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

W H I T E P A P E R

SAiGE : Agentic RTL Development on the CVDP Benchmark

Paper 2

A General-Purpose Agent Across Multi-Task Hardware Design

June 2026

Table of Contents

SAiGE : Agentic RTL Development on the CVDP Benchmark, A General-Purpose Agent Across Multi-Task Hardware Design

Abstract **3**

1. Introduction **4**

2. The CVDP Benchmark **5**

2.1 Overview 5

2.2 Task Types 5

2.3 Difficulty 5

3. SAiGE Approach **6**

4. Results **8**

4.1 Primary Results — Specification-Only 8

4.2 Pass@k with Sanitized Verdict Feedback 8

4.3 Comparison with Published Single-Turn Results 9

4.4 By Category 10

4.5 By Difficulty 10

4.6 Comparison with RTLLM 2.0 10

5. Comparison with CVDP-Specialized Systems **11**

5.1 ACE-RTL 11

5.2 Agentrys 11

5.3 What the Comparison Means 12

5.4 Comparability Caveats 13

6. Analysis **15**

6.1 Where Iterative Debug Succeeds 15

6.2 What the Sanitized Feedback Loop Adds 15

6.3 Where It Still Fails 15

7. Limitations **17**

8. Conclusion **18**

References **19**

Notice **21**

Abstract

This paper evaluates SAiGE, *a general-purpose RTL development agent*, on the NVIDIA CVDP agentic benchmark — 92 code-generation problems spanning greenfield RTL generation, code modification, testbench-driven implementation, and bug-fixing. In the specification-only setting, with no verdict feedback from the grading harness — the protocol under which prior single-turn results were reported — SAiGE reaches **64% functional pass rate with Claude Opus 4.6 and 62% with Claude Sonnet 4.6,** more than doubling the best previously reported single-turn result of 29%. Adding a **sanitized verdict-feedback loop** — a pass@k signal that returns only pass/fail counts, never test source or golden values — **raises the rate to 83% (76/92)**. We position these results against two recent CVDP-specialized systems, ACE-RTL (88.9%) and Agentrys (95.8%), which score higher but are purpose-built and tightly closed-loop on CVDP, in contrast to SAiGE as a task- and benchmark-agnostic agent. This paper analyzes where iterative debug succeeds, where it plateaus, and what separates a general-purpose agent from a CVDP-specialized one.

1. Introduction

In our prior paper, **SAiGE: Autonomous RTL Development — Agentic Verilog Generation with Closed-Loop Debugging**, we demonstrated a 100% functional pass rate on RTLLM 2.0 — a benchmark of 50 single-module Verilog designs with provided self-checking testbenches. While this established the effectiveness of disciplined agentic workflows for RTL development, RTLLM represents a narrow evaluation surface: all problems are greenfield, all have testbenches, and most involve straightforward sequential or combinational logic.

Real hardware development requires handling diverse task types, reading multi-file specifications, working with existing codebases, and verifying correctness without access to a golden reference. The NVIDIA CVDP benchmark provides this harder evaluation, testing whether the same agent architecture generalizes beyond the single-module, testbench-provided setting.

A central theme of this paper is the distinction between a **general-purpose agent** — one whose workflow, prompting, and tools are not tuned to any single benchmark — and **benchmark-specialized systems** that are engineered, trained, or architecturally evolved specifically against CVDP. SAiGE is evaluated here as the former: the same workflows and tools used for RTLLM, and day-to-day RTL work are applied to CVDP unchanged. We show this general-purpose agent is competitive, and we are explicit about the gap to specialized systems and why it exists.

2. The CVDP Benchmark

2.1 Overview

The CVDP agentic benchmark contains code-generation problems evaluated using open-source simulators. Each problem provides a natural-language prompt and specification documents describing the expected behavior. Evaluation uses a cocotb-based harness with a Python golden model that the agent develops against in isolation. We evaluate the 92-problem code-generation subset: categories cid003, cid004, cid005, and cid016.

2.2 Task Types

Category	Count	Description
RTL from spec (cid003)	34	Implement a complete module from prose specification
RTL modify (cid004)	25	Extend or change an existing module
TB-given (cid005)	22	Implement RTL that satisfies a provided testbench
Bug-fix (cid016)	11	Identify and repair bugs in provided RTL

2.3 Difficulty

Problems range from simple encoders and FSMs through cache controllers and protocol bridges to DES/AES cryptography, DSP equalizers, and multi-clock designs. The difficulty distribution is approximately 26% easy, 49% medium, and 25% hard.

3. SAiGE Approach

We apply the same general-purpose SAiGE agent used in our prior RTLLM work, described in **SAiGE: Autonomous RTL Development — Agentic Verilog Generation with Closed-Loop Debugging**, unchanged in its core design, to the multi-task CVDP setting. The agent pairs a frontier LLM with a hardware-design toolchain, including compilation, simulation, and waveform inspection, and a set of workflow disciplines that guide it through specification analysis, implementation, and verification. None of these disciplines are CVDP-specific — they encode general RTL-engineering practice. At a high level, SAiGE:

Adapts to the task type. It recognizes whether a problem is greenfield generation, modification of existing RTL, testbench-driven implementation, or bug-fixing, and adjusts its workflow accordingly — reading existing code before changing it, honoring a provided testbench, or producing its own verification when none exists.

Verifies against the specification, not just the prompt. When no testbench is provided, the agent derives expected behavior from the specification documents and checks its design against that understanding across a range of input and parameter configurations, rather than trusting a stimulus-only bench that can pass vacuously.

Resolves specification ambiguity explicitly. Before implementing, the agent surfaces under-specified aspects of a problem, such as reset semantics, overflow handling, or clock-domain behavior, commits to an interpretation with stated rationale, and proceeds from there. This directly targets the dominant failure mode in the specification-only setting: silently diverging from the grader's intended interpretation.

Debugs from evidence rather than guesswork. When simulations fail repeatedly, the agent is steered toward cycle-accurate analysis of actual signal behavior — locating where observed values diverge from expected and tracing the fault upstream — instead of speculative edits.

Requires evidence of correctness before completing. The agent cannot declare a task done on the basis of static checks alone; it must have exercised the design in simulation first.

3.1 Two Evaluation Settings

In both settings, the agent is blind to the harness internals — it never sees test source, assertion text, golden values, test names, or stack traces. The settings differ only in whether a pass/fail verdict is returned when the agent submits.

- **Specification-only.** The agent receives no feedback from the grading harness. It verifies solely against its own reading of the specification. This matches the protocol under which prior single-turn CVDP results were reported, so it is the directly comparable setting for those numbers.

- **Verdict-feedback, pass@k.** When the agent submits, the official harness is run and only a sanitized verdict is returned — the number of tests, how many passed and failed, and which parameter configurations failed. The agent gets a small bounded number of such attempts. This supplies a whether-correct signal while keeping the agent blind to how correctness is judged — similar to how an engineer learns that a regression failed on configuration X from a CI gate without reading the hidden test.

Neither setting exposes the grader's internals. The verdict-feedback setting simply adds the kind of pass/fail signal a continuous-integration gate would provide in practice.

4. Results

4.1 Primary Results — Specification-Only

Model Configuration	Functional Pass Rate
SAiGE + Claude Opus 4.6	59/92 (64%)
SAiGE + Claude Sonnet 4.6	57/92 (62%)

4.2 Pass@k with Sanitized Verdict Feedback

Adding the sanitized verdict-feedback loop — a small, bounded number of attempts, verdict only — raises the pass rate substantially. Starting from the specification-only Opus baseline, 17 additional problems flip from FAIL to PASS.

Setting	Pass Rate
Specification-only, Opus 4.6	59/92 (64%)
+ sanitized verdict feedback, pass@k	76/92 (83%)

Every flip was independently re-verified by re-running the official harness on the agent's final RTL. Notably, most flips landed on the second attempt: the agent's own testbench passed, the verdict revealed a residual functional error, and the agent corrected it without seeing the test. This includes problems that had resisted prior attempts, such as the DSP **phase_rotation** and **dynamic_equalizer** families and a secure thermostat FSM.

4.3 Comparison with Published Single-Turn Results

Prior evaluations on the CVDP agentic non-commercial dataset used single-turn, non-agentic inference — no tool use, no iterative debug, and no simulation feedback.

Model	Approach	Overall Pass@1
SAiGE + Claude Opus 4.6, spec-only	**Agentic, tool-using, iterative**	**64%**
SAiGE + Claude Sonnet 4.6, spec-only	**Agentic, tool-using, iterative**	**62%**
Claude 3.7 Sonnet + Thinking	Single turn	29%
GPT-4.1	Single turn	21%
Llama 3.1 405B	Single turn	21%
GPT o4-mini	Single turn	20%
Claude 3.5 Haiku	Single turn	20%
GPT o1	Single turn	14%

The specification-only SAiGE result more than doubles the best previously reported single-turn result, improving from 29% to 64%. The gap between single-turn generation and agentic evaluation is larger than the gap between models within either paradigm — the closed loop of compile, simulate, and debug matters more than model choice.

Note on scope: Published single-turn results span the full agentic dataset, including testbench-generation categories where all models score near zero. This evaluation uses the 92-problem code-generation subset. Comparisons should account for this difference in scope.

4.4 Results by Category, Opus 4.6

Category	Spec-only	Pass@k	Best Prior, single-turn
RTL from spec, cid003	74% (25/34)	88% (30/34)	49%
RTL modify, cid004	60% (15/25)	84% (21/25)	42%
TB-given / Module reuse (cid005)	50% (11/22)	68% (15/22)	24%
Bug-fix, cid016	73% (8/11)	91% (10/11)	53%
Overall	**64% (59/92)**	**83% (76/92)**	**29%**

4.5 Results by Difficulty, Specification-Only, Opus 4.6

Difficulty	SAiGE + Opus 4.6
Easy, 24 problems	~88%
Medium, 45 problems	~62%
Hard, 23 problems	~43%

4.6 Comparison with RTLLM 2.0

	RTLLM 2.0, 50 problems	CVDP spec-only, 92 problems	CVDP pass@k, 92 problems
SAiGE + Sonnet 4.6	100%	62%	-
SAiGE + Opus 4.6	100%	64%	83%

The gap between RTLLM and CVDP reflects the fundamental difference in problem structure. RTLLM provides self-checking testbenches that give the agent direct correctness feedback, while CVDP in the specification-only setting requires the agent to infer correct behavior from prose and verify against its own, potentially incomplete, understanding. The pass@k setting partially restores the RTLLM-style feedback channel, but in a sanitized form that never reveals the test itself.

5. Comparison with CVDP-Specialized Systems

Two recent systems report higher CVDP scores than SAiGE. Both are purpose-built and tightly closed loop on CVDP; neither is positioned as a general-purpose RTL agent. We summarize them and explain the architectural differences.

5.1 ACE-RTL, NVIDIA

ACE-RTL (Deng et al., 2026) reports up to 88.9% overall on CVDP. Its architecture unifies an RTL-specialized model trained on 1.7M RTL samples, the Generator, with a frontier reasoning LLM, Claude-4-Sonnet, acting as a Reflector and Coordinator. Key properties include:

- A domain-specialized generator fine-tuned specifically for RTL, based on Qwen2.5-Coder-32B, 3 epochs, 32×8 A100s.
- An Agentic Context Evolution loop that aggregates debugging history and triggers full restarts when progress stalls.
- A parallel scaling strategy: 5 concurrent processes, each up to 30 iterations, terminating once anyone passes.
- Reported per-category results: Code Completion 80.8%, Spec-to-RTL 96.2%, Code Modification 90.9%, and Code Debug 91.4%.

5.2 Agentrys

Agentrys (Tsai et al., 2026) reports 95.8% overall, including Code Debug at 100%, surpassing ACE-RTL by approximately 6.9 points. It is a self-improving multi-agent system developed across seven generations, Gen 0–6, evolving from a single agent to a 16-agent hierarchical network with competing solver teams, an evidence-weighted aggregator, an adversarial verifier, and a simulation specialist. Reported per-category results are: Code Completion 96.8%, Spec-to-RTL 96.2%, Code Modification 90.9%, and Code Debug 100%.

5.3 What the Comparison Means

System	CVDP Overall	Generality	Key mechanism
Agentrys	**95.8%**	CVDP-specialized	16-agent hierarchical network, 7 self-improving generations
ACE-RTL	**88.9%**	CVDP-specialized	1.7M-sample fine-tuned RTL model + reflector, 5× parallel × 30 iterations
SAiGE, pass@k	**83%**	**General-purpose**	Single general agent + sanitized verdict feedback
SAiGE, spec-only	**64%**	**General-purpose**	Single general agent, no verdict feedback

These systems score higher, and this paper does not dispute that. The relevant distinction is **specialization versus generality.**

- **Specialized training.** ACE-RTL's gains come substantially from a generator fine-tuned on 1.7M RTL samples. SAiGE uses an unmodified frontier model with no RTL-specific training.
- **CVDP-tuned architecture.** Agentrys evolved a 16-agent network across seven generations measured against CVDP. SAiGE is a single general-purpose agent also used on RTLLM and in everyday RTL work, unchanged.
- **Heavy parallel and iteration budgets.** Both specialized systems lean on large search budgets. ACE-RTL uses 5 parallel processes with up to 30 iterations each, while Agentrys uses multiple competing solver teams with debate. SAiGE pass@k uses a single agent with a small, bounded number of sanitized attempts.
- **Metric**. Both report an Agentic Pass Rate, or unique-solved-over-trials, that aggregates multiple runs. SAiGE figures are single-trajectory pass rates in the specification-only setting and a small-bounded-attempt pass@k result.

The takeaway is not that SAiGE outperforms these systems. Rather, **a general-purpose agent, applied to CVDP without benchmark-specific training or architecture, reaches 83% with a minimal sanitized-feedback loop,** within range of systems engineered specifically for this benchmark. For teams that need one agent across many tasks and codebases rather than a CVDP-optimized pipeline, this generality is the point.

5.4 Comparability Caveats

The headline results should not be interpreted as a strict system-to-system ranking, because the reported figures differ in evaluation scope, methodology, and search budget. The following caveats are provided to clarify comparability, not to question the validity of the reported results.

Different problem set. ACE-RTL evaluates on CVDP categories cid002, cid003, cid004, and cid016 with problem counts 94, 78, 55, and 35. These match the **non-agentic** split of CVDP almost exactly, not the agentic split. The non-agentic set is **easy and medium only** — the CVDP authors note that hard problems are too complex for single-turn evaluation and are placed exclusively in the agentic set. ACE-RTL's set also **includes cid002 Code Completion**, the easiest and highest-pass category, and **excludes cid005 Module Reuse**, an agentic-only category and the hardest in this evaluation. SAiGE's 92-problem set is the agentic code-generation split: cid003, cid004, cid005, and cid016. It includes hard problems and module reuse, and excludes the easier code-completion category. ACE-RTL's distribution is therefore materially easier than the evaluated SAiGE split.

Different metric and search budget. Both ACE-RTL and Agentrys report an **Agentic Pass Rate** — the fraction of unique problems solved across many trials — and use large search budgets. ACE-RTL uses 5 parallel processes with up to 30 iterations each, meaning up to approximately 150 attempts per problem, taking any success. The CVDP paper's published single-turn figures are pass@1 with n = 5, a stricter measure. Best-of-many-attempts and pass@1 are not the same quantity. SAiGE figures are single-trajectory pass rates in the specification-only setting and a small-bounded-attempt pass@k.

Anomalous baselines. The Agentrys write-up lists its zero-shot Claude Opus 4.5 baseline at 50.1% overall — above the best single-turn result reported in the CVDP paper. This suggests a different subset and/or a more permissive measurement protocol, so the 95.8% headline should not be placed on the same axis as published pass@1 numbers without adjustment. Agentrys is also a self-published company post, without per-problem disclosure or peer review.

Authorship overlap with the benchmark. The CVDP benchmark and both higher-scoring systems share authors. The teams reporting state-of-the-art on CVDP are, in part, the teams that authored it

and hold its golden solutions and hidden harnesses. This is a structural asymmetry relative to an external evaluator that never sees the harness; it is noted here as context, not as evidence of misuse.

No evidence of data contamination. We found no evidence that the higher scores stem from training on CVDP-derived data. CVDP is described as human-authored from scratch to avoid the GitHub-overlap problem of prior benchmarks, and ACE-RTL's 1.7M training pairs are drawn from public RTL with explicit Jaccard-similarity decontamination against benchmark golden solutions. The more accurate characterization is in-distribution specialization — a model and architecture tuned to a task taxonomy that mirrors CVDP — not leakage.

6. Analysis

6.1 Where Iterative Debug Succeeds

The closed-loop architecture is most effective when:

- The specification is unambiguous and the algorithm is well-defined.
- Errors are structural, such as wrong port widths or missing connections.
- Errors are timing-related, such as off-by-one counters.
- The agent's self-written testbench exercises the same cases the harness tests.

In these cases, SAiGE typically converges within 2–5 simulation iterations, and waveform analysis enables targeted fixes rather than guesswork.

6.2 What the Sanitized Feedback Loop Adds

The 64% to 83% improvement comes entirely from giving the agent a whether-correct signal it otherwise lacks. In the specification-only setting, the dominant failure is silent specification misinterpretation — the agent's own testbench passes because it encodes the same misunderstanding as the RTL. A small amount of external ground truth, such as "configuration X still fails," is often enough to break that symmetry: the agent re-reads the specification, finds the divergent clause, and fixes it.

Critically, this requires no exposure to the test itself — only the verdict — which makes it a deployable signal that preserves the agent's blindness to the grader. This is similar to what a CI regression gate provides in practice.

6.3 Where It Still Fails

After pass@k, the residual failures cluster into:

- **Fixed-point DSP precision** — correct algorithm, wrong bit-width or rounding behavior; several phase_rotation variants plateau even across all three attempts.
- **Plateaued partial passes** — for example, a run-length encoder stuck at 5/7 tests across attempts; more tries do not help without a different approach.
- **Capability ceilings** — complex hierarchical encoders or decoders that exhaust the iteration or time budget mid-convergence.

- **Harness-infrastructure issues**, orthogonal to the agent — a few problems whose local harness hangs or has a non-standard file layout.

Heavy iteration shows diminishing returns once the agent has converged on its own, possibly wrong, interpretation. The sanitized verdict helps most in the first one or two corrective attempts.

7. Limitations

- Specification-only results are single-run; pass@k uses a small, bounded number of attempts. LLM non-determinism means individual problems may flip on retry.
- The agent uses Icarus Verilog, which has known limitations with some SystemVerilog constructs, such as indexed part-selects in always blocks. A few failures may be simulator specific.
- The pass@k setting is a different and more permissive evaluation than specification-only because it adds a pass/fail signal. The two settings are reported separately, and only the specification-only numbers are compared directly against published single-turn results.
- Comparisons to ACE-RTL and Agentrys are not directly commensurable. They evaluate on a different and easier problem split, report best-of-many-attempts rather than pass@1, and share authorship with the benchmark. Section 5.4 provides the full set of caveats. Their numbers should be read as directional, not as a strict ranking.
- Multi-model evaluation on CVDP using DeepSeek, Qwen, and Haiku has not been conducted here.

8. Conclusion

The CVDP benchmark exposes the capability frontier for agentic RTL development. SAiGE, as a general-purpose agent with no RTL-specific training and no CVDP-tuned architecture, reaches 64% in the specification-only setting — more than 2× the best published single-turn result — and 83% with a minimal, sanitized pass@k feedback loop. CVDP-specialized systems score higher: ACE-RTL at 88.9% via a 1.7M-sample fine-tuned model, and Agentrys at 95.8% via a seven-generation, 16-agent network. However, those systems trade generality for benchmark-specific specialization and large search budgets.

The decisive lever for a general-purpose agent is specification comprehension. When the agent's understanding diverges from the golden model, no amount of self-directed iteration can fully bridge the gap. A single sanitized bit of external ground truth — whether the design passed — closes much of that gap while keeping the agent blind to the grader's internals.

The path to closing the remainder likely runs through better specification-ambiguity reasoning and fixed-point/DSP precision handling, rather than further elaboration of the debug loop alone.

References:

- Deng, C., Yu, Z., Liu, G., Pinckney, N., Khailany, B., & Ren, H. (2026, February 10). *ACE-RTL: When Agentic Context Evolution Meets RTL-Specialized LLMs*. arXiv.org. https://arxiv.org/abs/2602.10218

- Pinckney, N., Deng, C., Ho, C., Tsai, Y., Liu, M., Zhou, W., Khailany, B., & Ren, H. (2025, June 17). *Comprehensive Verilog Design Problems: a Next-Generation benchmark dataset for evaluating large language models and agents on RTL design and verification*. arXiv.org. https://arxiv.org/abs/2506.14074

- *QOR Highlights: CVDP — Agentrys*. (n.d.). https://agentrys.ai/blog-cvdp.html

- Deng, C., Tsai, Y., Liu, G., Yu, Z., & Ren, H. (2025). *ScaleRTL: Scaling LLMs with Reasoning Data and Test-Time Compute for Accurate RTL Code Generation* (pp. 1–9). https://doi.org/10.1109/mlcad65511.2025.11189212

- Liu, S., Fang, W., Lu, Y., Wang, J., Zhang, Q., Zhang, H., & Xie, Z. (2024). RTLCoder: Fully Open-Source and efficient LLM-Assisted RTL Code Generation technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *44*(4), 1448–1461. https://doi.org/10.1109/tcad.2024.3483089

- Liu, S., Lu, Y., Fang, W., Li, M., & Xie, Z. (2024). *OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation* (pp. 1–9). https://doi.org/10.1145/3676536.3697118

- Lu, Y., Liu, S., Zhang, Q., & Xie, Z. (2024). *RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model* (pp. 722–727). https://doi.org/10.1109/asp-dac58780.2024.10473904

- Min, K., Cho, K., Jang, J., & Kang, S. (2026). *REvolution: An Evolutionary Framework for RTL Generation driven by Large Language Models* (pp. 282–288). https://doi.org/10.1109/asp-dac66049.2026.11420420

- Nguyen, V., Do, D., Nguyen, N., & Le, D. (2025). *COMBA-PROMPT: Comprehensive Benchmark and Augmentation for Verilog Generation Leveraging Dual-LLM Prompting*. https://doi.org/10.36227/techrxiv.175339240.04076578/v2

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*