FERMIONS ASI CORP.
AI-powered intelligence for digital chip design
www.fermions.co

W H I T E P A P E R

SAiGE: Closed-Loop Agentic Bug-Fixing on a Production RISC-V Vector Processor

A Differential-Verification Campaign on PULP Ara

June 2026

Table of Contents

SAiGE: Closed-Loop Agentic Bug-Fixing on a Production RISC-V Vector Processor

Abstract

SAiGE (Silicon Architecture Intelligence Generation Engine) is Fermions' agentic AI platform for semiconductor engineering, designed to support workflows spanning RTL development, verification, debugging, and iterative design refinement. In this paper, we report a closed-loop, golden-model-verified bug-fixing campaign carried out by SAiGE on **PULP Ara**, a 64-bit RISC-V Vector (RVV 1.0) coprocessor for the CVA6 core. Unlike benchmark suites of small, self-contained designs, Ara is a large, in-production RTL codebase (multi-lane vector datapath, VLSU, mask unit, FP/INT multiply-FPU, dispatcher, and main sequencer), making it a stronger test of whether an agentic RTL system can operate in a production hardware environment.

Working from open GitHub issues and independent triage, SAiGE localized, root-caused, and fixed **15 distinct functional correctness bugs** across 7 RTL modules, producing for each an atomic, upstream-cherry-pickable commit and a **Spike-vs-Ara differential reproducer**. Every fix was validated against the Spike RISC-V golden ISA model before being committed; for the hardest bug (an out-of-order-retire read-after-read hazard), SAiGE rebuilt the *unfixed* model to demonstrate the reproducer triggers the failure, then rebuilt the fixed model to demonstrate the match — a manual pass/fail verdict loop. A final regression of all **18 reproducers against the combined model passed 18/18** at **2, 4, 8, and 16 lanes**, with three compute benchmarks (dotproduct, fmatmul, imatmul) confirming no regression to normal operation and the #434 fix measured cycle-for-cycle performance-neutral. The multi-configuration sweep itself caught and fixed a lane-count-dependent deadlock in an early version of one fix. This work positions **SAiGE as a practical agentic RTL debugging system** for real hardware codebases and as a real-world instance of the CVDP "code-debug" (cid016) and "code-modify" (cid004) task types, executed on a live codebase rather than a curated benchmark.

1. Introduction

SAiGE is designed to move agentic semiconductor engineering beyond isolated RTL generation and into closed-loop development workflows. Agentic RTL development has been evaluated almost exclusively on benchmark suites — RTLLM 2.0 (50 single-module designs with provided self-checking testbenches) and NVIDIA CVDP (92 multi-task problems with an agent-blind cocotb harness). These benchmarks measure an agent's ability to *generate* or *repair* small, isolated modules against a known oracle.

Production hardware bug-fixing is a different regime:

- **No provided testbench for the bug:** The failure is described informally in a GitHub issue (or not at all). SAiGE must construct a reproducer that exposes the bug and a golden reference that defines correct behavior.
- **Large, unfamiliar codebase:** A fix in Ara touches a single file but must be reasoned about against a multi-stage, multi-lane micro-architecture (issue → operand fetch → lanes/VLSU/mask-unit → writeback → retire) with subtle cross-unit handshakes.
- **Regression risk dominates:** A correctness fix that breaks an unrelated path is worse than no fix. Each change must be shown not to regress the rest of the design.
- **Upstream hygiene:** To be useful to the community, fixes must be atomic, individually justified, and accompanied by a test that fails before and passes after.

This SAiGE campaign applies the disciplines articulated in our RTL-dev-agent work — *verify against a golden model, debug from evidence not guesswork, require evidence of correctness before completing* — to this harder, real-world regime.

From a product perspective, the campaign exercises SAiGE's core closed-loop workflow: understand a design context, generate or adapt a reproducer, compare against a trusted oracle, localize the RTL failure, produce a minimal patch, and validate that the fix composes with the rest of the design. The goal is not simply to generate RTL, but to support the engineering loop required for production-quality semiconductor development.

1.1 Contributions

Using SAiGE, we demonstrate:

1. **15 root-caused RTL correctness fixes** in PULP Ara across the dispatcher, main sequencer, lane sequencer, mask unit, vector ALU, FP-MUL/FPU, and address generator, each mapped to a tracked upstream issue.
2. **A differential reproducer per fix** (18 reproducer apps total), each a minimal RISC-V Vector program checked against the Spike golden model, designed to be CI-safe (self-checking, terminating).
3. **A reproducible differential harness** (`spike-vs-ara.sh`, `run-regression.sh`) that builds each app for both the Spike golden model and the Verilated Ara RTL, runs both, and compares value tokens — robust to Spike's HTIF first-character line mangling.
4. **A full regression of the combined fix bundle** (18/18 reproducers + 3 compute benchmarks) demonstrating the fixes compose without interaction regressions.

2. Target and Tooling

Target: PULP Ara (`pulp-platform/ara`), branch `dev/rvv-correctness-bundle`.

Configuration: 4 lanes, VLEN=4096, RVV 1.0 with Zvfh, attached to CVA6 (sv39 MMU path present).

Toolchain:

- **Golden model:** Spike (`riscv-isa-sim`) with `--isa=rv64gcv_zfh_zvfh_zvl4096b`.
- **DUT:** Verilator 5.x model of the full Ara testharness (`make verilate / make simv`).
- **Compiler:** RISC-V GCC 13.2 + picolibc; apps built for both Ara (bare-metal `crt0.S`) and Spike (riscv-tests `crt.S`).
- **Differential harness:** builds an app for both targets, runs both, strips HTIF noise, and compares.

A recurring practical hazard: Spike's HTIF console mangles the *first character of every output line*, so `cause=4` prints as `uause=4`. All reproducer checks therefore match on **value tokens after `=`** (mid-line, never mangled) rather than on labels. The trap-based probes carry **two trap handlers in one binary** — a naked `mtvec_handler` for Ara's bare-metal runtime and a C `handle_trap()` for Spike's runtime — so the same source runs identically on both.

3. SAiGE Methodology: The Closed-Loop Differential Cycle

For each candidate bug, SAiGE ran the following loop:

1. **Triage / localize.** Read the issue (or hypothesize from spec knowledge), then search the RTL to localize the responsible unit and signal. Where the codebase was unfamiliar, a context-gathering pass mapped the relevant datapath before any edit.
2. **Reproduce.** Write a minimal RVV program that exercises the suspected path and prints observable state (loaded values, trap cause/tval, reduction result). Run it on **Spike** to capture the golden output.
3. **Confirm the bug.** Run the *unmodified* Ara model; confirm divergence from Spike. (For the sequencer hazard, this meant building the unfixed model explicitly to prove the reproducer is sensitive to the bug.)
4. **Root-cause and fix.** Reason about *why* the RTL diverges — not patch the symptom. Apply the minimal change.
5. **Verify.** Rebuild the Verilated model; confirm the reproducer now matches Spike, and that closely-related behavior is preserved (e.g. an *active* misaligned element must still trap even after we stop trapping *masked-off* ones).
6. **Guard against regression.** Run adjacent reproducers and compute benchmarks.
7. **Commit atomically.** One fix per commit, `:bug:` tag, `Refs #NNN`, CHANGELOG entry, and the reproducer (only `main.c` tracked).

This mirrors the CVDP **specification-only + verdict-feedback** protocol: SAiGE verifies against its own golden-model reading (specification-only), and the unfixed-vs-fixed differential provides a sanitized pass/fail verdict (no exposure to a hidden harness — the harness *is* Spike, an independent reference).

4. The Fix Catalog

All fixes are on `dev/rvv-correctness-bundle`, atomic and individually cherry-pickable.

S.No.	Commit	Module	Issue	Summary
1	`649658f6`	dispatcher	#436	Clear `use_vs1` for VWXUNARY0 (`vmv.x.s`) so a stale vs1 read does not create a false hazard
2	`b2f9a893`	valu	#451	Start reductions only when the result queue is empty, fixing a reduction deadlock
3	`ef5c5a8d`	masku	#448	Drain ALU operands during `vid.v`, fixing a mask-unit drain deadlock (also closes #437)
4	`993783d2`	vmfpu	#447	Use non-comparison latency for FP compares, fixing mask-routing desync
5	`d2b4974c`	masku	#446	Trim `vcpop/vfirst` operand to the active `vl`
6	`c9fb00a7`	lane_sequencer	#455	Do not rescale the indexed-memory index fetch when data SEW ≠ index EEW (hang)
7	`cd5eceb7`	dispatcher	#453	Tag the full `vd..vd+nf` register range for segment loads/stores
8	`392c2760`	masku	#450	Tag the last selected `vcompress` element as terminator (completion hang)
9	`b875ed3e`	dispatcher	#452	Force `vsext/vzext.vf8` source width to EW8
10	`d05327b6`	lane_sequencer	#459	Fix mask-row skip for large-stride `vslideup`
11	`04a5cd52`	lane_sequencer	#462	Fix mask-row index for masked memory ops with `vstart`
12	`3e956446`	addrgen	#457	Report misaligned indexed access as `LD/ST_ADDR_MISALIGNED` with faulting tval (not `ILLEGAL_INSTR`/0)
13	`821190ed`	dispatcher	#460	Reject masked `vmerge/vfmerge` with `vd=v0` as illegal-instruction
14	`d0599762`	addrgen	#456	Skip the misalignment exception for masked-off indexed elements
15	`1e3e5524`	ara_sequencer	#434	Track all readers per register for correct WAR under out-of-order retire

In addition, issues **#458** and **#461** were investigated and confirmed to already behave correctly (no fix; a passing probe was kept locally for #458), and **#437** is closed by fix #3.

Failure-mode distribution. Of the 15 fixes: 4 were **deadlocks/hangs** (#451, #448, #450, #455), 3 were **exception-semantics** bugs (#457, #460, #456), 5 were **mask/operand-routing** bugs (#447, #446, #459, #462, #436), 2 were **decode/width** bugs (#452, #453), and 1 was a **structural hazard** under out-of-order retire (#434). This distribution is notable: over a quarter of the bugs are *liveness* failures (deadlock/hang) that a single-shot generation benchmark would never surface, because they only manifest in the full pipeline under specific operand-timing.

5. Case Studies

5.1 #434: Read-after-read correctness under out-of-order retire (`'ara_sequencer.sv'`)

The bug: Ara issues in program order but retires out of order. The main sequencer's read table recorded only the *last* instruction reading each vector register, so a writer built a Write-After-Read hazard against only that last reader. A slow earlier reader could be overtaken:

```
vdivu.vv v6, v4, v2   ; slow, reads v2
vadd.vv  v8, v2, v2   ; fast, reads v2 -> overwrites the read-table entry
vadd.vv  v2, v12, v12 ; writes v2 -> WAR built only vs the fast vadd (bug)
```

Once the fast `vadd` retired, the writer to v2 was released while the slow `vdivu` had not finished reading `v2`, so `vdivu` consumed the *new* `v2` for its tail elements.

The fix: Replace the per-register "last reader id" with a per-register **bitmask of every in-flight reader** (one bit per instruction id). The WAR build then ORs the whole reader mask into the writer's hazards, so the writer waits for *all* pending readers. Reads decay with `vinsn_running`; writes still track only the last writer (serialized by WAW). This is the maintainer's preferred approach (it touches only writers — no reader-reader serialization, hence no slowdown on read-heavy code such as matmul).

Verification (the most rigorous in the campaign): A 256-element e32/m2 divide-then-clobber reproducer:

- Spike golden: `sum=2048 bad=0`
- **Unfixed** Ara: `sum=1040 bad=168` (168/256 elements wrong — the bug, demonstrated by building the unfixed model)
- **Fixed** Ara: `sum=2048 bad=0` (matches Spike)

5.2 #456: Masked-off misaligned indexed element must not trap (`'addrgen.sv'`)

The bug: For a masked indexed load/store, an element whose mask bit is 0 must not raise a misalignment exception (RVV: masked-off elements generate neither exceptions nor architectural accesses). The address generator checked alignment per element with no visibility into the per-element mask (a literal `TODO` in the source) and trapped.

The fix (scoped for zero regression risk): Route the mask into the addrgen as a *peek-only* input (the load/store units still own the handshake). For a misaligned indexed element whose mask bit is 0, suppress the exception, align the address down to the element width, and emit a normal beat — the VLDU/VSTU byte strobes already zero masked-off elements, so the element is dropped exactly like any other masked-off element and element accounting stays in sync. The mask is resolved only when the whole vector fits in one mask chunk (`vl ≤ elements-per-chunk`); otherwise the conservative trapping behavior is kept, so multi-chunk vectors cannot regress.

Verification: Reproducer at **e16/e32/e64**: masked-off misaligned element no longer traps (`cause=0`) and keeps its prior value; an *active* misaligned element still traps with `LD_ADDR_MISALIGNED (cause=4)` — confirmed identical on Spike. This guards against over-suppression.

6. Evaluation

6.1 Combined-bundle regression

The decisive test of a multi-fix campaign is whether the fixes ***compose***. We rebuilt the Verilated model with all 15 fixes applied and ran the entire reproducer suite against that single model.

Suite	Result
Self-checking reproducers (5)	5/5 PASS
Differential reproducers vs. Spike (13)	13/13 PASS
Total reproducer regression	18/18 PASS

Self-checking apps: `'vcompress_sweep', 'vid_drain_deadlock', 'vmfeq_route', 'vmv_x_s_hazard', 'vredsum_deadlock'.`

Differential apps: `'rar_retire_hazard', 'vcpop_vl', 'vid_m4_hang', 'vle_vstart_mask', 'vlseg_eew', 'vlseg_mask', 'vlxe_masked_misalign', 'vlxe_misalign_cause', 'vlxe_sew_eew', 'vmerge_v0_illegal', 'vse_vstart_mask2', 'vslideup_mask', 'vzext_vf8'.`

6.2 Normal-operation regression

To confirm the fixes (especially the central sequencer change) do not regress ordinary execution, three compute kernels were run end-to-end on the fixed model:

Kernel	Path exercised	Result
dotproduct	reductions, mixed SEW	SUCCESS
fmatmul (8×8×8)	FP MAC chains, heavy register reuse	PASSED
imatmul (8×8×8 / 16×16×16)	integer MAC chains	PASSED

6.3 Multi-configuration regression (lane-count independence)

Because VLMAX depends only on VLEN/LMUL/SEW (not lane count), the lane-count variable can be isolated by holding **VLEN=4096** (the maximum the Spike golden model supports) and rebuilding the DUT at 2, 4, 8, and 16 lanes; the same app binaries (RVV is lane-agnostic) and the same Spike@VLEN=4096 oracle apply. The full reproducer suite was run at each:

Configuration	Reproducer regression
2 lanes, VLEN=4096	18/18 PASS
4 lanes, VLEN=4096 (default)	18/18 PASS
8 lanes, VLEN=4096	18/18 PASS
16 lanes, VLEN=4096	18/18 PASS

The fixes are also VLEN-independent by construction (none index by VLEN — the mask bitmask tracks `NrVInsn`, the addrgen chunk/lane math tracks `NrLanes`), and VLEN=4096 is both the reference and the largest VLEN the oracle supports. This sweep **caught a real lane-count-dependent defect** in the first version of the #456 fix: its mask-wait condition required *all* lanes' mask beats to be valid (`&mask_valid_i`), which deadlocks whenever `vl` does not span every lane (e.g. `vl`=4 on 8 lanes — only 4 lanes carry elements, so the others never produce a mask beat). The fix was corrected to wait only on the **specific lane** holding the element's mask bit (`mask_valid_i[lane]`); the 8-lane reproducer then passed, and 2/4-lane were re-confirmed. This is the value of multi-configuration differential testing: a deadlock invisible at the default 4-lane config surfaced immediately at 8 lanes.

6.4 Performance neutrality of the #434 fix

The #434 change adds Write-After-Read edges only for writers, so it should not slow read-heavy code. Measured directly on imatmul (16×16×16) at 4 lanes, with and without the fix:

Build	imatmul stage cycles
With #434 (fixed)	496 / 1025 / 2621
Without #434 (reverted)	496 / 1025 / 2621

Cycle counts are **identical** — the additional reader tracking is performance-neutral on this matmul workload.

6.5 Why the differential method is sound here

Spike is an independent, widely-trusted RISC-V ISA reference; it executes sequentially with no micro-architectural hazards, so it defines the architecturally-correct result for every reproducer. A bug that is a *micro-architectural* artifact of Ara (out-of-order retire, operand streaming, mask-chunk timing) is invisible to Spike by construction, which is exactly what makes Spike a valid oracle: any Ara-vs-Spike divergence is an Ara bug. The =-token comparison neutralizes the only systematic nuisance (HTIF first-char mangling).

7. SAiGE Beyond Benchmark-Based Agentic RTL

This campaign demonstrates SAiGE's application to the **CVDP code-debug (cid016) and code-modify (cid004) task types on a real codebase**, with two differences that make it harder than the benchmark:

- **No curated golden Python model per problem.** The oracle is a full ISA simulator, and the "specification" is the RVV 1.0 standard plus the issue text. The system must itself decide what correct behavior is and encode it as a reproducer.
- **Cross-unit micro-architectural reasoning.** CVDP problems are typically single modules; several Ara bugs (e.g. #434, #456, #462) are *interaction* bugs spanning sequencer ↔ lanes ↔ VLSU ↔ mask-unit, where the fix is in one file but the reasoning spans the pipeline.

The verdict-feedback idea from the CVDP evaluation maps cleanly onto our unfixed-vs-fixed differential: a single sanitized bit ("does it match Spike") is what converts a plausible-but-wrong interpretation into a correct fix — most visibly in #434, where the unfixed build's bad=168 verdict confirmed both the bug and, after the fix, its resolution.

8. Validation Scope and Remaining Limitations

Now covered (previously listed as limitations). Verification spans **2, 4, 8, and 16 lanes** (VLEN=4096) with the full reproducer suite passing 18/18 at each (§6.3), and the #434 fix is measured **cycle-for-cycle performance-neutral** on imatmul (§6.4). The single-configuration and performance caveats are resolved.

Remaining limitations (honestly open):

- **No formal verification.** All verification is dynamic (Spike differential), not formal. A formal WAR-hazard proof for #434 or a bounded model check of the addrgen exception logic would raise assurance beyond simulation.
- **Not exhaustive instruction-class coverage.** Each reproducer is a targeted regression witness for its specific scenario, not a proof of full instruction-class correctness. The 2/4/8/16-lane sweep broadens coverage, but exhaustive coverage would require the upstream vector ISA test suite in CI.

9. Future Work

- Open the upstream pull requests (the commits are structured for this) and engage maintainer review.
- Wire the reproducer suite + lane sweep into CI; add the upstream vector ISA test suite for exhaustive coverage.
- Introduce formal property checks for the hazard and exception logic.
- Extend the campaign to the remaining open issues, prioritizing liveness bugs (the highest-value, benchmark-invisible class).

10. Conclusion

SAiGE, operating as a closed-loop, golden-model-verified RTL agent, fixed 15 distinct correctness bugs in a production RISC-V vector processor, each with a differential reproducer, and demonstrated that the combined bundle passes an 18/18 regression with no impact on normal compute kernels. over a quarter of the bugs were liveness failures invisible to single-shot generation benchmarks, underscoring that the highest-value target for agentic RTL is not greenfield generation but **differential debugging of real, concurrent hardware against a trusted reference**.

References:

[1] PULP Platform, *"Ara: 64-bit RISC-V Vector Unit,"* GitHub repository. Available: https://github.com/pulp-platform/ara

[2] M. Perotti, M. Cavalcante, N. Wistoff, R. Andri, L. Cavigelli, and L. Benini, "*A 'New Ara' for Vector Computing: An Open Source Highly Efficient RISC-V V 1.0 Vector Processor Design*," arXiv:2210.08882, 2022. Available: https://arxiv.org/abs/2210.08882

[3] M. Perotti, M. Cavalcante, R. Andri, L. Cavigelli, and L. Benini, "*Ara2: Exploring Single- and Multi-Core Vector Processing with an Efficient RVV 1.0 Compliant Open-Source Processor*," arXiv:2311.07493, 2023. Available: https://arxiv.org/abs/2311.07493

[4] OpenHW Group, "*CVA6 RISC-V CPU*," GitHub repository. Available: https://github.com/openhwgroup/cva6

[5] OpenHW Group, "*CVA6 User Manual.*" Available: https://docs.openhwgroup.org/projects/cva6-user-manual/

[6] RISC-V International, "*The RISC-V Vector Extension, Version 1.0.*" Available: https://docs.riscv.org/reference/isa/extensions/vector/_attachments/riscv-v-spec.pdf

[7] RISC-V Software, "*Spike, the RISC-V ISA Simulator,*" GitHub repository. Available: https://github.com/riscv-software-src/riscv-isa-sim

[8] Verilator Project, "*Verilator SystemVerilog Simulator,*" GitHub repository. Available: https://github.com/verilator/verilator

[9] Picolibc Project, "*Picolibc: C Library for Embedded Systems*," GitHub repository. Available: https://github.com/picolibc/picolibc

[10] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "*RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Models,*" arXiv:2308.05345, 2023. Available: https://arxiv.org/abs/2308.05345

[11] HKUST Zhiyao Lab, "*RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Models*," GitHub repository. Available: https://github.com/hkust-zhiyao/RTLLM

[12] S. Liu, Y. Lu, W. Fang, M. Li, and Z. Xie, "*OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation*," arXiv:2503.15112, 2025. Available: https://arxiv.org/abs/2503.15112

[13] N. Pinckney, C. Deng, C.-T. Ho, Y.-D. Tsai, M. Liu, W. Zhou, B. Khailany, and H. Ren, "*Comprehensive Verilog Design Problems: A Next-Generation Benchmark Dataset for Evaluating Large Language Models and Agents on RTL Design and Verification*," arXiv:2506.14074, 2025. Available: https://arxiv.org/abs/2506.14074

[14] NVIDIA Research / NVLabs, "*CVDP Benchmark*," GitHub repository. Available: https://github.com/NVlabs/cvdp_benchmark

Notice

This document is provided for informational purposes only and shall not be regarded as a warranty of any functionality, condition, or quality of any product or system. Fermions ASI Corp. (“Fermions”) makes no representations or warranties, expressed or implied, as to the accuracy, completeness, or reliability of the information contained in this document and assumes no responsibility for any errors or omissions.

This document does not constitute a commitment to develop, release, or deliver any product, feature, code, or functionality described herein. Fermions reserves the right to make changes, corrections, enhancements, or modifications to the content of this document at any time without notice.

The information presented is intended for general guidance only. Users are responsible for independently evaluating the applicability of this information for their specific use cases and for performing all necessary validation, testing, and verification.

Fermions shall not be liable for any direct, indirect, incidental, special, or consequential damages arising from the use of, or reliance on, this document or the information contained herein.

No license, express or implied, is granted under any intellectual property rights of Fermions. References to third-party products, services, or technologies do not constitute endorsement or recommendation and may require separate permissions or licenses from their respective owners.

Reproduction, distribution, or use of this document, in whole or in part, is permitted only with prior written consent from Fermions and must be reproduced without modification and with all notices included.

Important Disclaimer

THIS DOCUMENT IS PROVIDED “AS IS.” FERMIONS MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

Copyright

Fermions — AI-powered intelligence for digital chip design

contact@fermions.co | *www.fermions.co*