Structured Human-Like Agentic RTL Design Flow

Yu-Tung Liu
University of Maryland
College Park, MD, USA
liuyt@umd.edu

Zhan Song
University of Maryland
College Park, MD, USA
zhansong@umd.edu

Chenhui Deng
NVIDIA
Santa Clara, CA, USA
cdeng@nvidia.com

Chia-Tung Ho
NVIDIA
Santa Clara, CA, USA
chiatungh@nvidia.com

Cunxi Yu
University of Maryland
College Park, MD, USA
cunxiyu@umd.edu

Abstract

Long-horizon RTL agents often collapse under context pollution, lacking the active, compartmentalized debugging strategies of human engineers. We introduce OneVeriAgent, a human-like framework modeled directly on expert cognitive workflows. First, mimicking how engineers probe waveform viewers, Agentic Temporal Exploration (ATE) enables dynamic, targeted signal queries, distilling dense simulator dumps into sparse, causal traces. Second, reflecting how designers compartmentalize tasks, a task-scoped Orchestrator enforces strict context isolation, preventing raw tool outputs from polluting the global reasoning trace. On the CVDP benchmark, OneVeriAgent achieves 97.7% APR with Claude Opus 4.6, matching a state-of-the-art industrial agent swarm system. Finally, we propose Tool-Validated Self-Distillation (TVSD) to bridge the open-source capability gap. By fine-tuning on self-generated, syntactically valid trajectories without outcome filtering, TVSD lifts Gemma4-31B-Instruct's Pass@1 from 61.1% to 75.2%, demonstrating that fully open-weight RTL agents are viable.

1 Introduction

Autonomous LLM agents increasingly tackle complex engineering tasks by interleaving reasoning with tool execution over long-horizon trajectories [1]. However, as these multi-turn interactions lengthen, the agent's context window becomes heavily saturated with raw execution logs, compiler errors, and histories of failed repair attempts. This unregulated accumulation severely degrades the model's reasoning capabilities, leading to a critical failure mode we term *context pollution* [1, 2].

Register-transfer-level (RTL) design acutely amplifies this challenge due to its strict, cycle-accurate simulation environments. Recent training-driven approaches [3–5] have largely saturated on compositional benchmarks like VerilogEvalv2 [6]. Consequently, the primary bottleneck has shifted from zero-shot generation to the agent's ability to iteratively verify, diagnose, and refine its policy. While closed-loop systems [6–8] incorporate simulator feedback, current implementations either overwhelm a flat agent with dense VCD waveforms or partition tasks across static, shared-context pipelines. Neither strategy isolates the state space or scales effectively to long-horizon debugging. Furthermore, many frameworks expose the golden testbench directly to the agent, inadvertently bypassing the realistic engineering requirement to deduce design intent under partial observability.

To overcome these limitations, we propose **OneVeriAgent**, a structured, human-like agentic flow modeled directly on the cognitive workflows of expert RTL designers. First, mirroring how engineers actively probe commercial waveform viewers, **Agentic Temporal Exploration (ATE)** enables

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

a debugging subagent to dynamically query specific signals across targeted time windows, distilling dense waveform dumps into sparse, causal traces. Second, reflecting how human designers compartmentalize working memory between tasks, **Task-Scoped Orchestration** isolates state by dispatching subagents with task-scoped contexts. This confines raw tool outputs to the generating subagent, systematically eliminating context pollution. Finally, interacting with the testbench strictly as a black-box oracle ensures the agent infers design intent under realistic partial observability.

To demonstrate that OneVeriAgent natively supports offline policy improvement, we introduce **Tool-Validated Self-Distillation (TVSD)**. TVSD fine-tunes a single open-weight model to seamlessly execute all agentic roles using solely its own multi-level interaction traces. We sample trajectories from Gemma4-31B-Instruct on VerilogEval-v2 and RTLLM-v2 [9], filtering them strictly by syntactic tool-call validity. Crucially, this supervised fine-tuning (SFT) pipeline relies entirely on self-generated data, requiring neither outcome filtering nor distillation from a proprietary teacher. This self-alignment strategy yields a 14.1% performance improvement on the held-out CVDP benchmark, substantially bridging the capability gap with closed-source backbones.

Contributions. In summary, our key contributions are:

- **Architecture for long-horizon tool-use RTL agents.** We formulate *Agentic Temporal Exploration* and *task-scoped orchestration* as a unified solution to context pollution. ATE provides action-space completeness over the spatiotemporal observation space, while *Task-Scoped Orchestration* ensures raw tool logs never degrade the global reasoning trace.
- **Industrial-frontier performance on CVDP and beyond.** OneVeriAgent reaches **97.7%** APR on CVDP with Opus 4.6, matching the strongest closed-source industrial agents and surpassing the strongest closed-source academic system by **8.8%**. We also set new state-of-the-art on VerilogEval-v2 (100%), RTLLM-v2 (100%), VerilogEval-v2-EXT Hard (95.5%), and ChipBench (84.5%, vs. 37.8% for the strongest prior agentic baseline).
- **Closing the open-weight gap via Tool-Validated Self-Distillation.** We present *Tool-Validated Self-Distillation*, a training procedure that fine-tunes a single open-weight model to play every role in the agentic structure using only its own raw trajectories, filtered only by syntactic tool-call validity. TVSD lifts Gemma4-31B-Instruct from 61.1% to 75.2% Pass@1 on CVDP, the strongest reported result for a fully open-weight agentic system on this benchmark.

2 Background

Long-horizon agentic learning Deploying autonomous language models in engineering domains requires agents to dynamically construct long-horizon trajectories [10]. While the ReAct paradigm [11] provides the foundational loop of interleaved reasoning and tool execution, optimizing these highly stateful interactions is structurally bottlenecked by sparse and delayed environmental feedback. Consequently, recent approaches have shifted away from static, single-turn preference alignment. Frameworks such as Rejection Fine-Tuning (RFT) [12] and Dynamic Sampling Policy Optimization (DAPO) [13] allow models to optimize multi-turn reasoning policies directly from their own execution histories.

Despite these algorithmic gains, managing massive observation spaces remains a critical architectural bottleneck. Agents are routinely bombarded with dense simulator feedback and functionally similar distractors [14]. Forcing a flat LLM agent to absorb this undifferentiated feedback rapidly induces severe performance degradation, frequently characterized as "context pollution" [2]. Driven by the mathematical dilution of self-attention weights and the introduction of semantic noise, this phenomenon severely degrades reasoning capacity [15]. As the signal-to-noise ratio collapses, agents succumb to hallucinated variables and catastrophic forgetting, demonstrating long-horizon autonomy requires structural context orchestration rather than mere parameter scaling [16].

Observation spaces in software and RTL debugging While debugging in software and RTL domains both relies on analyzing execution evidence, they present structurally divergent observation problems. In procedural software, bugs manifest sequentially along a single thread of execution. Agents can cleanly backtrack through a linear chain of cause and effect, an observation space that naturally aligns with the one-dimensional context window of LLMs [1, 17, 18].

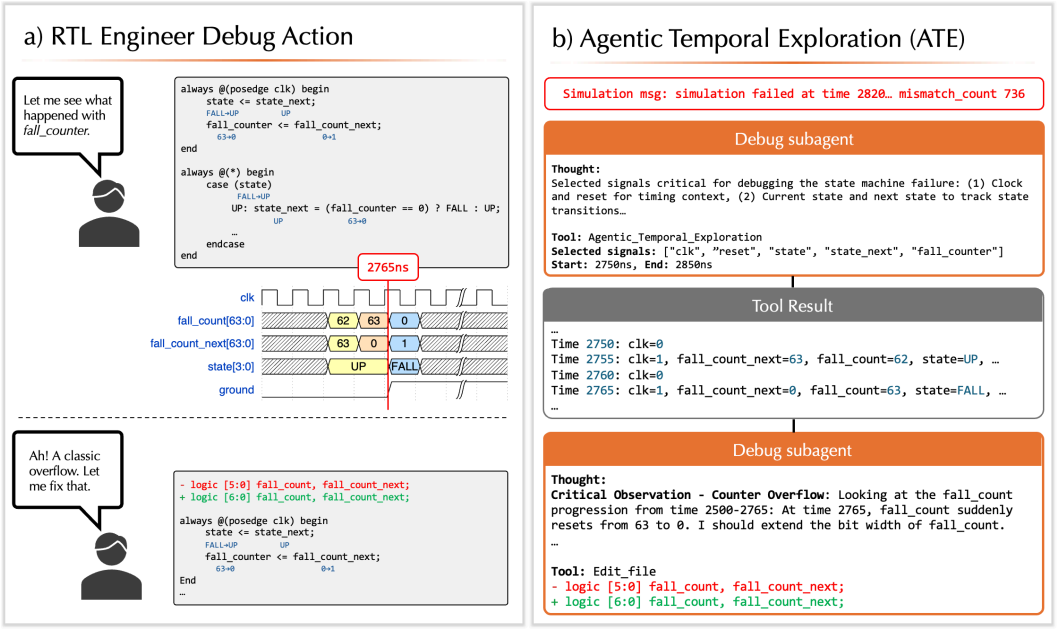


Figure 1: **ATE mirrors the human RTL debugging workflow. (a)** An engineer manually isolates suspect signals and targeted time windows in a waveform GUI to diagnose a bit-width overflow on a state counter. **(b)** The Debug subagent performs the same loop via the ATE tool, observing the same sparse causal trace to emit the correct edit to the RTL source.

Conversely, RTL designs evolve concurrently. At each clock edge, thousands of registers update in strict parallel. A failure observed at simulation time t_{err} is rarely the fault's origin. It is typically a delayed manifestation propagated through complex combinational paths [19]. The RTL observation space is therefore not a linear trace, but a massive, two-dimensional spatiotemporal surface of signals over time. Directly porting software techniques to EDA, such as dumping a raw VCD waveform as a naive print-equivalent, exposes the LLM to tens of thousands of irrelevant transitions and induces catastrophic context pollution [20].

Agentic LLM workflows for RTL design As static RTL generation approaches its capability limits [3, 4], the frontier has shifted toward closed-loop multi-agent frameworks [7, 8, 21] and evolutionary methods [22, 23]. While simulation-in-the-loop debugging markedly improves functional correctness, current pipelines attempt to manage the observation space through restrictive heuristics that severely bottleneck long-horizon reasoning.

Flat multi-agent architectures induce observation collapse to survive the context pollution inherent to dense waveform logs [24]. They artificially restrict the state space to a single temporal checkpoint [7] or rely on static AST back-tracing [8]. This restriction blinds agents to the emergent, cross-cycle temporal interactions necessary for true root-cause analysis. Furthermore, many frameworks provide access to the content of golden testbench instead of treating it as simulation environment [6]. This inadvertently couples the agent to a hidden oracle, bypassing the realistic engineering requirement to deduce design intent from the natural language specification and environment feedback alone.

3 Methodology

The OneVeriAgent architecture rests on two observations about expert RTL workflows. Engineers isolate faults by probing specific simulation windows rather than parsing flat execution logs, motivating *Agentic Temporal Exploration* (§3.1). They also compartmentalize attention across distinct cognitive tasks, motivating *Task-Scoped Orchestration* (§3.2).

3.1 Agentic Temporal Exploration

Insights from human RTL verification. When an RTL simulation reports a mismatch, a human designer does not examine the failing timestamp in isolation. Instead, using commercial tools like Synopsys Verdi [25], their diagnostic workflow is highly iterative. The designer selects suspect signals, inspects their transitions over a targeted time window alongside the Design Under Test (DUT) source, and traces backward to confirm or revise hypotheses. Crucially, both the signal set and time window are dynamically updated as their understanding evolves. Figure 1(a) illustrates this process on a state-machine bug: suspecting `fall_counter` is involved in an incorrect FALL→UP transition, the engineer isolates this signal and surrounding state registers around the failure timestamp. By observing a counter overflow from `63` to `0`, they readily conclude the bit width is too narrow. Ultimately, root-cause diagnosis is driven not by passively reading the entire waveform, but by actively selecting the right slice.

Action formulation. Let $\mathcal{W}$ denote the full simulated waveform for the DUT C. ATE is exposed to the debugging subagents as a discrete tool whose action takes the form

$$a_i^{\text{ATE}} = \big(\sigma_i,\ t_i^{\text{start}},\ t_i^{\text{end}}\big), \tag{1}$$

where σ_i specifies the chosen subset of probed signals and $[t_i^{\text{start}}, t_i^{\text{end}}]$ bounds the temporal window of interest. The tool returns a sparse representation of the waveform restricted strictly to $(\sigma_i, [t_i^{\text{start}}, t_i^{\text{end}}])$, as formalized below.

The number of exploratory rounds is dynamically bounded, and the subagent terminates the ATE loop once its hypothesis is sufficiently constrained to confidently act upon. Figure 1(b) illustrates one such diagnostic loop, directly mirroring the manual root-cause analysis performed by the human engineer in Figure 1(a).

Sparse temporal representation as a state-representation choice. In commercial waveform viewers, this representation choice is implicit: the GUI continuously renders every selected signal's value at every timestamp within the window. LLM agents, however, operate under strict context limits. A literal textual port of the GUI would emit one row per signal per clock tick, meaning the token cost would scale with the raw duration of the simulation rather than the actual behavioral complexity it contains. We instead return a sparse representation that preserves only state transitions, mirroring what an engineer's eye actually attends to.

The shape of the observation returned by the ATE tool is structurally optimized for LLM context constraints. A naive implementation would return the dense matrix

$$W^{\text{dense}} \in \mathbb{R}^{|\sigma| \times T_{\text{cyc}}}, \tag{2}$$

where T_{cyc} is the number of simulated clock cycles within $[t^{\text{start}}, t^{\text{end}}]$. The resulting token cost of W^{dense} scales as $\mathcal{O}(|\sigma| \cdot T_{\text{cyc}})$, inducing severe attention dilution from signals that hold their logic states across consecutive cycles. To eliminate this redundancy, we instead return an initial temporal snapshot alongside the sequence of active transitions on the queried signals:

$$W^{\text{sparse}} = \Big(w_\sigma(t^{\text{start}}),\ \ \big\{(t_k, s_k, v_k) : s_k \in \sigma,\ v_k \neq w_{s_k}(t_k - 1),\ t_k \in [t^{\text{start}}, t^{\text{end}}]\big\} \Big), \tag{3}$$

where $w_\sigma(t)$ is the value vector at time t.

This representation provides two critical advantages. First, the size of W^{sparse} scales precisely as $\mathcal{O}(|\sigma| + |\Delta_\sigma|)$, where the transition count $|\Delta_\sigma|$ depends strictly on behavioral complexity rather than raw simulation duration. A ten thousand cycle simulation with one state change yields an observation space linear in $|\sigma|$ plus one event, completely independent of T_{cyc}. Second, this sparsity shifts the agent's focus onto informative events. Figure 1(b) demonstrates this: the discrete transition $(t = 2765,\ \texttt{fall_count} = 0)$ following $(t = 2755,\ \texttt{fall_count} = 63)$ exposes a clear causal link between counter saturation and state transition. Stripping away redundant cycles eliminates attention dilution, fostering an expert agentic flow where the model isolates causality exactly as a human engineer does.

3.2 Task-Scoped Orchestration

Emulating the cognitive task of RTL design A human designer debugging a module does not rely on a fixed team of specialists sharing a global context. Instead, they operate in a dynamic flow,

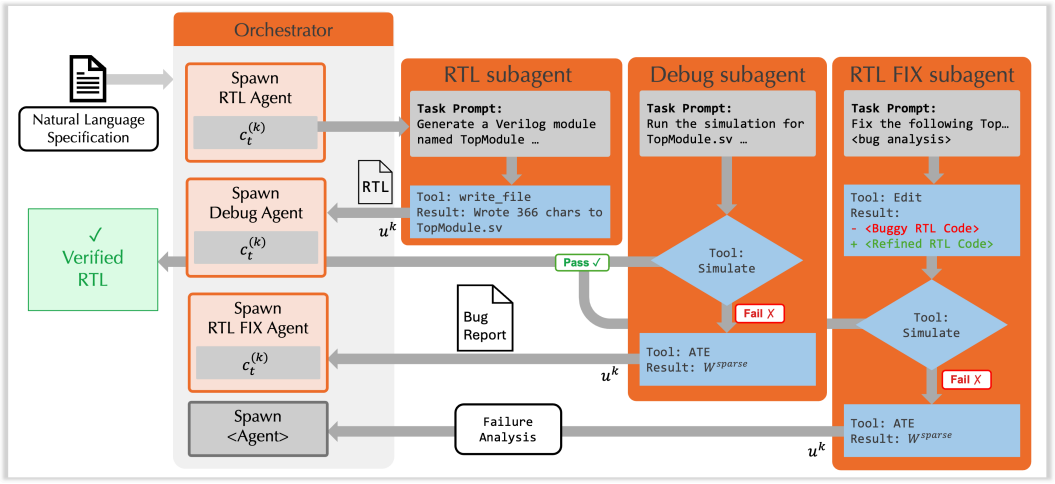


Figure 2: OneVeriAgent inference architecture. The Orchestrator receives the natural-language specification and dispatches subagent tasks one at a time, each with a freshly constructed task-scoped context. Each subagent runs its own bounded ReAct loop over a scoped action set and returns only a textual summary to the Orchestrator. Raw tool outputs never escape the subagent that produced them. The hierarchy terminates when simulation passes or the agent depletes its iteration budget.

alternating between distinct cognitive tasks like writing code, analyzing simulation logs, probing waveforms, and applying edits. Each task requires a different set of artifacts in working memory. When writing a module, the specification matters; when isolating a bug, waveform values take precedence. Working memory is filtered by *the task at hand*, not by static role assignments.

Structured decomposition. To mirror this human-like agentic flow, OneVeriAgent partitions the global execution trace τ from Eq. 5 into an outer Orchestrator trace and a sequence of inner subagent traces, as shown in Figure 2. The Orchestrator π^{orc} maintains the high level problem state. At each step, it executes a direct action or issues a `Task` to dispatch a specialized subagent $k \in \{\text{RTL}, \text{DEBUG}, \text{RTL_FIX}\}$. Each subagent executes a bounded ReAct loop within a scoped action space and returns control alongside a concise textual summary $u^{(k)}$. By observing only this terminal summary instead of the dense internal trace, the Orchestrator avoids global context pollution.

Task-scoped context construction. This architectural isolation is enforced at dispatch. Let $\tau_{<t}^{\text{orc}}$ be the Orchestrator trace prefix when issuing a `Task`. Rather than passing this entire history, the Orchestrator constructs a strictly filtered context:

$$c_t^{(k)} = \phi_k\left(\tau_{<t}^{\text{orc}},\ p_t^{(k)}\right), \tag{4}$$

where $p_t^{(k)}$ is the dispatch prompt and ϕ_k is a deterministic projection retaining only artifacts relevant to scope k. Specifically, for RTL, ϕ_k keeps only the specification $\mathcal{S}$ and interface. For DEBUG, it isolates $\mathcal{S}$, the source C, and the mismatch summary, purging prior outputs. For RTL_FIX, it provides only C and the bug hypothesis u^{debug} from the preceding DEBUG call. Notably, raw waveform transitions are explicitly dropped here, having already been distilled into the hypothesis. The Orchestrator remains the sole component holding the full τ^{orc}, ensuring focused memory allocation. Appendix A details these specifications.

3.3 Tool-Validated Self-Distillation

While the OneVeriAgent architecture in §3.2 excels with proprietary models, closed APIs hinder reproducible research. Even Self-Distillation Fine-Tuning (SDFT) relies on proprietary traces as expert demonstrations [26]. To natively align open-weight models with our human-like agentic flow, we propose Tool-Validated Self-Distillation (TVSD). Extending the simple self-distillation (SSD) paradigm [27], TVSD samples rollouts directly from the base model, filters them strictly on syntactic tool-call validity, and applies SFT back to the base model.

Trace formalism. A OneVeriAgent rollout generates a hierarchical trajectory that reflects the architecture defined in §3.2. The Orchestrator interacts with the environment via the global sequence:

$$\tau^{\text{orc}} = (o_1, a_1, \ldots, o_T, a_T), \tag{5}$$

where each action a_t is either a direct operation or a `Task` dispatch. A dispatch delegates control to a specialized subagent $k \in \{\text{RTL}, \text{DEBUG}, \text{RTL_FIX}\}$ seeded with a strictly isolated context $c_t^{(k)}$ (Eq. 4). This generates an inner trace $\tau_j^{(k)}$ constrained to a scoped action space. We define the complete rollout as:

$$\mathcal{T} = \left(\tau^{\text{orc}},\ \{\tau_j^{(k)}\}_{j=1}^{J}\right). \tag{6}$$

An external verification harness evaluates the final source state $C_{\mathcal{T}}$ to compute terminal correctness $R(\mathcal{T}) = \mathbb{1}[\textsc{Simulate}(C_{\mathcal{T}}, \mathcal{T}_{\text{golden}}) = \text{pass}]$.

Trajectory collection and tool-validity filter. We sample K rollouts from the base policy π_{θ_0} across a training pool of VerilogEval-v2 and RTLLM-v2 problems, strictly holding out CVDP. We discard any trajectory $\mathcal{T}$ containing malformed tool calls, specifically if (i) a tool schema fails to parse, or (ii) an ATE call omits temporal bounds (`start_time` or `end_time`). Enforcing bounded queries ensures ATE operates as an active, human-like diagnostic probe rather than a dense waveform logger (§3.1). Crucially, we do *not* filter on terminal correctness $R(\mathcal{T})$ or trajectory length. Consistent with SSD principles, syntactically valid self-samples provide distributional information regardless of terminal correctness.

Training objective. We expand each retained rollout into a dataset of supervised instances $\mathcal{D}$ formatted as observation-action pairs (x, y). An Orchestrator turn at step t yields $(x, y) = (\tau_{<t}^{\text{orc}}, a_t)$, whereas a subagent turn at inner step i yields $(x, y) = (c_t^{(k)} \| \tau_{j,<i}^{(k)}, a_i)$. Both instance types are optimized jointly under a standard autoregressive loss:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\sum_{t=1}^{|y|} m_t \log \pi_\theta(y_t | x, y_{<t}) \right], \tag{7}$$

where $m_t \in \{0, 1\}$ masks the loss to evaluate only assistant tokens. Crucially, we train a single, unified policy θ across Orchestrator and subagents. The model dynamically assumes specialized roles driven by the task-scoped context, directly mirroring how a human engineer seamlessly shifts cognitive focus across distinct design tasks. Implementation details are provided in Appendix B.

4 Experiment

We evaluate OneVeriAgent across five diverse benchmarks that span single-module generation, complex system-level RTL design, and difficulty-stratified debugging tasks to comprehensively assess its long-horizon reasoning capabilities.

Comprehensive Verilog Design Problems (CVDP) [28] serves as our primary evaluation suite to stress-test agentic workflows on industrial-scale hardware challenges requiring rigorous external harness verification. We evaluate on all four core code-generation and refinement subsets: Code Completion (`cid002`, 94 problems), Natural Language Spec to Code (`cid003`, 78 problems), Code Modification (`cid004`, 55 problems), and Debugging / Bug Fixing (`cid016`, 35 problems), totaling 262 expert-reviewed hardware design tasks.

To demonstrate architectural robustness beyond CVDP, we also evaluate our framework on four standard and latest RTL benchmarks:

- **VerilogEval-v2** [6]: The established baseline for specification-to-RTL generation, consisting of 156 self-contained functional verification problems.
- **RTLLM-v2** [9]: A structurally diverse set of 50 design tasks spanning Arithmetic, Control, Memory, and Miscellaneous blocks that are significantly more challenging than VerilogEval-v2.
- **VerilogEval-v2-EXT** [29]: A difficulty-stratified extension categorizing problems into Easy ($n = 51$), Medium ($n = 91$), and Hard ($n = 67$) splits, explicitly designed to isolate where multi-agent orchestration yields diminishing or expanding returns.

- **ChipBench** [30]: A system-level benchmark that differentiates between self-contained components, non-self-contained modules, and complex CPU IP blocks to evaluate cross-file dependency resolution and multi-module integration.

4.1 Experiment results

CVDP. Baselines drawn from industrial agent reports (Agentrys [31], ChipCraftBrain [32]) and the closed-source academic system ACE-RTL [21] report Agentic Pass Rate (APR), the fraction of problems uniquely solved across multiple runs, which by construction upper-bounds Pass@1 at matching n. With Opus 4.6 [33], OneVeriAgent reaches **97.7%** overall on Table 1, surpassing ACE-RTL (88.9%; fine-tuned generator plus two context managers on Claude Sonnet 4 [34]) and matching the industrial agents Agentrys (95.8%, a 16-agent swarm) and ChipCraftBrain (94.3%) according to their reported APR. The gap to training-only systems run inside an agentic harness is larger still. For other evaluation, we report Pass@1 over $n = 5$.

Table 1: Evaluation on CVDP. All entries report Agentic Pass Rate (APR). The first and second highest accuracies are annotated.

Method	Model	cid002 APR (%)	cid003 APR (%)	cid004 APR (%)	cid016 APR (%)	Overall APR (%)
Generic LLM	Opus 4.6	50.0	56.4	52.7	57.1	53.4
Training Based w/ Agentic Setting						
ScaleRTL	ScaleRTL-32B	29.8	35.9	32.7	40.0	33.6
Industrial Chip Design Agent						
ChipCraftBrain	ChipCraftBrain	93.6	**96.2**	**96.4**	88.6	94.3
Agentrys	16 Agent Swarm	**96.8**	**96.2**	90.9	**100**	**95.8**
Multi-Agent Systems						
ACE-RTL	Hybrid	80.9	**96.2**	90.9	**91.4**	88.9
OneVeriAgent (Ours)	Opus 4.6	**98.9**	**98.7**	**92.7**	**100**	**97.7**

VerilogEval-v2 and RTLLM-v2. Table 2a reports results on the two most popular self-contained benchmarks [21]. Most multi-agent systems are within a few points of saturation on VerilogEval-v2. We report 100% with Opus 4.6. RTLLM-v2 is more discriminating: OneVeriAgent reaches 100%. We treat these benchmarks as evidence that the architecture does not regress on the easier setting that prior work focused on, rather than as the primary evaluation.

ChipBench and VerilogEval-v2-EXT. Table 2b reports results on ChipBench and Table 2c on VerilogEval-v2-EXT. On ChipBench, OneVeriAgent (Opus 4.6) reaches 84.5% overall, compared to 37.8% for the same backbone in single-pass mode and 37.8% for MAGE on DeepSeek-V3, which is primarily designed for self-contained problems. The gap on the CPU IP subset (88.9% vs 22.2%) is the most informative entry in the table: cross-file context is precisely the regime where flat-context agents collapse. On VerilogEval-v2-EXT, OneVeriAgent reaches 95.5% on the Hard subset, compared to 85.1% for Veri-Sure and 77.6% for MAGE.

4.2 TVSD results

Table 3 reports the effect of TVSD on OneVeriAgent with the open-weight backbone Gemma4-31B-Instruct. Across all four CVDP subsets, the self-distilled model (GEMMA4-31B$_{\text{TVSD}}$) improves over the same base model run in the same agentic loop. Overall CVDP Pass@1 rises from 61.1% to 75.2%, an improvement of 14.1%. We find that before TVSD, the vanilla backbone often cannot reason correct tool usage, leading to huge retrying cycles that deplete the iteration budget. The result is consistent with the SSD claim of [27] that distributional reshaping is recoverable from raw self-samples and does not require outcome filtering, and it generalizes that claim to multi-level agentic trajectories with token-level masking on assistant turns.

Table 2: Table of VerilogEval-v2, RTLLMv2, ChipBench, and VerilogEvalv2-EXT. † denotes APR. The first and second highest accuracies are annotated.

Method	Model	VerilogEval-v2 Pass@1 (%)	RTLLM-v2 Pass@1 (%)
Generic LLM	Opus 4.6	85.9	70
SiliconMind-V1 [35]	SiliconMind-V1	76.5	67.9
ScaleRTL [5]	ScaleRTL-32B	93.6	N/A
ACE-RTL [21]	Hybrid	95.5	N/A
VeriAgent [36]	GPT-5.4-mini	92.2	83.2
MAGE [7]	Claude 3.5 Sonnet	95.7	N/A
VerilogCoder [8]	GPT4-Turbo	94.2	N/A
VerilogCoder [8]	GPT-5.3	**100**	N/A
REvolution [37]	GPT-5.4-mini	93.5	85.4
COEVO [38]	GPT-5.4-mini	97.5	**94.5**
EvolVE [22]	Deepseek-R1	**98.1**†	92†
OneVeriAgent (Ours)	Opus 4.6	**100**	**100**

(a) VerilogEval-v2 and RTLLM-v2.

Method	Model	Self-Contained Pass@1 (%)	Non-Self-Contained Pass@1 (%)	CPU IP Pass@1 (%)	Overall Pass@1 (%)
Generic LLM	Opus 4.6	**40.0**	**50.0**	**22.2**	**37.8**
MAGE	DeepSeek-V3	**40.0**	**50.0**	**22.2**	**37.8**
ChipCraftBrain	ChipCraftBrain	36.7	**50.0**	11.1	33.4
OneVeriAgent (Ours)	Opus 4.6	**86.7**	**66.7**	**88.9**	**84.5**

(b) ChipBench.

Method	Model	Easy (n = 51) Pass@1 (%)	Medium (n = 91) Pass@1 (%)	Hard (n = 67) Pass@1 (%)	Overall Pass@1 (%)
Generic LLM	Opus 4.6	86.3	83.5	65.7	78.5
MAGE	GPT-5.2	**96.1**	**95.6**	77.6	90.0
VerilogCoder	GPT-5.2	94.1	90.1	68.7	84.2
Veri-Sure	GPT-5.2	**100.0**	**95.6**	**85.1**	**93.3**
OneVeriAgent (Ours)	Opus 4.6	**100.0**	**98.9**	**95.5**	**98.1**

(c) VerilogEval-v2-EXT.

Table 3: Performance improvements of the open-weight Gemma4-31B-Instruct model after TVSD.

Model	**CVDP-cid002 Pass@1 (%)**	**CVDP-cid003 Pass@1 (%)**	**CVDP-cid004 Pass@1 (%)**	**CVDP-cid016 Pass@1 (%)**	**Overall Pass@1 (%)**
Gemma4-31B-it	48.9	66.7	69.9	68.6	61.1
+TVSD	**68.1**$_{\uparrow 19.2}$	**84.2**$_{\uparrow 17.5}$	**77**$_{\uparrow 7.1}$	**71.4**$_{\uparrow 2.8}$	**75.2**$_{\uparrow 14.1}$

4.3 Ablation study: effectiveness of ATE

To isolate ATE's contribution from that of orchestration, we vary only the debugging subagents' waveform action space. The Orchestrator, subagent scopes, all other tools, iteration budgets, and the Opus 4.6 backbone stay fixed. We compare three variants. In **w/o ATE**, the waveform tool is removed, so the subagents observe only the simulator's mismatch report: which outputs diverged and at which timestamps. In **ATE w/o temporal**, the agent still selects the signal subset σ, but the time window is fixed to the full simulation, so each query returns every transition of the selected signals over the entire run. **Full ATE** is the proposed action $(\sigma, t^{\text{start}}, t^{\text{end}})$ of §3.1. We evaluate on RTLLM-v2 (Pass@1) and on CVDP-cid002, the largest CVDP subset (n=94), where we report APR for consistency with Table 1.

Table 4 shows that both components of the ATE action contribute, and that the temporal window contributes most. On CVDP-cid002, full ATE solves 8 more problems than the variant without waveform access. Signal selection alone accounts for 2 of these 8, and the agent-chosen time window accounts for the remaining 6. RTLLM-v2 shows the same ordering at smaller scale: 1 of the 3 recovered problems comes from signal selection and 2 come from temporal targeting. Without a window, a query's observation grows with the number of transitions over the whole simulation rather than within the region of interest. Long-running counters and periodic signals then dominate

the returned trace, which reintroduces at the observation level the dilution that Eq. 3 is designed to avoid.

Table 4: ATE ablation under fixed orchestration, backbone (Opus 4.6), tools, and budgets. Numbers in parentheses are solved problems.

Variant	Signal selection	Temporal exploration	RTLLM-v2 Pass@1 (%)	CVDP-cid002 APR (%)
w/o ATE	✗	✗	94.0 (47/50)	90.4 (85/94)
ATE w/o temporal	✓	✗	96.0 (48/50)	92.6 (87/94)
Full ATE (Ours)	✓	✓	**100.0** (50/50)	**98.9** (93/94)

The problems that require temporal targeting are also the longest-horizon ones. The two RTLLM-v2 problems that neither ablated variant solves are the two with the longest single-agent trajectories in the benchmark (§4.4). On these, the single agent spends roughly 100 reasoning rounds without reaching a correct design, while the full system converges in roughly 40 rounds in total, including all subagent dispatches. The margin from ATE therefore widens with debugging difficulty, from RTLLM-v2 to CVDP, rather than shrinking. This supports the view that bounded, hypothesis-driven waveform queries matter most where root causes lie many cycles before the observed mismatch.

4.4 Ablation study: context pollution

We test the central claim of §3.2 that *Task-Scoped Orchestration* eliminates the long-context degradation a flat agent exhibits. We compare OneVeriAgent against a Single Agent with the full toolset on VerilogEval-v2 and RTLLM-v2 under the same backbone, tool definitions, and iteration budgets. Table 5 reports pass rate and per-request token statistics. Both configurations saturate at 100% on the easier benchmark, but on the harder RTLLM-v2 the Orchestrator reaches 100% while the Single Agent stalls at 96%. The mechanism is visible in the token statistics: the Single Agent makes fewer, larger calls (12.1k tokens per request, 5.54 requests per problem) because it cannot offload state; the Orchestrator makes a tradeoff of roughly twice as many calls of substantially smaller context (4.7k tokens per request, 11.81 requests per problem) because the task-scoped projection ϕ_k prunes state at every dispatch boundary. The pattern matches the prediction of §3.2: context pollution scales with trajectory length, and a flat agent's accuracy degrades only when problems force long trajectories. Figure 3a plots tokens per request against requests per problem. Single Agent runs cluster at few requests with large contexts and trail into a 32k-token tail; Orchestrator runs spread rightward but stay below 8k tokens per request (Appendix D).

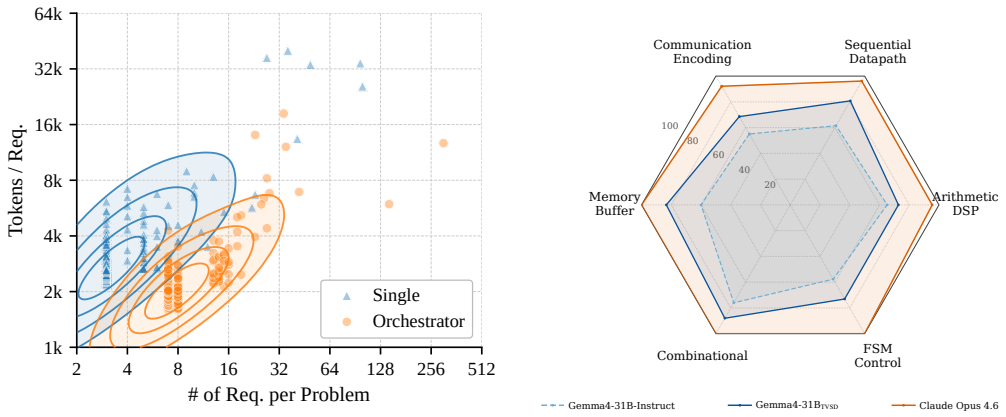
Table 5: Ablation study comparing a single to the proposed orchestrator.

	Tokens per Req.	Tokens per Problem	# of Req. per Problem	VerilogEval-v2 Pass@1 (%)	RTLLM-v2 Pass@1 (%)
Single Agent	12.1k	66.9k	**5.54**	100	96
Orchestrator	**4.7k**	**56k**	11.81	**100**	**100**

5 Discussion and limitations

Per category analysis. We categorize the 262 CVDP problems into six categories (Figure 3b). Communication Encoding is the category where Opus 4.6 finds the most challenge (92.1%), reflecting a domain knowledge gap. The largest vanilla open-weight deficit to Opus lies on FSM Control (42.3%). Conversely, Gemma4-31B$_{TVSD}$ improves most on Memory Buffer (+23.3%), Sequential Datapath (+19.3%), and FSM Control (+15.4%), categories whose bugs surface in multi-cycle state evolution. The SFT trajectories teach the open-weight model the ATE patterns it failed to invoke at baseline, unlocking waveform-driven debugging on these problems (Appendix E).

Limitations Four limitations bound our claims (Appendix F). (i) The absolute ceiling tracks the underlying backbone, 97.7% with Opus 4.6, 75.2% with Gemma4-31B$_{\text{TVSD}}$, though the architectural lift is robust across both. (ii) Credit assignment for RL across an Orchestrator dispatch and inner subagent calls remains an open problem. (iii) The ATE tool reads from a full VCD on disk, a storage



(a) (b)

Figure 3: (a) Tokens per request vs. requests per problem. (b) Per category analysis on CVDP

overhead requiring selective dumping for industrial multi-module designs. (iv) All benchmarks pair a complete spec with a matching golden testbench, leaving debug subagents without recourse when the spec underdetermines intent.

6 Conclusion

We presented OneVeriAgent, a structured agentic system that addresses context pollution in long-horizon tool-use through two coupled mechanisms: ATE, which grants a debugging subagent action-space completeness over signal selection and time windows, and *Task-Scoped Orchestration*, which ensures raw tool output never escapes the subagent context. On CVDP, the flagship industrial RTL benchmark, OneVeriAgent reaches **97.7%** APR with Opus 4.6, matching closed-source 16-agent industrial systems under APR and surpassing the strongest closed-source academic system. TVSD closes much of the open-weight gap, lifting Gemma4-31B-Instruct from 61.1% to 75.2% on the held-out benchmark using only its own raw trajectories. The structural properties that make this architecture both effective at inference and trainable from raw self-samples suggest a path toward fully open-weight agentic EDA systems for verifiable engineering domains beyond RTL.

Acknowledgments and Disclosure of Funding

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under awards HR0011269E189 and HR001125C0058, and by the National Science Foundation (NSF) under awards CCF-2403134, CCF-2349670, and CCF-2428520. The views, opinions, and findings expressed in this material are those of the authors and do not necessarily reflect the official views or policies of DARPA, the NSF, the Department of Defense, or the U.S. Government.

The authors acknowledge the University of Maryland supercomputing resources (`http://hpcc.umd.edu`) made available for conducting the research reported in this paper.

The authors declare no competing interests.

References

[1] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652, 2024.

[2] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[3] Yang Zhang, Rui Zhang, Jiaming Guo, Huang Lei, Di Huang, Yunpu Zhao, Shuyao Cheng, Pengwei Jin, Chongxiao Li, Zidong Du, Xing Hu, Qi Guo, and Yunji Chen. QiMeng-SALV: Signal-aware learning for Verilog code generation. In *Advances in Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=vaosMuNvOt`.

[4] Yaoyu Zhu, Di Huang, Hanqi Lyu, Xiaoyun Zhang, Chongxiao Li, Wenxuan Shi, Yutong Wu, Jianan Mu, Jinghua Wang, Yang Zhao, Pengwei Jin, Shuyao Cheng, Shengwen Liang, Xishan Zhang, Rui Zhang, Zidong Du, Qi Guo, Xing Hu, and Yunji Chen. QiMeng-CodeV-R1: Reasoning-enhanced Verilog generation. In *Advances in Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=ly5DnRIgCZ`.

[5] Chenhui Deng, Yun-Da Tsai, Guan-Ting Liu, Zhongzhi Yu, and Haoxing Ren. ScaleRTL: Scaling LLMs with reasoning data and test-time compute for accurate RTL code generation. In *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD)*, pages 1–9. IEEE, 2025.

[6] Nathaniel Pinckney, Christopher Batten, Mingjie Liu, Haoxing Ren, and Brucek Khailany. Revisiting VerilogEval: A year of improvements in large-language models for hardware code generation. *ACM Transactions on Design Automation of Electronic Systems*, 2025.

[7] Yujie Zhao, Hejia Zhang, Hanxian Huang, Zhongming Yu, and Jishen Zhao. MAGE: A multi-agent engine for automated RTL code generation. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pages 1–7. IEEE, 2025.

[8] Chia-Tung Ho, Haoxing Ren, and Brucek Khailany. VerilogCoder: Autonomous Verilog coding agents with graph-based planning and abstract syntax tree (AST)-based waveform tracing tool. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 300–307, 2025.

[9] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. RTLLM: An open-source benchmark for design RTL generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 722–727. IEEE, 2024.

[10] Alexander Golubev, Maria Trofimova, Sergei Polezhaev, Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Sergey Abramov, Andrei Andriushchenko, Filipp Fisin, et al. Training long-context, multi-turn software engineering agents with reinforcement learning. *arXiv preprint arXiv:2508.03501*, 2025.

[11] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[12] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

[13] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[14] Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts LLM performance. Chroma Technical Report, `https://research.trychroma.com/context-rot`, 2025.

[15] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[16] Haoran Sun, Shaoning Zeng, and Bob Zhang. H-MEM: Hierarchical memory for high-efficiency long-term reasoning in LLM agents. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 341–350, 2026.

[17] Jiangping Huang, Wenguang Ye, Weisong Sun, Jian Zhang, Mingyue Zhang, and Yang Liu. TraceCoder: A trace-driven multi-agent framework for automated debugging of LLM-generated code. *arXiv preprint arXiv:2602.06875*, 2026.

[18] Cheryl Lee, Chunqiu Steven Xia, Longji Yang, Jen-tse Huang, Zhouruixing Zhu, Lingming Zhang, and Michael R. Lyu. UniDebugger: Hierarchical multi-agent framework for unified software debugging. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 18248–18277, 2025.

[19] Ke Xu, Jialin Sun, Yuchen Hu, Xinwei Fang, Weiwei Shan, Xi Wang, and Zhe Jiang. MEIC: Re-thinking RTL debug automation using LLMs. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–9, 2024.

[20] Luca Collini, Baleegh Ahmad, Joey Ah-kiow, and Ramesh Karri. MARVEL: Multi-agent RTL vulnerability extraction using large language models. *arXiv preprint arXiv:2505.11963*, 2025.

[21] Chenhui Deng, Zhongzhi Yu, Guan-Ting Liu, Nathaniel Pinckney, and Haoxing Ren. ACE-RTL: When agentic context evolution meets RTL-specialized LLMs. *arXiv preprint arXiv:2602.10218*, 2026.

[22] Wei-Po Hsin, Ren-Hao Deng, Yao-Ting Hsieh, En-Ming Huang, and Shih-Hao Hung. EvolVE: Evolutionary search for LLM-based Verilog generation and optimization. *arXiv preprint arXiv:2601.18067*, 2026.

[23] Ping Guo, Yiting Wang, Wanghao Ye, Yexiao He, Ziyao Wang, Xiaopeng Dai, Ang Li, and Qingfu Zhang. EvoVerilog: Large langugage model assisted evolution of Verilog code. *arXiv preprint arXiv:2508.13156*, 2025.

[24] Zhuolun He, Yuan Pu, Haoyuan Wu, Tairu Qiu, and Bei Yu. Large language models for EDA: Future or mirage? *ACM Transactions on Design Automation of Electronic Systems*, 30(6): 1–53, 2025.

[25] Synopsys, Inc. Verdi automated debug system. `https://www.synopsys.com/verification/debug/verdi.html`, 2024. Accessed: 2025-02-05.

[26] Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

[27] Ruixiang Zhang, Richard He Bai, Huangjie Zheng, Navdeep Jaitly, Ronan Collobert, and Yizhe Zhang. Embarrassingly simple self-distillation improves code generation. *arXiv preprint arXiv:2604.01193*, 2026.

[28] Nathaniel Pinckney, Chenhui Deng, Chia-Tung Ho, Yun-Da Tsai, Mingjie Liu, Wenfei Zhou, Brucek Khailany, and Haoxing Ren. Comprehensive Verilog design problems: A next-generation benchmark dataset for evaluating large language models and agents on RTL design and verification. *arXiv preprint arXiv:2506.14074*, 2025.

[29] Jiale Liu, Taiyu Zhou, and Tianqi Jiang. Veri-Sure: A contract-aware multi-agent framework with temporal tracing and formal verification for correct RTL code generation. *arXiv preprint arXiv:2601.19747*, 2026.

[30] Zhongkai Yu, Chenyang Zhou, Yichen Lin, Hejia Zhang, Haotian Ye, Junxia Cui, Zaifeng Pan, Jishen Zhao, and Yufei Ding. ChipBench: A next-step benchmark for evaluating LLM performance in AI-aided chip design. *arXiv preprint arXiv:2601.21448*, 2026.

[31] Yun-Da Tsai, Duo Ding, Wuxi Li, and Haoxing Ren. Agentrys: Self-improving agent solving CVDP RTL coding tasks. Blog post, https://agentrys.ai/blog-cvdp.html, February 2026.

[32] Cagri Eryilmaz. ChipCraftBrain: Validation-first RTL generation via multi-agent orchestration. *arXiv preprint arXiv:2604.19856*, 2026.

[33] Anthropic. Introducing Claude Opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, February 2026.

[34] Anthropic. Introducing Claude 4. https://www.anthropic.com/news/claude-4, May 2025. Accessed: 2026-05-05.

[35] Mu-Chi Chen, Yu-Hung Kao, Po-Hsuan Huang, Shao-Chun Ho, Hsiang-Yu Tsou, I Wu, En-Ming Huang, Yu-Kai Hung, Wei-Po Hsin, Cheng Liang, et al. SiliconMind-V1: Multi-agent distillation and debug-reasoning workflows for Verilog code generation. *arXiv preprint arXiv:2603.08719*, 2026.

[36] Yaoxiang Wang, Qi Shi, ShangZhan Li, Qingguo Hu, Xinyu Yin, Bo Guo, Xu Han, Maosong Sun, and Jinsong Su. VeriAgent: A tool-integrated multi-agent system with evolving memory for PPA-aware RTL code generation. *arXiv preprint arXiv:2603.17613*, 2026.

[37] Kyungjun Min, Kyumin Cho, Junhwan Jang, and Seokhyeong Kang. REvolution: An evolutionary framework for RTL generation driven by large language models. In *2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 282–288. IEEE, 2026.

[38] Heng Ping, Peiyu Zhang, Shixuan Li, Wei Yang, Anzhe Cheng, Shukai Duan, Xiaole Zhang, and Paul Bogdan. COEVO: Co-evolutionary framework for joint functional correctness and PPA optimization in LLM-based RTL generation. *arXiv preprint arXiv:2604.15001*, 2026.

[39] Gemma Team and Google DeepMind. Gemma 4: State-of-the-art open models at the edge. https://ai.google.dev/gemma/docs/core/model_card_4, 2026. Accessed: 2026-05-05.

A Subagent specification

RTL. Receives the spec $\mathcal{S}$, the module interface, and any architectural notes from the Orchestrator. Action set: `write`, `Skill`, `finish`. The subagent emits a single Verilog source file and finishes; it does not simulate. Decoupling generation from simulation prevents a single subagent context from accumulating both code-writing and waveform-debugging artifacts. The Orchestrator decides whether the next step is to dispatch DEBUG or to finish.

DEBUG. Receives the spec, the current source C, and a directive to diagnose. Action set: `simulate`, ATE, `read`, `write` (for an agent-authored probe testbench only), `Skill`, `finish`. The subagent runs a bounded loop of simulating, reading mismatch summaries, and querying the waveform via the ATE loop of §3.1. The terminal summary u^{debug} contains either a confirmation that all tests pass or a bug hypothesis specifying which signals diverge from intent, at which times, and a suggested fix.

RTL_FIX. Receives C and u^{debug}. Action set: `simulate`, ATE, `read`, `write`, `edit`, `Skill`, `finish`. Edits the source, re-simulates, and may trace the waveform of the modified design to confirm the fix. Bundling simulate-and-trace into the same subagent that performs the edit reflects the structure of the task: validating an edit is part of the edit itself. This avoids the round-trip back to the Orchestrator that fixed pipelines require after every repair attempt, and lets the subagent iterate on its own fix until the mismatch count strictly decreases or it gives up and returns to the Orchestrator with what it learned.

Additionally, to avoid unnecessarily frequent context switching that drives up the number of requests, RTL_FIX and DEBUG agent can do a quick simulate-and-ATE loop or edit minor logical errors. If it finds the problem out of its reach, it will return to the Orchestrator for the dedicated task scope.

B Tool-Validated Self-Distillation

This appendix expands on §3.3 with detailed mechanics of trajectory collection, the tool-validity filter, instance construction, and the unified-policy argument. Hyperparameter values, distributed-training setup, and corpus statistics appear in §C.

B.1 Trajectory collection

Base policy and sampling configuration. The base policy π_{θ_0} is the open-weight Gemma4-31B-Instruct [39], used at every level of the hierarchy during collection: the same model serves as the Orchestrator and as each subagent type, with the dispatch context $c_t^{(k)}$ (Eq. 4) doing the work of role differentiation. We sample with a fixed training-time temperature T_{train} and a maximum number of steps per Orchestrator and per subagent invocation.

Training problem pool. The training pool $\mathcal{P}_{\text{train}}$ consists of all problems in VerilogEval-v2 and RTLLM-v2. CVDP problems do not enter $\mathcal{P}_{\text{train}}$ at any stage; they are reserved for evaluation. For each $p \in \mathcal{P}_{\text{train}}$, we run K independent rollouts. Each rollout yields a multi-level trajectory $\mathcal{T}$ as in Eq. 6, terminating either when the Orchestrator emits `finish` or when its step budget is exhausted. We retain all such trajectories at this stage; filtering happens next.

Why not filter on $R(\mathcal{T})$? A success filter would discard every rollout in which the agent failed to produce a passing design, and would therefore train the student on a censored slice of the base policy's behavior. Two things would be lost. First, partially successful trajectories contain valid local behavior (correct simulation invocation, well-formed ATE queries, reasonable bug hypotheses) even when the final fix is wrong. This behavior is on-distribution and worth imitating. Second, the SSD result of [27] attributes the gain not to verification but to the distributional reshaping that supervised cross-entropy on temperature-shifted self-samples induces. Censoring on outcome is not part of that mechanism.

B.2 Tool-validity filter

The tool-validity filter is a syntactic check on tool-call format, applied uniformly across Orchestrator and subagent turns. A trajectory $\mathcal{T}$ passes the filter if every model-generated tool call in it satisfies all three conditions below. It is discarded in its entirety otherwise.

1. **Tool exists at the dispatching level.** The tool name must be one of the available tools at the level where the call is issued. The Orchestrator tool palette (`Task`, `read`, `Skill`, `finish`) and the subagent palettes (§3.2) are disjoint in some entries; calling `simulate` from the Orchestrator turn or `Task` from a subagent turn is malformed.
2. **Argument schema parses and is complete.** The tool arguments must parse as JSON and contain every required field declared by the tool's schema. Missing required fields, type mismatches, or extra unrecognized fields each disqualify the call.
3. **ATE calls specify a window.** Every ATE call must include both `start_time` and `end_time`. A call lacking either field would default the tool to its degenerate "dump the entire simulation" behavior, which collapses the active-observation loop that ATE is designed to support (§3.1). Trajectories containing such calls are removed because keeping them would supervise the student to bypass the very mechanism we want it to learn.

The filter is strictly weaker than verification: a trajectory whose final source produces $R(\mathcal{T}) = 0$ but whose every tool call is well-formed and whose every ATE query is windowed remains in the training corpus. This is the sense in which our procedure follows SSD's unverified-sample setting up to the structural requirement that the trajectory be parseable as a valid agentic interaction.

B.3 Instance construction

We expand each retained $\mathcal{T}$ into supervised instances by walking the trajectory and emitting one instance per assistant turn.

Orchestrator instances. At each Orchestrator step t, the instance is

$$(x, y) \;=\; \big(\, \texttt{SYS}_{\text{orc}} \,\|\, \tau^{\text{orc}}_{<t},\;\; a_t \,\big),$$

where $\texttt{SYS}_{\text{orc}}$ is the Orchestrator's system prompt (a fixed string declaring the available Orchestrator tools and the policy of *Task-Scoped Orchestration*), $\tau^{\text{orc}}_{<t}$ is the Orchestrator's observable trace prefix up to and including o_t, and a_t is the next Orchestrator action. If $a_t = \texttt{Task}(k, p_t^{(k)})$, the target consists of the model's reasoning together with the dispatched task type and prompt; the inner subagent trace that this dispatch will produce is *not* part of this instance's target.

Subagent instances. Within the inner trace $\tau_j^{(k)}$ generated by the j-th dispatch, each assistant step i contributes

$$(x, y) \;=\; \big(\, \texttt{SYS}_k \,\|\, c_t^{(k)} \,\|\, \tau^{(k)}_{j,<i},\;\; a_i \,\big),$$

where $\texttt{SYS}_k$ is the subagent system prompt for type k (declaring the scoped action set), $c_t^{(k)}$ is the dispatch context constructed by the Orchestrator at the moment of dispatch (Eq. 4), and $\tau^{(k)}_{j,<i}$ is the subagent's own trace prefix at the moment it must select a_i.

Why instances are split this way. Splitting Orchestrator and subagent instances ensures that the training signal for each role is conditioned on exactly the information that role observes at inference. A RTL instance never has a waveform observation in x, because a RTL subagent never sees one at inference. An Orchestrator instance never sees the inside of a subagent's ReAct loop, because the Orchestrator never observes that loop at inference either. Mixing these conditioning structures would teach the model to expect information that the architecture refuses to surface, defeating the point of task-driven context filtering.

B.4 Unified policy across roles

A single set of weights θ is trained on the union of all Orchestrator and subagent instances. We do not maintain role-specific weights, role-specific adapters with hard routing, or any architectural

mechanism that gates parameters on role identity. The model conditions on role exclusively through the input: the system prompt `SYS` and, for subagent turns, the dispatch context $c_t^{(k)}$, both of which are visible in x at training time and at inference time.

Role-specific weights would commit the model to a role labeling at inference, which is in tension with the task-driven view in §3.2: the role is a property of the input the agent currently sees, not a property of the model. A unified policy lets the same parameters specialize implicitly through the conditioning distribution induced by the system prompt and dispatch context, with no architectural changes required if a future subagent type (such as the open slot in Figure 2) is added.

C Experiment.

TVSD. At inference time, the trained model is loaded once and used for every dispatch in the hierarchy. The Orchestrator runs the same OneVeriAgent loop used at trajectory collection, dispatching subagents through `Task` actions, and each dispatch hands the subagent the task-scoped context $c_t^{(k)}$ that the model was trained to act on.

We collected 3153 assistant turns and extract 2201 of them after Tool-Validity Filter. We then conduct a 0.9, 0.05, and 0.05 train, eval, and test split. The default gemma4 chat template would hide its reasoning traces so we write a custom template to preserve the thinking traces. We use LoRA r=64, α=128 over MLP + self-attention projections (489M trainable / 31.8B parameters, 1.54%), bf16, FSDP2 with sequence length=32768, micro_batch_size=1. Cosine schedule, lr=2e-5, warmup 5 steps, 2 epochs. Run took $\sim$96 GPU hours on 2 nodes of 4-H100 cluster.

Inference. All Pass@1 evaluations report unbiased Pass@1 over $n = 5$ runs. Opus 4.6 sampling: $T_{\text{eval}} = 1$, top-$p = 0.95$, max output tokens $32k$. Iteration budgets: Orchestrator 20 steps; subagents $10/30/30$ for RTL/DEBUG/RTL_FIX for long horizon CVDP tasks.

Note on VerilogEval-v2 Prob099. VerilogEval-v2's Prob099 has a mismatched testbench and prompt; after communicating with the benchmark authors [6], we apply the corresponding correction.

D Ablation study: context pollution

D.1 Configuration

Single Agent. One agent with the full toolset and no orchestration structure. Every observation enters the same expanding context. **Orchestrator.** The full OneVeriAgent system with task-scoped subagent dispatch (§3.2). Both configurations use the same backbone, same tool definitions, same iteration budgets, and the same $n = 5$ unbiased Pass@1 protocol.

D.2 Per-problem distribution of context size

Figure 3a plots tokens per request against number of requests, one point per problem, for the two configurations. The Single Agent points cluster at the top-left, few requests, large per-request contexts, with several problems trailing upward into the 32k-token regime; these are the problems where the context has accumulated dozens of tool results from prior debugging iterations, and where the agent's accuracy degrades visibly. The Orchestrator points spread to the right (more requests) but stay below 8k tokens per request for almost every problem; the task-scoped projection ϕ_k keeps each subagent invocation bounded regardless of how long the overall trajectory becomes. The two clouds are visually disjoint above 16k tokens per request, where only Single Agent points appear.

D.3 Interpretation

The Single Agent is not slower because it is doing more work; it is making fewer, larger calls because it cannot offload state. The Orchestrator is not faster because it skips work; it is making more, smaller calls because it actively prunes state at every dispatch boundary. Pass-rate convergence on the easier benchmark and divergence on the harder one is consistent with the prediction in §3.2:

context pollution is a property that scales with trajectory length, and a flat agent's accuracy degrades only when problems force long trajectories.

E Per category analysis

We take the liberty to classify the 262 CVDP problems (cid002 + cid003 + cid004 + cid016) into six categories along two orthogonal axes, circuit style (combinational vs. sequential) and functional role (computation, control, storage, or communication):

- **Combinational** purely combinational logic with no clocked state: gate networks, multiplexers, priority/binary/one-hot encoders and decoders, comparators, barrel shifters, bit-reversal, swizzling, popcount, reduction, and Boolean conversions (gray↔binary).
- **Arithmetic / DSP** datapath kernels for numerical computation: integer adders/subtractors (BCD, signed, carry-lookahead, cascaded), multipliers (Booth, complex, matrix), dividers (restoring, Gaussian-rounding, radix-2), GCD, square root, ALU, MAC, dot product, FIR / configurable low-pass filters, moving average, perceptron / Hebbian rule / neuromorphic kernels, BCD conversions, factorial / Fibonacci / arithmetic-progression generators, and sorting circuits.
- **FSM / Control** finite state machines and control-dominated logic: sequence/pattern detectors, vending machines, elevator and parking controllers, traffic / fan / coffee-machine controllers, microcode sequencers, branch predictors, arbiters (fixed, round-robin, bus), watchdog/secure timers, password generators, and pipeline-stall logic.
- **Sequential Datapath (non-FSM)** clocked datapath elements without explicit FSM structure: counters (BCD, precision, perf), shift registers, LFSRs, flip-flops, edge detectors, debouncers, clock/frequency dividers, PWM, signal generators, accumulators, digital stopwatches, CDC synchronizers (bit, pulse, handshake), and Muller C-elements.
- **Memory & Buffers** clocked storage and queue structures: synchronous/asynchronous FIFOs, LIFO/stack, register files, caches and LRU, MSHR, TLBs, SDRAM/instruction-cache controllers, skid / ping-pong / elastic / line / write buffers, memory allocators, and load/store units.
- **Communication & Encoding** interface protocols and data-encoding pipelines: bus protocols (AXI / AXI-Stream, APB, AHB, Wishbone, GMII), serial interfaces (UART / RS-232, SPI, PS/2, IR), line and channel coding (8b10b, 64b66b, Manchester, 16-QAM, DBI, PRBS, scrambling), error-correcting codes (Hamming, CRC, Reed-Solomon, convolutional encoders), cryptographic primitives (Caesar / Hill cipher, Galois encryption, GF / Montgomery multipliers, HMAC), data compression (Huffman, run-length), packet processing (Ethernet parser, packet/interrupt controllers, crossbar switches), serializers / deserializers (PISO, SIPO, width converters), and image / video pipelines (VGA, sprite, image rotate, sound generator).

Combinational and Arithmetic / DSP. Both contain no or few sequential elements. Opus 4.6 saturates Combinational and reaches 95.5% on Arithmetic / DSP, while TVSD gains over Instruct are smaller here (+12.0% and +7.5%) combinational errors typically arise from constant-folding or width mistakes that do not benefit much from a multi-cycle debugging trajectory, so the SFT signal transfers least.

Communication & Encoding is the largest bucket (n=89) and involves industrial protocols where correctness depends on precise spec adherence. This is the category where Opus 4.6 falls furthest from saturation (92.1%), suggesting a difficulty that exceeds what general-purpose pretraining provides. Opus also fails to saturate Arithmetic / DSP (95.5%) and Sequential Datapath (96.2%).

FSM / Control. While Opus saturates this category (100%), TVSD reaches only 73.1%, the largest gap to Opus across all six categories (26.9%). The +15.4% gain over Instruct indicates that the trajectories carry useful FSM debugging patterns, but the residual gap is noticeable.

Memory & Buffers and Sequential Datapath are the two categories where TVSD gains the most over its Instruct baseline (+23.3% and +19.3%). Both involve temporal patterns, FIFO/RAM readwrite semantics, pipeline registers, accumulators, CDC synchronizers, where observing internal state evolve across cycles localizes bugs more efficiently than re-reading the spec. We attribute the strong transfer to the dominance of waveform-tracing actions in our SFT trajectories, which encode exactly this debugging style.

F Extended discussion of limitations

Dependence on backbone model. OneVeriAgent's headline numbers are reported on a strong proprietary model (Opus 4.6). The architectural lift over flat single-pass generation is large, but the absolute level the system reaches still tracks the underlying model. With Opus 4.6, the system reaches 97.7% overall on CVDP. With Gemma4-31B-Instruct in the same loop, the same architecture reaches only 61.1%. Even though we recover to 75.2% with TVSD, the gap to proprietary models remains significant. A stronger open-weight base, or a longer training schedule on a larger trajectory pool, are both expected to raise the ceiling. We have not isolated which of these dominates, and we caution against reading the absolute Gemma4-31B numbers in Table 3 as ceilings on the open-weight version of this architecture.

SFT is a starting point. TVSD is deliberately the most intuitive training procedure that closes the loop without proprietary teachers or learning from verifiable rewards. The interesting open question is not whether RL can be applied to RTL agents, but how to formulate the action space and credit assignment when the reward is delayed across an Orchestrator dispatch, several inner subagent calls, and tool responses. This is the formulation we leave to future work.

Infrastructure assumptions in the agent loop. Two infrastructure assumptions shape what the system can and cannot do today. First, the ATE tool reads from a full VCD waveform on disk. While the agent's observation is sparse and bounded by behavioral complexity, the simulator still writes the dense waveform, and storage scales with simulation duration. For the single-module benchmarks we evaluate, this overhead is invisible. For industrial multi-module designs with longer simulations and larger signal counts, on-demand re-simulation or selective dump generation will likely be necessary. Second, our toolchain uses open-source simulation (`iverilog`, `vvp`). Many real RTL design flows depend on commercial simulators (VCS, Xcelium) and commercial waveform viewers (Verdi). Porting the ATE formulation to commercial backends is still an open problem.

Benchmark assumptions: specifications and the testbench oracle. Both VerilogEval-v2 and RTLLM-v2 provide complete natural-language specifications and corresponding golden testbenches that exactly match those specifications. CVDP is closer to industrial practice but still operates on this assumption: the harness testbench encodes the same intent the natural-language spec describes. Real RTL design rarely starts from a spec this clean. Industrial design organizations maintain rigorously formal specification documents and review them precisely because natural language alone fails to capture corner-case behavior unambiguously. Our system inherits whatever ambiguity the spec contains, and a debug subagent reasoning from spec and waveform alone has no recourse when the spec underdetermines the intended behavior. Mitigating this gap is partly a benchmark design problem and partly a system design problem, and we suspect that future agentic RTL systems will need to integrate spec-clarification dialogues or coverage-aware probing rather than treating the spec as ground truth.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 clearly state the contributions, including the ATE framework, task-scoped orchestration, CVDP benchmark results, and TVSD methodology, which are supported by the experiments in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Section 5, along with Appendix F, explicitly details four limitations regarding proprietary backbone dependence, verifiable reward assignment, dense waveform storage overhead, and benchmark specification assumptions.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper proposes an empirical agentic architecture and training methodology; it does not contain theoretical results or proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 3 provides detailed algorithmic descriptions, and Appendices A, B, and C provide extensive details on subagent specifications, training methodology, and hyperparameters.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Code is not provided in the current submission to preserve double-blind anonymity, but extensive implementation details and prompts are documented in Section 3 and the Appendices. We will open-source our implementation on Github.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 describes the evaluation benchmarks, while Section 3.3 and Appendices B and C detail the training pool, inference configurations, and hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Statistical replication of the TVSD training pipeline would require independent retraining of the 31B-parameter open-weight model for each replicate, at approximately 96 GPU-hours on 8 H100s per replicate. The compute required for k-fold replication of the training stage is prohibitive at this scale. For all inference-only results, Pass@1 is reported as the mean over n=5.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix C explicitly states the compute resources used for the experiments (96 GPU hours on H100 clusters).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research focuses on automated hardware design via language models and complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: The paper focuses strictly on the technical challenges of EDA and RTL design agent frameworks. We do not foresee direct negative societal impacts arising specifically from this tool-use architecture.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk models or sensitive scraped datasets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All existing datasets (CVDP, VerilogEval, RTLLM, etc.) and base models (Gemma4, Claude) are properly credited and cited in the text and References section.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper evaluates models on existing, publicly available benchmarks and does not introduce new datasets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The research does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The research does not involve human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The paper explicitly describes the use of Large Language Models (Claude Opus 4.6, Gemma4-31B-Instruct) as the core agents driving the proposed RTL design framework throughout Sections 3 and 4.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.