

ARTEMIS: A Multi-Agent AI System for Automated Root-Cause Analysis and Optimization of RTL Verification Workflows

Nupur Bhonge¹, Dhara Vaishnav¹, Sameer Farooq¹, Felix David¹

¹Amazon Web Services (AWS), Santa Clara, CA, USA

Correspondence: Nupur Bhonge, nbhonge@amazon.com

WISH 2026 — Track: Artificial Intelligence (AI) in Hardware

Abstract

Debugging failed RTL verification jobs consumes engineering time and compute that could otherwise go to design. This paper presents ARTEMIS (Automated RTL Testing & Error Management Intelligent System), a multi-agent system that closes the loop from a failed simulation job to a verified, human-approved fix that is resubmitted automatically. ARTEMIS combines two large-language-model (LLM) agents, a Supervisor Agent that classifies and routes failures and a Debug Agent that performs root-cause analysis (RCA) and repairs RTL, with deterministic Scheduler and Monitor services that integrate with a SLURM cluster on AWS Parallel Computing Service. The Debug Agent grounds every proposed fix in tool feedback from Verible and Icarus Verilog and iterates until the fix lints and compiles cleanly; an engineer then approves or rejects the fix with a single action. New in this work, approved fixes are traced into source control through a provider-agnostic Git integration, clean runs are preserved as Golden Run baselines, and a closed-loop Job Optimizer right-sizes job memory requests on an LSF cluster. In four end-to-end debug scenarios on two test designs, ARTEMIS repaired all detected errors, including 48 errors across six files of a 32-bit ALU, with AI debug latency of roughly two to four minutes. In validation on a synthetic LSF workload, the Job Optimizer right-sized 44% of submitted jobs. ARTEMIS is in pilot engagements with multiple customers.

1. Introduction

Functional verification of register-transfer-level (RTL) designs runs as large batches of simulation, lint, and compile jobs on shared compute clusters managed by workload managers such as SLURM or LSF. When a job fails, an engineer typically has to notice the failure, locate and read the logs, identify the root cause across several source files, edit the RTL, and resubmit. Two kinds of waste follow. First, engineering time goes to triage rather than design. Second, cluster capacity is consumed by jobs that fail, and by jobs whose memory requests do not match their needs, which affects how many jobs a fixed cluster can run concurrently.

Recent work has shown that LLMs can generate and repair Verilog when evaluated offline on benchmarks [3]–[7]. Using such models inside a production verification flow raises additional requirements: fixes must be verified by real EDA tools, not just proposed; engineers must stay in control of every change to design source; changes must be traceable in source control; and the system must integrate with existing schedulers and shared file systems. ARTEMIS is designed around these requirements.

Purpose. The purpose of this work is to design, implement, and evaluate an agentic system that (i) detects failed verification jobs, (ii) produces tool-verified RTL fixes, (iii) keeps a human approval gate before any fix is re-executed or committed, and (iv) targets wasted cluster resources through closed-loop resource right-sizing.

Contributions. This paper makes four contributions:

- A multi-agent architecture that separates LLM reasoning (Supervisor and Debug agents) from deterministic job control (Scheduler and Monitor services), deployed on managed AWS services (Section 3).
- A tool-grounded repair loop in which the Debug Agent iterates against Verible lint and Icarus Verilog compilation until the fix is clean, followed by human-in-the-loop (HITL) approval and automatic resubmission (Sections 4–5).
- Source-control traceability for AI-generated fixes and Golden Run baselines for clean runs (Section 6).
- A Job Optimizer that right-sizes job memory limits at submission time within a nightly training loop, with validation results on a synthetic LSF workload (Section 7).

The source-control integration, Golden Run baselines, and Job Optimizer (the third and fourth contributions) are new in this paper, together with the Job Optimizer validation results.

2. Background and Related Work

Machine learning for EDA. Machine learning has been applied across the EDA flow, including placement, routing, timing, and verification [1]. Surveys of LLMs for EDA describe applications in RTL generation, script generation, and verification assistance [2].

LLMs for RTL generation and repair. VerilogEval [3] and VeriGen [4] benchmark and fine-tune LLMs for Verilog generation; ChipNeMo [5] adapts LLMs to chip-design tasks; and Chip-Chat [6] and ChipGPT [7] study conversational and automated hardware design. Closest to our debug loop, RTLFixer [8] repairs Verilog syntax errors with an LLM agent that uses compiler feedback and retrieval. These works evaluate generation or repair offline on benchmark problems.

Agent frameworks. ReAct [9] interleaves reasoning with tool actions, AutoGen [10] composes multiple conversational agents, and the Model Context Protocol (MCP) [11] standardizes how agents call external tools. ARTEMIS builds on these ideas using the Strands Agents SDK [12] and Amazon Bedrock AgentCore [13].

HPC resource prediction. Prior work predicts job resource needs from historical accounting data using supervised learning to improve cluster utilization [14].

How ARTEMIS advances the art. Rather than evaluating repair in isolation, ARTEMIS embeds tool-verified repair into the operational verification loop: it watches jobs on a production-style scheduler, repairs failures, requires engineer approval, resubmits automatically, records fixes in source control, and preserves verified baselines. It pairs this debug loop with scheduler-level resource right-sizing in one system, so that both failed jobs and over-provisioned jobs are addressed.

3. System Architecture

ARTEMIS is deployed in a single AWS Region. Engineers submit jobs through an API; requests are queued in Amazon SQS and processed by an AWS Lambda scheduler, which submits them to SLURM on AWS Parallel Computing Service (PCS) [15] through an MCP-based SLURM interface. RTL sources, tools, and logs reside on Amazon FSx for NetApp ONTAP. An Amazon EventBridge rule triggers a monitor function every 60 seconds to poll job state through the SLURM REST API and update a job-state table in Amazon DynamoDB. On failure, the Supervisor Agent classifies the error and routes it to the Debug Agent. Notifications are sent through Amazon SNS and Amazon SES, approvals are received through Amazon API Gateway, and DynamoDB Streams trigger resubmission. Job and RCA history is replicated to Amazon Redshift Serverless through zero-ETL integration for analytics.

ARTEMIS defines four agent roles (Figure 1). The **Supervisor Agent** (LLM) receives failure events, classifies them (for example, lint or syntax errors versus timeouts or license errors), and routes them; it

exposes 12 tools, including job submission, RCA retrieval, approval requests, resubmission, Golden Run storage, and source-control push. The **Debug Agent** (LLM) performs RCA and repair using five tools for project discovery, dual analysis with Verible and Icarus Verilog, file retrieval, saving fixed files, and error-log analysis. The **Scheduler** and **Monitor** roles are implemented as deterministic, event-driven serverless services, which keeps job control predictable and inexpensive and reserves LLM reasoning for diagnosis and repair. Both LLM agents run on Amazon Bedrock AgentCore and are built with the Strands Agents SDK; the experiments in this paper used Anthropic Claude Sonnet 4.5 on Amazon Bedrock.

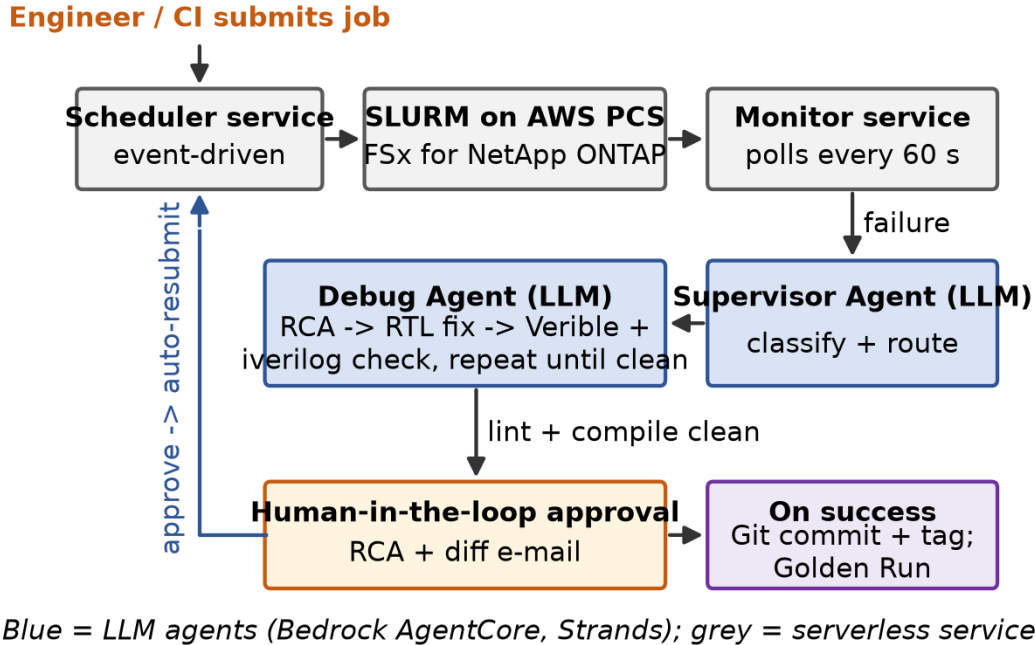


Figure 1. Agent roles and the self-healing loop. A single human action (approve) separates a failed job from its automatic resubmission.

4. Tool-Grounded Debug and Repair Loop

When the Supervisor routes a failure, a debug orchestrator invokes the Debug Agent with the job context. The agent discovers the project's source files, runs Verible [16] (lint) and Icarus Verilog [17] (compile) on them, retrieves the files that contain errors, and generates fixes. Each time it saves fixed files, both tools are re-run and any remaining errors are returned to the agent, which repeats until no errors remain. The orchestrator then independently verifies the result: it checks that every file with errors was addressed and compiles all fixed files with Icarus Verilog. Missing files or compile errors are fed back to the agent for up to three attempts.

Fixes are written to a separate fixed/ directory on the shared file system rather than overwriting the original sources, and are also stored as a structured record in Amazon S3. An approval request is sent **only if** fixes exist and the compile check has passed; otherwise the job is marked as blocked and the owner is notified. Grounding every fix in real tool output, and never re-running unverified output, limits the impact of incorrect LLM suggestions.

5. Human-in-the-Loop Governance

Engineers remain in control of every change. Each approver receives an e-mail containing the job details, the issues found by category, a unified diff of the proposed changes per file, and the location of the fixed files (Figure 2). A personalized approve or reject link opens a confirmation page before the decision is

recorded. Approval sets the job's approval status in DynamoDB; the resulting stream event resubmits the job from the fixed files, copies the approver list to the new job, and links the original and resubmitted jobs so that the full chain is traceable. Rejected fixes are never executed.

ARTEMIS DEBUG REPORT - Job 645 (alu32bit_v3): 6 fixes ready

Issues identified: syntax errors (7 instances);
missing port connections detected and fixed

Changes made (diff) - alu_system.sv

```
@@ -15,14 +15,14 @@
-    // ERROR: Missing comma after .func(func)
+    // Fixed: Added comma after .func(func)
-    .func(func)
+    .func(func),
-    // ERROR: Missing port connection for flag_negative
+    // Fixed: Added flag_negative port connection
@@ -32,8 +32,8 @@
-    .flag_carry(flag_carry)
-    // Missing: .flag_negative(flag_negative)
+    .flag_carry(flag_carry),
+    .flag_negative(flag_negative)
```

Fixed files: /projects/ALU32bit_v3/645/fixed/

Action required: [APPROVE AND RESUBMIT] [REJECT]

Figure 2. Content of the RCA approval e-mail for job 645 (Scenario D), transcribed from the e-mail: issue summary, proposed diff for alu_system.sv, fixed-file location, and approval actions. Tokenized approval links are omitted.

6. Source-Control Traceability and Golden Runs

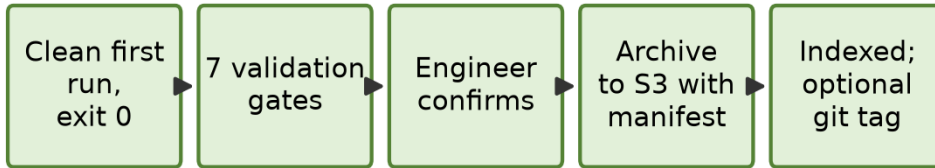
Source-control integration. To make AI-generated fixes auditable, ARTEMIS can record an approved fix in the project's Git repository once the resubmitted job has completed successfully (Figure 3a). The push is triggered explicitly by the engineer and is guarded by a conditional write in DynamoDB, so it happens at most once per job. The fix is committed on a dedicated branch named artemis-fix/{project}/{job} under a bot identity, and an annotated tag is created; the branch, commit hash, and tag are written back to the job record, linking each simulation result to the exact code change. Projects without a configured repository are skipped gracefully. The integration is provider-agnostic: a Git provider interface is paired with pluggable SSH-key, token, and IAM-based authentication strategies.

Golden Runs. A clean first-attempt run is a useful reference for later comparisons. When a job completes, the monitor evaluates seven validation gates, including successful completion, exit code 0, no RCA, and not being a resubmission. Jobs that pass become Golden Run candidates (Figure 3b). On engineer confirmation, logs, RTL sources, and a manifest are archived to Amazon S3, the run is indexed in DynamoDB through a secondary index for retrieval, and an optional Git tag is created. The golden-run validation, archival, and deployment-packaging logic is covered by 68 automated unit tests, all of which passed when run on 30 September 2026. ARTEMIS is packaged as a parameterized AWS CloudFormation template with six nested stacks and least-privilege IAM roles for its functions.

(a) AI-repaired run -> source control



(b) Clean first-attempt run -> Golden Run baseline



Provider-agnostic Git interface: SSH-key, token and IAM authentication

Figure 3. (a) Source-control traceability for AI-repaired runs. (b) Golden Run baseline creation for clean first-attempt runs.

7. Job Optimizer: Closed-Loop Resource Right-Sizing

The ARTEMIS Job Optimizer targets memory requests that do not match what jobs need. It runs as a continuous learning loop on an LSF cluster (Figure 4): submission-time right-sizing, accounting collection, nightly training, and refreshed recommendations. Right-sizing can reduce a memory limit or raise it. The optimizer uses cohorts; by design, jobs in new cohorts and jobs that have opted out are left unchanged, and the submission path supports fail-open operation.

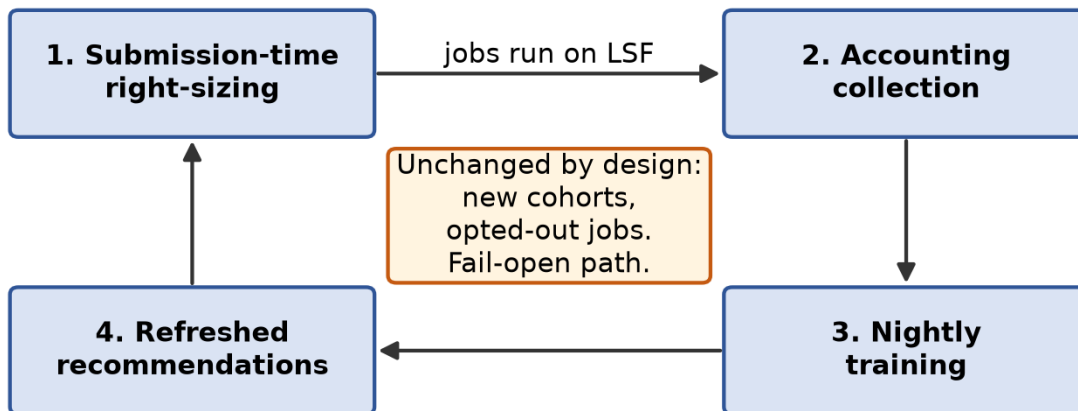


Figure 4. Job Optimizer learning loop.

The optimizer was validated on a synthetic workload on an LSF cluster in an AWS account (Table 1). The learning loop ran continuously for one week with no pipeline failures. Further evaluation is ongoing.

Table 1. Job Optimizer validation on a synthetic LSF workload.

Metric	Result
Jobs processed	~35,000
Jobs right-sized (as reported)	15,500 (44%)
Memory limits reduced / raised	14,550 / 1,049
Jobs unchanged by design	Remainder (new cohorts, opt-outs)
Continuous operation	1 week; no pipeline failures

8. Experimental Setup

Table 2 summarizes the experimental conditions for the debug experiments. They used open-source EDA tools and small test designs with intentionally injected errors. To emulate a long-running regression, the ALU32bit_v3 job script inserts a 30-minute delay before compilation, so reported runtimes for that scenario include this delay.

Table 2. Experimental conditions for the debug experiments.

Item	Debug experiments
Environment	AWS account, single Region; SLURM on AWS PCS; FSx for NetApp ONTAP
AI models	Anthropic Claude Sonnet 4.5 on Amazon Bedrock (Supervisor and Debug agents)
EDA tools	Verible (lint), Icarus Verilog -g2012 (compile and simulate)
Workload	FIFO8 (2 files); ALU32bit and ALU32bit_v3 (6 SystemVerilog files each: package, adder, ALU core, control, system, testbench)
Injected faults	Missing semicolons and commas, missing port connections, missing end or endmodule, undeclared signals, typos, style violations
Period	February to March 2026

9. Results: End-to-End Debug and Repair

Table 3 lists four end-to-end scenarios. In each, the job failed, ARTEMIS produced tool-verified fixes, the fix was approved, and the resubmitted job completed successfully. The largest case, ALU32bit, contained 48 detected errors across six files (15 reported by Verible and 33 by Icarus Verilog); all were repaired and the design then compiled and simulated.

Table 3. End-to-end debug scenarios (n/r = not recorded).

Scenario	Design	Jobs (failed → resubmitted)	Errors detected	Files fixed	Outcome of resubmitted job
A	FIFO8	434 → 435	3 (2 trailing whitespace, 1 line length)	2	Passed; fixed in 1 iteration
B	FIFO8	532 → 533	Missing semicolons (count n/r)	n/r	Passed (FIFO test complete, exit code 0)
C	ALU32bit	535 → 536	48 (15 Verible, 33 Icarus Verilog)	6	Compiled and simulated
D	ALU32bit_v3	645 → 646	7 syntax errors and missing port connections; 6 fixes proposed	n/r	Completed in 1,801 s (includes 30-min emulated delay)

Table 4 breaks down the latency of Scenario A, and Figure 5 shows the timeline of Scenario D. In Scenario D, the RCA and six proposed fixes were e-mailed about two minutes after the failure was

detected. The approval was granted roughly 12 minutes later, and resubmission followed within 2 seconds. ARTEMIS recorded the two jobs as one resubmission chain: two attempts, one success, final status completed. The time between failure and approval was dominated by the human decision rather than by AI processing.

Table 4. Latency breakdown, Scenario A (jobs 434 → 435).

Step	Measured time
API response to submission (asynchronous via SQS)	200 ms
Submission to SLURM	2 s
Failure detection (60-second monitoring interval)	≤ 60 s
RCA and fix generation (Debug Agent)	240 s
Human approval	295 s
Resubmission after approval	2 s
Iterations to a clean fix	1

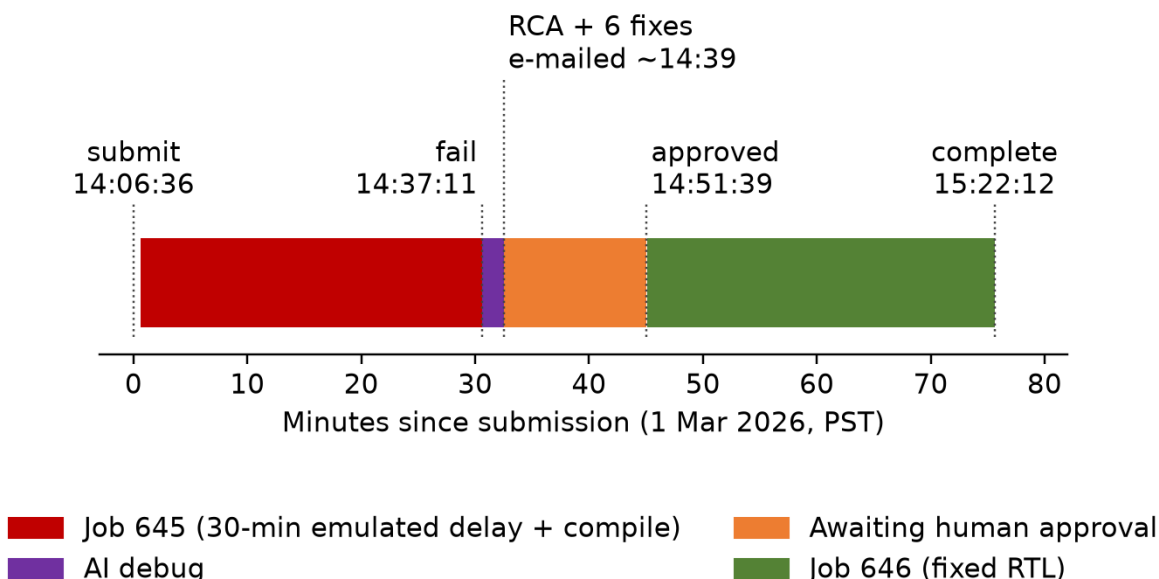


Figure 5. Timeline of Scenario D (job 645 → 646). Job 645 includes a 30-minute emulated delay before compilation.

Figure 6 shows how the Debug Agent classified 71 issues across the 50 most recent failed jobs, as retrieved from the analytics warehouse. Syntax errors (18), style and whitespace violations (15), and issues the agent classified as logic errors (14) were most common. These categories are the agent's own classifications and were not independently audited.

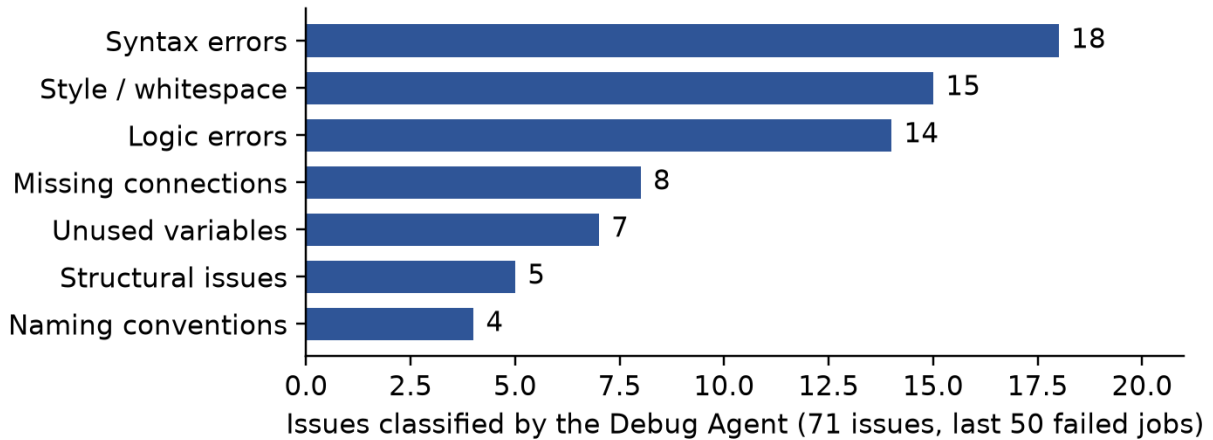


Figure 6. Distribution of issue types classified by the Debug Agent across the last 50 failed jobs.

10. Discussion and Limitations

- **Optimizer scope.** The optimizer results cover a synthetic workload; the training method and resulting resource savings are not reported in this paper.
- **Workload realism.** The debug experiments used small test designs with injected errors that were mostly syntactic or structural. Evaluation on production designs and real regression suites is future work.
- **Tooling.** The debug loop has been validated with open-source tools (Verible and Icarus Verilog). Support for commercial simulators requires tool-specific adapters for their error formats.
- **LLM reliability.** LLMs can propose incorrect fixes for unfamiliar errors. ARTEMIS mitigates this by requiring clean lint and compile results before any approval request and by keeping a human approval gate, but tool-clean code is not proof of functional correctness.
- **Baselines.** This paper reports absolute latencies and counts. A controlled comparison against manual debugging time is future work.
- **Source control.** The Git integration is implemented. Automated pull or merge request creation and end-to-end validation against customer repositories are next steps.
- **Throughput.** In the measured scenarios, turnaround in the HITL flow was dominated by human response time. ARTEMIS supports additional approvers per job.

11. Industry Pilots and Next Steps

ARTEMIS is in pilot engagements with multiple customers. Next steps include further evaluation of the Job Optimizer, extending the debug loop to commercial simulators and to waveform and log analysis for functional failures, creating pull or merge requests automatically, and measuring debug-time savings against engineers' baselines in customer environments.

12. Conclusion

ARTEMIS shows that LLM agents can be embedded safely in an RTL verification flow when every fix is grounded in EDA tool feedback, gated by an engineer, and traceable in source control. In four end-to-end scenarios it repaired all detected errors, including 48 errors across a six-file ALU design, with AI debug latency of two to four minutes and a single human action between failure and resubmission. Combined with a Job Optimizer that right-sizes job memory limits in a closed loop, ARTEMIS addresses both failed jobs and over-provisioned jobs, two major sources of wasted verification capacity.

References

- [1] G. Huang et al., "Machine learning for electronic design automation: A survey," *ACM Trans. Design Automation of Electronic Systems*, vol. 26, no. 5, 2021, doi: 10.1145/3451179.
- [2] R. Zhong et al., "LLM4EDA: Emerging progress in large language models for electronic design automation," *arXiv:2401.12224*, 2023.
- [3] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: Evaluating large language models for Verilog code generation," in *Proc. IEEE/ACM ICCAD*, 2023, doi: 10.1109/ICCAD57390.2023.10323812.
- [4] S. Thakur et al., "VeriGen: A large language model for Verilog code generation," *ACM Trans. Design Automation of Electronic Systems*, vol. 29, no. 3, 2024, doi: 10.1145/3643681.
- [5] M. Liu et al., "ChipNeMo: Domain-adapted LLMs for chip design," *arXiv:2311.00176*, 2023.
- [6] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-Chat: Challenges and opportunities in conversational hardware design," in *Proc. ACM/IEEE MLCAD*, 2023, doi: 10.1109/MLCAD58807.2023.10299874.
- [7] K. Chang et al., "ChipGPT: How far are we from natural language hardware design," *arXiv:2305.14019*, 2023.
- [8] Y. Tsai, M. Liu, and H. Ren, "RTLFixer: Automatically fixing RTL syntax errors with large language model," in *Proc. 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, doi: 10.1145/3649329.3657353.
- [9] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *Proc. ICLR*, 2023, *arXiv:2210.03629*.
- [10] Q. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," *arXiv:2308.08155*, 2023.
- [11] Anthropic, "Introducing the Model Context Protocol," 2024. [Online]. Available: <https://www.anthropic.com/news/model-context-protocol>
- [12] Strands Agents SDK. [Online]. Available: <https://strandsagents.com>
- [13] Amazon Web Services, "Amazon Bedrock AgentCore Developer Guide." [Online]. Available: <https://docs.aws.amazon.com/bedrock-agentcore/>
- [14] M. Tanash et al., "Improving HPC system performance by predicting job resources via supervised machine learning," in *Proc. PEARC*, 2019, doi: 10.1145/3332186.3333041.
- [15] Amazon Web Services, "AWS Parallel Computing Service User Guide." [Online]. Available: <https://docs.aws.amazon.com/pcs/>
- [16] CHIPS Alliance, "Verible: A suite of SystemVerilog developer tools." [Online]. Available: <https://github.com/chipsalliance/verible>
- [17] S. Williams, "Icarus Verilog documentation." [Online]. Available: <https://steveicarus.github.io/iverilog/>