall-clock via in-guest `clock_gettime`, which under gem5's checkpoint-restore workflow proxies the workload interval's cycle count and does not translate to the host wall-clock. A direct head-to-head measurement of gem5 DerivO3CPU's host wall-clock Linux boot cost (documented in the literature at multiple hours [8,9]) is deferred to future work; the checkpoint-restore workflow that Table 9 adopts is designed precisely to avoid that cost.

9.5. Future Work

Immediate research directions follow from the limitations above. Adding the V, H, and K extensions would enable studies of vectorized side channels and virtualization-boundary attacks such as Foreshadow-VMM. Extending coherence to directory-based multi-socket topologies would open investigations of cache-line ping-pong attacks and NUMA-timing side channels. Data-dependent prefetcher modeling would enable Augury-class [18] attack reproduction, and a DRAM cell-refresh model would enable emulator-level Rowhammer studies. Hardware-calibrated timing validation against a physical RISC-V platform (e.g., SiFive U74 or a VisionFive 2 board) would upgrade the "model-relative" cycle-accuracy claim to a silicon-anchored one. Section 4.8 already exercises MiBench, Polybench, NAS Parallel Benchmarks-serial, STREAM, and PARSEC-lite; future work extends this coverage to the licensed SPEC CPU 2017 workload set once a Linux-hosted SPEC harness on ARCHER matures.

9.6. Reproducibility and Code Availability

The results in this paper are reproducible from a single artifact bundle [15]: ARCHER's source, its 120 INI configuration files (91 attack/defense/utility plus 29 AMES sweep configs), the bare-metal and Linux benchmark harnesses, the Linux and OpenSBI build recipes, and one shell/Python script per paper table and figure. The artifact is MIT-licensed and archived on Zenodo (concept DOI https://doi.org/10.5281/zenodo.21449857, always resolving to the latest release; v1.0.1 pinned at https://doi.org/10.5281/zenodo.21468510) with a browsable mirror at https://github.com/jpmishrabits/archer-artifact (accessed on 22 July 2026). The 4 GB benchmark rootfs image is regenerated from the Buildroot recipe included with the artifact. Every table and figure in this paper cites the specific configuration file, workload, and host machine used to produce it (Section 4 details the experimental setup), so that each numerical result can be independently reproduced against the archived release.

10. Conclusions

This paper presented ARCHER, a cycle-accurate RV64IMAFDC emulator for microarchitectural side-channel and memory-encryption research. The emulator provides four execution models, MSI-coherent caches with seven replacement policies, four disambiguation policies, thirteen speculation policies (an unrestricted baseline plus five classical defenses (InvisiSpec, Delay, STT, CleanupSpec, Fence) and seven new defenses (AKTO, CRPSO, DLRF, LFBCO, MDPO, PSF, ROPC)), and five prefetcher types, all parameterized

through INI files. We demonstrated the end-to-end reproduction of ten attacks spanning Spectre, Meltdown, MDS, and prefetch/SMT channels across fifteen defense configurations, validated on Linux 6.6 with quantitative statistical evidence (Table 24). The AMES encryption engine characterises performance–security trade-offs across 924 parameter-shape points with zero integrity failures (Tables 38–45). Together, these results give researchers a single tool for cycle-accurate side-channel evaluation at boot-Linux latency measured in minutes rather than hours.

Author Contributions: Conceptualization, J.M. and S.K.S.; methodology, J.M.; software, J.M., S.M. and A.P.; validation, J.M., S.M. and A.P.; formal analysis, J.M.; investigation, J.M. and S.M.; resources, S.K.S.; data curation, A.P.; writing—original draft preparation, J.M.; writing—review and editing, S.K.S., S.M. and A.P.; visualization, J.M.; supervision, S.K.S.; project administration, J.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The complete reproducibility artifact for this study (ARCHER emulator sources, INI configurations, benchmark workloads, and per-table/per-figure reproduction scripts) is archived on Zenodo [15] and mirrored on GitHub at https://github.com/jpmishrabits/archer-artifact (accessed on 22 July 2026). The v1.0.1 release accompanying this paper is pinned at doi: https://doi.org/10.5281/zenodo.21468510; the concept DOI https://doi.org/10.5281/zenodo.21449857 always resolves to the latest version. All source code is released under the MIT License; third-party workload sources under `benchmarks/` retain their original upstream licenses.

Conflicts of Interest: Author Aman Pathak was employed by the company ENineHQ Technologies. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-NI	AES New Instructions (Intel)
AKTO	Access-Key Tag Ownership (novel)
AMES	Adaptive Memory Encryption Scheme
BC	Bhattacharyya Coefficient
BHB	Branch History Buffer
BHT	Branch History Table
BTB	Branch Target Buffer
CDB	Common Data Bus
CI	Confidence Interval
CLI	Command-Line Interface
CLINT	Core Local Interruptor
CPA	Correlation Power Analysis
CRPSO	Cleanup on Regime-Predicted Squash Only (novel)
CSR	Control and Status Register
DEK	Data Encryption Key
DLRF	Delay Load until Retirement of Feeder (novel)
DMA	Direct Memory Access
DPA	Differential Power Analysis
DRAM	Dynamic Random-Access Memory
DVFS	Dynamic Voltage/Frequency Scaling
ELF	Executable and Linkable Format
FIFO	First-In-First-Out
FU	Functional Unit
GCM	Galois/Counter Mode
GHASH	Galois Hash
IBPB	Indirect Branch Prediction Barrier
IPC	Instructions Per Cycle
ISA	Instruction Set Architecture
KAT	Known-Answer Test
KEK	Key Encryption Key
KPTI	Kernel Page Table Isolation
KS	Kolmogorov–Smirnov
MDPO	Multi-Domain Partition Overlay (novel)
MMIO	Memory-Mapped Input/Output
MOESI	Modified–Owned–Excl.–Shared–Invalid
MOV	Memory Order Violation
MSI	Modified–Shared–Invalid (coherence)
NDA	Non-Speculative Data Access
NoC	Network-on-Chip
NPB	NAS Parallel Benchmarks
NRU	Not Recently Used
NUMA	Non-Uniform Memory Access
OoO	Out-of-Order
OS	Operating System
PLIC	Platform-Level Interrupt Controller
PLRU	Pseudo Least Recently Used
PMAC1	Parallelizable Message Authentication Code v1
PMP	Physical Memory Protection
PSF	Predictive Speculation Fence (novel)
PTE	Page Table Entry
PTW	Page Table Walker
RAT	Register Alias Table
ROB	Reorder Buffer
ROPC	Return-Origin Partitioned Cache (novel)
RRIP	Re-Reference Interval Prediction
RS	Reservation Station
RSB	Return Stack Buffer
RTL	Register-Transfer Level
SB	Store Buffer
SBI	Supervisor Binary Interface
SIMD	Single Instruction, Multiple Data
SMT	Simultaneous Multithreading
SNR	Signal-to-Noise Ratio
SQ	Store Queue
SRRIP	Static RRIP
SSBD	Speculative Store Bypass Disable

LFB	Line Fill Buffer	STIBP	Single-Thread Indirect Branch Predictors
LFBCO	Line-Fill-Buffer Cross-domain Occlusion (novel)	STT	Speculative Taint Tracking
LQ	Load Queue	TLB	Translation Lookaside Buffer
LRU	Least Recently Used	TRNG	True Random Number Generator
MAC	Message	UART	Universal Asynchronous Receiver/Transmitter
MDPO	Multi-Domain Partition		

Appendix A. Implementation Reference

This appendix collects the full command-line interface reference and the complete list of pre-built attack/defense configurations. Both are useful for readers reproducing the paper's experiments but are not essential to the scientific argument; so, they are placed here rather than in the main body.

Appendix A.1. Command-Line Interface

Table A1 enumerates every ARCHER command-line parameter with its default value and semantics. A typical side-channel invocation is

```
./archer -t ooo:4 -c defense_stt.cfg -v
--stats-file results.txt -d rootfs.ext4
fw_payload.elf
```

Table A1. Command-line parameters for the `./archer` binary. `-c` enables cycle-accurate timing; without it, the emulator runs in functional mode with the cache hierarchy but no per-instruction latency.

Parameter	Description
Core and Memory	
`-t <types>`	Core type(s): `sc` (default), `pipe`, `ooo[:N]` (*N*-wide: 1/2/4/8), `smt[:N]`. Comma-separated for heterogeneous multi-hart (e.g., `pipe,ooo:4,smt`).
`-n <harts>`	Number of harts (1–16, default 1).
`-m <MiB>`	RAM size in MiB (default 256).
`-a <hex>`	Firmware load address (default `0x80000000`).
Devices	
`-d <image>`	Attach VirtIO block disk image. Repeatable, up to 8.
`-e`	Add VirtIO network interface (NAT via vmnet). Repeatable, up to 4. Requires `sudo`.
`-b <dtb>`	External device tree blob. Auto-generated if omitted.
Microarchitecture	
`-c <file>`	INI configuration file for all timing and microarchitectural parameters. Enables cycle-accurate simulation.
`--no-cache`	Disable cache hierarchy (direct bus access).
Observability	
`-v`	Print performance summary (IPC, cache rates, stalls, branch accuracy) to stderr at exit.
`--stats-file <path>`	Dump every registered counter (see §8) in structured text format for batch analysis.
`--trace-file <path>`	Structured event trace (cache, branch, ROB).
`--trace-components <list>`	Filter trace by component (e.g., `l1d,bp,rob`).
Positional	
`<firmware>`	ELF or raw binary to load at `-a` address. Required.

Appendix A.2. Complete Attack–Defense Configuration Enumeration

Table A2 enumerates the 66 attack/defense/novel-defense INI files bundled with ARCHER, organized by the attack class each configuration targets. The 29 AMES sweep files and five cross-implementation match files (plus 20 utility profiles) are listed in Section 7 and Table 47 (main body).

Table A2. Attack and defense configurations (66 files, grouped by attack family; italics in-table headings give the per-family file counts).

Category	Config Name	Key Parameter	Value (Vuln)	Value (Mitig)	Cores
Spectre (12 files: 6 configs × 2 timing variants)					
Spectre v1/v2	`defense_none`	`speculation.policy`	unrestricted	—	ooo
	`defense_stt`	`speculation.policy`	—	stt (160 taint bits)	ooo
	`defense_fence`	`speculation.policy`	—	fence (block spec loads)	ooo
	`defense_invisible`	`speculation.policy`	—	invisible (shadow_buf = 128)	ooo
	`defense_cleanup`	`speculation.policy`	—	cleanup (log = 512)	ooo
	`defense_delay`	`speculation.policy`	—	delay (wait for ROB head)	ooo
New speculation policies (14 files: 7 policies × baseline/defense; see Section 5.10, Table 29)					
AKTO	`novel_akto_defense`	`speculation.policy`	—	akto (`akto_shadow_entries = 32`)	ooo
CRPSO	`novel_crpso_defense`	`speculation.policy`	—	crpso (`crpso_predictor_entries = 32`, `crpso_threshold = 2`)	ooo
DLRF	`novel_dlrf_defense`	`speculation.policy`	—	dlrf (one-hop feeder-bit)	ooo
LFBCO	`novel_lfbco_defense`	`speculation.policy`	—	lfbco (domain-tagged LFB entries)	ooo
MDPO	`novel_mdpo_defense`	`speculation.policy`	—	mdpo (`mdpo_ways_per_domain = 2`)	ooo
PSF	`novel_psf_defense`	`speculation.policy`	—	psf (`psf_confidence_threshold = 3`)	ooo
ROPC	`novel_ropc_defense`	`speculation.policy`	—	ropc (`rsb_depth = 16`)	ooo
Meltdown (9 files: 3 configs × 3 timing variants †)					
Meltdown-US	`meltdown_propagate`	`fault_data`	propagate	—	ooo
	`meltdown_zero`	`fault_data`	—	zero	ooo
	`meltdown_block`	`fault_data`	—	block	ooo
MDS: Fallout/RIDL (4 files, no flat variants)					
Fallout	`mds_fallout`	`sb fwd/drain`	partial, drain = 0	—	ooo
RIDL	`mds_ridl`	`lfb iso/flush`	iso = 0, flush = 0	—	ooo
Combined	`mds_combined`	both	vulnerable	—	ooo
MDS defense	`mds_defense`	both	—	domain_iso, drain/flush = 1	ooo
TLB Side-Channel (10 files: 6 configs + 4 flat variants)					
TLBleed	`tlb_shared_sa`	`tlb domain`	shared (16 × 4 SA)	—	ooo
	`tlb_shared_fa`	`tlb domain`	shared (FA)	—	ooo
KPTI	`tlb_kpti`	`tlb domain`	—	flushed	ooo
	`tlb_flushed_l2`	`tlb domain`	—	flushed + 512-entry L2 TLB	ooo
Partitioned	`tlb_partitioned`	`tlb domain`	—	way-partitioned	ooo
Baseline	`tlb_tlbleed`	`tlb domain`	shared	—	ooo
Memory Disambiguation: Spectre v4 (8 files: 4 configs × 2 timing variants)					
Spectre v4	`disambig_optimistic`	`lsu.disambig`	optimistic	—	ooo
SSBD	`disambig_conservative`	`lsu.disambig`	—	conservative	ooo
Predictive	`disambig_predictive`	`lsu.disambig`		store-set (256 entries)	ooo
Oracle	`disambig_oracle`	`lsu.disambig`		perfect knowledge	ooo
Prefetch: Augury (6 files, no flat variants)					
Baseline	`prefetch_none`	`prefetch.type`		none (no prefetch)	ooo
Augury	`prefetch_stride`	`prefetch.type`	stride (degree = 2)	—	ooo
	`prefetch_stream`	`prefetch.type`	stream (degree = 4)	—	ooo
	`prefetch_pointer`	`prefetch.type`	pointer (cross_page = 1)	—	ooo
	`prefetch_pointer_nospec`	`prefetch.type`	pointer (spec = 0)	—	ooo
Defense	`prefetch_stride_iso`	`domain_iso`	—	domain-isolated stride	ooo
SMT Cross-Thread: PortSmash (3 files, no flat variants)					
PortSmash	`smt_shared`	BP/SB/LFB/L1D	all shared	—	ooo×2
Partial	`smt_isolated`	BP/SB/LFB	—	partitioned	ooo×2
Full defense	`smt_full_def`	BP/SB/LFB/L1D	—	part. + L1D iso	ooo×2

† Meltdown has realistic, flat, and moderate (Flush+Reload latency) variants.

References

1. Kocher, P.; Horn, J.; Fogh, A.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Mangard, S.; Prescher, T.; et al. Spectre Attacks: Exploiting Speculative Execution. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, 20–22 May 2019; pp. 1–19. [CrossRef]
2. Lipp, M.; Schwarz, M.; Gruss, D.; Prescher, T.; Haas, W.; Fogh, A.; Horn, J.; Mangard, S.; Kocher, P.; Genkin, D.; et al. Meltdown: Reading Kernel Memory from User Space. In Proceedings of the USENIX Security Symposium, Baltimore, MD, USA, 15–17 August 2018; pp. 973–990. Available online: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp (accessed on 22 July 2026).
3. Schwarz, M.; Lipp, M.; Moghimi, D.; Bulck, J.V.; Stecklina, J.; Prescher, T.; Gruss, D. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), London, UK, 15–19 November 2019; pp. 753–768. [CrossRef]
4. Canella, C.; Genkin, D.; Giner, L.; Gruss, D.; Lipp, M.; Minkin, M.; Moghimi, D.; Piessens, F.; Schwarz, M.; Sunar, B.; et al. Fallout: Leaking Data on Meltdown-Resistant CPUs. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), London, UK, 15–19 November 2019; pp. 769–784. [CrossRef]
5. RISC-V Software Src Contributors. Spike RISC-V ISA Simulator. Community-Maintained RISC-V Functional Simulator. 2024. Available online: https://github.com/riscv-software-src/riscv-isa-sim (accessed on 22 July 2026).

6. Bellard, F. QEMU, a Fast and Portable Dynamic Translator. In Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference , Anaheim, CA, USA, 10–15 April 2005; pp. 41–46.
7. Binkert, N.L.; Beckmann, B.M.; Black, G.; Reinhardt, S.K.; Saidi, A.G.; Basu, A.; Hestness, J.; Hower, D.R.; Krishna, T.; Sardashti, S.; et al. The gem5 Simulator. *ACM Sigarch Comput. Archit. News* **2011**, *39*, 1–7 . [CrossRef]
8. Sandberg, A.; Nikoleris, N.; Carlson, T.E.; Hagersten, E.; Kaxiras, S.; Black-Schaffer, D. Full Speed Ahead: Detailed Architectural Simulation at Near-Native Speed . In Proceedings of the IEEE International Symposium on Workload Characterization (IISWC), Atlanta, GA, USA, 4–6 October 2015; pp. 183–192. [CrossRef]
9. Butko, A.; Garibotti, R.; Ost, L.; Sassatelli, G. Accuracy Evaluation of GEM5 Simulator System . In Proceedings of the 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), York, UK, 9–11 July 2012; pp. 1–7. [CrossRef]
10. Yu, J.; Yan, M.; Khyzha, A.; Morrison, A.; Torrellas, J.; Fletcher, C.W. Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data. In Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO), Columbus, OH, USA, 12–16 October 2019; pp. 954–968. [CrossRef]
11. Weisse, O.; Neal, I.; Loughlin, K.; Wenisch, T.F.; Kasikci, B. NDA: Preventing Speculative Execution Attacks at Their Source. In Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO), Columbus, OH, USA, 12–16 October 2019; pp. 572–586. [CrossRef]
12. Yan, M.; Choi, J.; Skarlatos, D.; Morrison, A.; Fletcher, C.W.; Torrellas, J. InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy. In Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka, Japan, 20–24 October 2018; pp. 428–441. [CrossRef]
13. Saileshwar, G.; Qureshi, M.K. CleanupSpec: An "Undo" Approach to Safe Speculation. In Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO), Columbus, OH, USA, 12–16 October 2019; pp. 73–86. [CrossRef]
14. Osvik, D.A.; Shamir, A.; Tromer, E. Cache Attacks and Countermeasures: The Case of AES. In *Topics in Cryptology—CT-RSA 2006, Proceedings of the Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, 13–17 February 2005*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 1–20. [CrossRef]
15. Mishra, J.; Sahay, S.K.; Mishra, S.; Pathak, A. ARCHER + AMES Reproducibility Artifact (v1.0.1). Software, Zenodo, 2026. [CrossRef].
16. Bulck, J.V.; Minkin, M.; Weisse, O.; Genkin, D.; Kasikci, B.; Piessens, F.; Silberstein, M.; Wenisch, T.F.; Yarom, Y.; Strackx, R. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In Proceedings of the USENIX Security Symposium, Baltimore, MD, USA, 15–17 August 2018; pp. 991–1008. Available online: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck (accessed on 22 July 2026).
17. van Schaik, S.; Milburn, A.; Österlund, S.; Frigo, P.; Maisuradze, G.; Razavi, K.; Bos, H.; Giuffrida, C. RIDL: Rogue In-Flight Data Load. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, 20–22 May 2019; pp. 88–105. [CrossRef]
18. Vicarte, J.R.S.; Flanders, M.; Paccagnella, R.; Garrett-Grossman, G.; Morrison, A.; Fletcher, C.W.; Kohlbrenner, D. Augury: Using Data Memory-Dependent Prefetchers to Leak Data at Rest. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, 23–26 May 2022; pp. 1491–1505. [CrossRef]
19. Aldaya, A.C.; Brumley, B.B.; ul Hassan, S.; García, C.P.; Tuveri, N. Port Contention for Fun and Profit. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, 20–22 May 2019; pp. 870–887. [CrossRef]
20. Magnusson, P.S.; Christensson, M.; Eskilson, J.; Forsgren, D.; Hållberg, G.; Högberg, J.; Larsson, F.; Moestedt, A.; Werner, B. Simics: A Full System Simulation Platform. *Computer* **2002**, *35*, 50–58. [CrossRef]
21. Yarom, Y. *Mastik: A Micro-Architectural Side-Channel Toolkit* ; The University of Adelaide: Adelaide, Australia, 2016. Available online: https://cs.adelaide.edu.au/~yval/Mastik/Mastik.pdf (accessed on 22 July 2026).
22. Trippel, C.; Lustig, D.; Martonosi, M. CheckMate: Automated Synthesis of Hardware Exploits and Security Litmus Tests. In Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka, Japan, 20–24 October 2018; pp. 947–960. [CrossRef]
23. Intel Corporation. Intel Architecture Memory Encryption Technologies Specification. Intel Document 336907-004US, Rev. 1.4. 2022. Available online: https://www.intel.com/content/www/us/en/content-details/679154/intel-architecture-memory-encryption-technologies.html (accessed on 22 July 2026).
24. Kaplan, D.; Powell, J.; Woller, T. *AMD Memory Encryption*; White Paper; Advanced Micro Devices, Inc.: Santa Clara, CA, USA, 2021. Available online: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf (accessed on 22 July 2026).
25. Gueron, S. Memory Encryption for General-Purpose Processors. *IEEE Secur. Priv.* **2016**, *14*, 54–62. [CrossRef]
26. Waterman, A.; Asanović, K.; Hauser, J. The RISC-V Instruction Set Manual. In V*olume II: Privileged Architecture*; Document Version 20211203; RISC-V International: Zürich, Switzerland, 2021.

27. Waterman, A.; Asanović, K. The RISC-V Instruction Set Manual. In *Volume I: Unprivileged ISA*; Document Version 20191213; RISC-V Foundation: Zürich, Switzerland, 2019.
28. Yarom, Y.; Falkner, K. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, 20–22 August 2014; pp. 719–732.
29. Bernstein, D.J. *Cache-Timing Attacks on AES*; Technical report; Department of Mathematics, Statistics, and Computer Science, University of Illinois Chicago: Chicago, IL, USA, 2005. Available online: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf (accessed on 22 July 2026).
30. Rogaway, P. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *Advances in Cryptology—ASIACRYPT 2004, Proceedings of the 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Republic of Korea, 5–9 December 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 16–31. [CrossRef]
31. National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*; Federal Information Processing Standards Publication 197; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2001. [CrossRef]
32. Dworkin, M. *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*; Technical Report NIST Special Publication 800-38A; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2001. [CrossRef]
33. Dworkin, M. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*; Technical Report NIST Special Publication 800-38D; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2007. [CrossRef]
34. Dobraunig, C.; Eichlseder, M.; Mendel, F.; Schläffer, M. *Ascon v1.2 – Submission to the NIST Lightweight Cryptography Standardisation Process*; NIST Lightweight Cryptography final specification; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2021.
35. Borghoff, J.; Canteaut, A.; Güneysu, T.; Kavun, E.B.; Knezevic, M.; Knudsen, L.R.; Leander, G.; Nikov, V.; Paar, C.; Rechberger, C.; et al. PRINCE—A Low-Latency Block Cipher for Pervasive Computing Applications. In *Advances in Cryptology—ASIACRYPT 2012, Proceedings of the 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, 2–6 December 2012*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 208–225. [CrossRef]
36. Avanzi, R. The QARMA Block Cipher Family. *IACR Trans. Symmetric Cryptol.* **2017**, *2017*, 4–44. [CrossRef]
37. Brier, E.; Clavier, C.; Olivier, F. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems—CHES 2004, Proceedings of the 6th International Workshop Cambridge, MA, USA, 11–13 August 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 16–29. [CrossRef]
38. Chari, S.; Jutla, C.S.; Rao, J.R.; Rohatgi, P. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology—CRYPTO 1999, Proceedings of the 19th Annual International Cryptology Conference, Santa Barbara, CA, USA, 15–19 August 1999*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 398–412. [CrossRef] [PubMed]
39. Ishai, Y.; Sahai, A.; Wagner, D.A. Private Circuits: Securing Hardware against Probing Attacks. In *Advances in Cryptology—CRYPTO 2003, Proceedings of the 23rd Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2003*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 463–481. [CrossRef]
40. Prouff, E.; Rivain, M. Masking against Side-Channel Attacks: A Formal Security Proof. In *Advances in Cryptology—EUROCRYPT 2013, Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, 26–30 May 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 142–159. [CrossRef]
41. Wang, Y.; Paccagnella, R.; He, E.T.; Shacham, H.; Fletcher, C.W.; Kohlbrenner, D. Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86. In Proceedings of the USENIX Security Symposium, Boston, MA, USA, 10–12 August 2022; pp. 679–697. Available online: https://www.usenix.org/conference/usenixsecurity22/presentation/wang-yingchen (accessed on 22 July 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.