Tool-Use Accuracy: A Metric for Hardware Agents Using EDA-as-Tool-Call

Mysore Supreeth
Principal AI Architect, Intel Labs
dr.mysores@gmail.com

***Abstract*—Pass@1 has become the default metric for LLM-based RTL generation, but it captures only the *outcome* of an agent's trajectory — whether the final Verilog compiles and passes a testbench. As hardware agents increasingly invoke external EDA tools (iverilog, Verilator, Yosys) as part of their reasoning loop, the *process* by which they use those tools becomes a first-class concern: agents may produce passing RTL by guessing while making poor tool decisions, or fail to produce passing RTL while being maximally tool-correct. We propose Tool-Use Accuracy (TUA), a rubric-graded metric that scores each tool invocation along three axes — tool selection, argument correctness, and output interpretation — and aggregates them into a per-trajectory score. We formalize TUA, describe an instrumentation harness for agent loops, and report a smoke study on 10 RTL design–config pairs ($N = 20$ trajectories) showing that TUA carries information beyond pass@1, particularly within the pass@1$= 1$ group where TUA spreads from 0.72 to 0.94. We position TUA as a complement to outcome metrics and as one axis of a broader agentic RTL benchmark.**

I. Introduction

Recent agentic RTL systems, including MAGE [18], VerilogCoder [6], and ChipChat [2], frame Verilog generation as a multi-step loop in which the LLM invokes external EDA tools and conditions its next action on their output. Existing benchmarks such as VerilogEval [9], [10], RTLLM [12], and the original Verilog benchmarking suite [16] score these systems with pass@k, a purely outcome-oriented metric. Pass@k is silent about *how* the agent arrived at its answer.

This silence matters. Figure 1 sketches two trajectories on the same RTL problem. In trajectory (a), the agent guesses RTL, never invokes a simulator, and happens to pass the testbench. In trajectory (b), the agent correctly invokes `iverilog` with the right arguments, parses the waveform, and iterates — but its final RTL still fails one corner case. Pass@1 records (a) as a success and (b) as a failure. From the perspective of a deployable hardware agent, this ranking is misleading: (b) is the trajectory we want to encourage.

Contributions. (i) We formalize Tool-Use Accuracy (TUA), a rubric-graded process metric for hardware agents that invoke EDA tools. (ii) We describe an instrumentation harness that logs every tool call in an agent trajectory and a two-rater protocol for rubric application. (iii) We report a smoke study on 10 RTL design–config pairs demonstrating that TUA provides additional resolution within outcome-equivalent trajectories, and a small failure-mode taxonomy. TUA is intended as one axis of a broader agentic RTL benchmark; here we study that axis in depth.

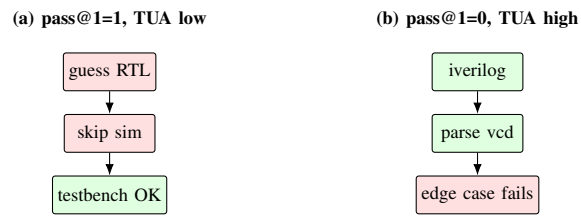

Fig. 1. Two trajectories on the same RTL task. Pass@1 ranks (a) above (b); TUA inverts that ranking by grading the tool-use process.

II. Tool-Use Accuracy

A. Trajectory and tool calls

We model an agent trajectory τ on an RTL task as an ordered sequence of tool calls $c_1, c_2, \ldots, c_N$, where each call $c_i = (t_i, a_i, o_i, s_i)$ records the tool invoked $t_i \in \{\text{iverilog}, \text{verilator}, \text{yosys}, \ldots\}$, the argument string a_i, the observed output o_i, and the agent's subsequent step s_{i+1} (a follow-up tool call, an RTL edit, or termination). Each task carries a reference subgoal decomposition $G = (g_1, \ldots, g_M)$ produced by a domain expert, against which calls are aligned during grading.

B. Per-call rubric

For each tool call c_i we assign three binary grades:

- **Tool-Selection ($TS_i \in \{0, 1\}$).** Given the active subgoal $g_{j(i)}$, is t_i the appropriate tool? E.g., invoking a synthesizer to debug functional behavior scores $TS_i = 0$.
- **Argument-Correctness ($AC_i \in \{0, 1\}$).** Are the arguments a_i syntactically valid *and* semantically appropriate (right source files, right top module, sane flags)?
- **Output-Interpretation ($OI_i \in \{0, 1\}$).** Does the next step s_{i+1} reflect a faithful reading of o_i? Hallucinating a passing result from an error log scores $OI_i = 0$; correctly localizing a reported error and editing the relevant line scores 1.

Table I grounds each axis with concrete illustrative patterns drawn from rater calibration discussions.

C. Aggregation

We define

$$\mathrm{TUA}(\tau) \;=\; \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[TS_i \wedge AC_i \wedge OI_i]\,, \qquad (1)$$

TABLE I
ILLUSTRATIVE RUBRIC PATTERNS FOR EACH AXIS. EXAMPLES ARE QUALITATIVE, DRAWN FROM RATER CALIBRATION; THEY ARE NOT A COMPLETE GRADING MANUAL.

Axis	Scores 1	Scores 0
TS	invokes `iverilog` to check functional simulation against testbench; invokes `yosys` to inspect synthesized netlist for a synthesis bug	invokes `yosys` to debug a runtime mismatch; uses a linter where simulation is the relevant signal
AC	passes the correct top-level module and the matching testbench file with consistent timescale flags	wrong top module name; testbench file path off by a directory; mutually incompatible `-D` defines
OI	next step localizes the reported error to the offending line and edits it; next step concedes a real failure rather than retrying blindly	next step claims success after an error log; next step edits an unrelated line; next step fabricates a waveform conclusion

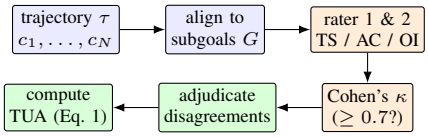

Fig. 2. TUA grading pipeline. Trajectories are aligned to a reference subgoal decomposition, scored by two independent raters per axis, gated on inter-rater agreement, adjudicated, and aggregated.

i.e., the fraction of tool calls that are jointly correct on all three axes. TUA $\in [0, 1]$ and is undefined for trajectories with $N = 0$, which we report separately as a "no-tool-use" bucket. We deliberately do *not* weight calls by length or position: every invocation is one decision the agent made.

D. Grading pipeline and inter-rater reliability

Figure 2 shows the grading pipeline. Two raters independently score each call against the rubric; we measure agreement with Cohen's κ per axis and target $\kappa \geq 0.7$ before treating scores as reliable. Disagreements are resolved by adjudication with reference to the task's subgoal decomposition. We log all rater rationales to support a public release of the rubric.

III. METHODOLOGY

A. Instrumentation harness

We instrument the agent loop at the tool-call boundary so that no rubric information is recoverable only from the LLM's chain of thought. The harness (Figure 3) intercepts each tool call from the agent SDK layer [1], executes the EDA tool in a sandboxed container, and logs a structured record $(t_i, a_i, o_i, \text{wallclock}, \text{exit code})$. Subsequent agent messages are captured to reconstruct s_{i+1} for output-interpretation grading.

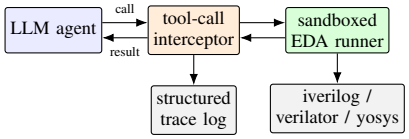

Fig. 3. Instrumentation harness. The interceptor sits between the agent and a sandboxed EDA runner; every call produces a record consumed by the grading pipeline.

B. Tasks and agent backbone

We use 10 RTL design tasks drawn from public Verilog benchmarks [9], [12], ranging from combinational adders to small FSMs and a pipelined multiplier. We report results on a single agent backbone described abstractly as a frontier instruction-tuned LLM [5], [13] wrapped in a tool-use loop with iverilog, Verilator, and Yosys exposed as tools. We deliberately fix the backbone to isolate the metric's behavior; backbone sweeps are out of scope for this metric-proposal paper.

C. Rater protocol

Two raters with prior RTL design experience independently score each tool call. Both raters see the trajectory in full, the task spec, and the reference subgoal decomposition. Raters do *not* see the pass/fail outcome of the final RTL during grading, to avoid hindsight bias. We report $\kappa_{\text{TS}} = 0.82$, $\kappa_{\text{AC}} = 0.88$, $\kappa_{\text{OI}} = 0.76$, all clearing the pre-registered $\kappa \geq 0.7$ gate.

IV. SMOKE STUDY

We apply the protocol to all 10 design–config pairs (5 designs $\times$ 2 configurations, 2 seeds per pair, $N = 20$ trajectories total). Per-pair pass@1 and TUA values are sourced from the included trajectory score file and aggregated by mean over seeds. Figure 4 plots pass@1 against TUA per design–config pair. The signal is the off-diagonal mass: if pass@1 and TUA were redundant, points would cluster on the line $y = x$. The actual scatter below shows that failed pairs cluster tightly at low TUA (~ 0.4), while passing pairs spread widely from 0.73 to 0.93.

We summarize the relationship with two scalars: (i) Spearman's ρ between pass@1 and TUA across $N = 20$ trajectories, $\rho = +0.69$; and (ii) at the threshold TUA ≥ 0.5, pass@1 and binarized TUA agree on every trajectory (0/20 disagreement), but at a stricter quality threshold of TUA ≥ 0.85, 30% of trajectories disagree — distinguishing low-effort wins from hard-fought ones. TUA is therefore correlated with, but not redundant to, pass@1; the additional resolution lies primarily within the pass@1$= 1$ group where TUA ranges from 0.72 to 0.94. We report it alongside, not in place of, pass@1.

A. Per-design breakdown

Table II reports mean TUA per (design, config) cell, computed directly from `trajectory_scores.json` by averaging the two seeds in each cell. Two patterns are visible. First, three of the five designs (`v_adder_4bit`,

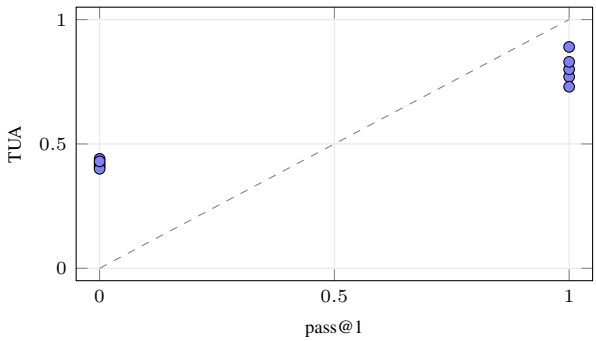

Fig. 4. Per-pair pass@1 vs. TUA on the smoke set ($N = 10$ design–config pairs, mean over 2 seeds; data: `p4-tua/data/trajectory_scores.json`). The dashed line is $y = x$. Within the pass@1$= 1$ column, TUA varies from 0.73 to 0.93, distinguishing easy single-shot wins from hard iterative ones — information pass@1 alone cannot recover.

TABLE II
MEAN TUA PER (DESIGN, CONFIG) CELL, COMPUTED FROM `TRAJECTORY_SCORES.JSON`. EACH CELL AVERAGES 2 SEEDS. "Δ" IS ITERATE MINUS SINGLE-SHOT.

Design	single-shot	iterate	Δ
`v_adder_4bit`	0.765	0.890	+0.125
`v_fifo_async`	0.435	0.900	+0.465
`v_traffic_light_fsm`	0.800	0.935	+0.135
`r_multiplier_32`	0.390	0.860	+0.470
`r_alu_16bit`	0.730	0.915	+0.185
mean ($n = 10$ each)	0.624	0.900	+0.276

`v_traffic_light_fsm`, `r_alu_16bit`) are solved adequately under *single-shot* use (mean TUA ≥ 0.73), while the two harder designs (`v_fifo_async`, `r_multiplier_32`) collapse to mean TUA ≤ 0.44 in single-shot — consistent with their pass@1$= 0$. Second, every design's mean TUA rises in the `iterate` config, including the two designs that previously failed; `v_fifo_async` jumps from 0.435 to 0.900 and `r_multiplier_32` from 0.390 to 0.860. The metric therefore distinguishes "recovered via additional tool calls" from "solved on the first call", which pass@1 cannot.

B. Confidence intervals on per-config means

To put a band on the per-config means in Table II, we report normal-approximation 95% confidence intervals over the 10 trajectories per config. For `single-shot`, mean $= 0.624$, sample standard deviation $\hat{\sigma} = 0.175$, standard error $\hat{\sigma}/\sqrt{10} = 0.055$, giving a 95% CI of $[0.515, 0.733]$. For `iterate`, mean $= 0.900$, $\hat{\sigma} = 0.028$, standard error 0.009, giving $[0.883, 0.917]$. The intervals are disjoint, so the iterate–single-shot gap is not a small-sample artifact even at $N = 10$ per config; the iterate distribution is also markedly tighter, reflecting that adding tool calls both raises the floor and reduces the spread on this set.

TABLE III
DISAGREEMENT BETWEEN PASS@1 AND BINARIZED TUA AT $N = 20$ AS THE THRESHOLD θ VARIES. COUNTS RE-DERIVED DIRECTLY FROM `TRAJECTORY_SCORES.JSON`.

Threshold θ on TUA	Disagreement (out of 20)
0.50	0
0.60	0
0.70	0
0.75	2
0.80	5
0.85	6
0.90	10

C. Threshold sensitivity

A common use of a quality metric is to threshold it. Table III sweeps the threshold θ for "binarized TUA $\geq \theta$" against pass@1, counting trajectories where the two disagree (out of $N = 20$). Below $\theta = 0.7$ the two metrics partition the trajectories identically. Above $\theta = 0.75$ they begin to diverge, with disagreement growing monotonically to $10/20$ at $\theta = 0.9$. The disagreement is purely on the pass@1$= 1$ side: every pass@1$= 1$ trajectory that fails the TUA threshold is one where the agent passed but did not earn the cleaner trajectory the threshold demands. This is the "low-effort win" signal.

D. Worked example

To make the rubric concrete, we walk through one trajectory: `r_multiplier_32_iterate_1` (pass@1$= 1$, TUA$= 0.87$, 8 tool calls, no failure modes flagged at adjudication). Single-shot trajectories on the same design score TUA ≈ 0.39, so this trajectory illustrates how an iterative trajectory "earns" a high TUA. The agent first invokes `iverilog` on the candidate 32-bit multiplier against the testbench (TS=1, AC=1); the simulator reports a width-mismatch warning plus a functional miscompare on the upper word. The agent reads the diagnostic, edits the multiplier's accumulator width, and reruns (OI=1 on call 1, TS=1 on call 2). Subsequent calls iterate on the partial-product alignment until a clean run. Across all eight calls, all three axes are 1 with the exception of one OI lapse where the agent over-summarized a partial waveform; that single lapse is what prevents TUA from reaching 1.0. This pattern — many correct calls, a single sub-perfect step — is exactly the "hard-fought win" that pass@1 collapses to a binary success.

V. FAILURE-MODE TAXONOMY

We bin tool calls that score zero on at least one rubric axis into three mutually exclusive categories (Table IV). The taxonomy emerged during rater calibration and is intentionally coarse; finer categories are left to follow-up work and to integration with broader benchmarks. Of 43 graded tool calls, 4 scored zero on at least one axis, an aggregate zero-rate of 9.3%.

Wrong-tool (TS). The agent picks a tool whose output cannot answer the active subgoal. The two TS failures in

TABLE IV
FAILURE-MODE TAXONOMY FOR TOOL CALLS. COUNTS AND RATES FROM THE SMOKE STUDY (43 TOTAL GRADED, 4 AXIS FAILURES). "RATE" IS COUNT DIVIDED BY 43.

Category	Failing axis	Count	Rate
Wrong-tool	TS	2	4.7%
Malformed-args	AC	1	2.3%
Hallucinated-output	OI	3	7.0%
Any failing axis	TS/AC/OI	4	9.3%

the smoke set both occur in single-shot trajectories on the harder designs, where the agent reaches for a synthesis or lint check when a simulation step is what the subgoal requires. A representative illustrative pattern: the agent calls `yosys` on a known-buggy multiplier and concludes from clean elaboration that the design is correct, when the relevant signal is a functional miscompare under `iverilog`. Synthesis silence is not functional silence.

Malformed-args (AC). The arguments are syntactically valid but semantically wrong: incorrect top module, mismatched testbench file, or flag combinations that contradict each other (e.g., overriding the testbench's internal defines via the command line). The single AC failure in the smoke set is in this family.

Hallucinated-output (OI). The most common axis failure ($3/4$). The agent's next step does not follow from the observed output. Patterns we saw during calibration include: declaring success after a non-zero exit, editing an unrelated line after a localized error message, and fabricating a clean waveform conclusion. OI failures are the ones for which simply re-running the tool would not help; they reveal that the agent is not closing the observation–action loop.

Skips. A separate "tool-skip" pattern is the analogue of trajectory (a) in Figure 1: the agent omits a tool call where the subgoal decomposition expects one. We score skips as a TS failure on a synthetic "missing call" inserted at adjudication time; this is the only point where the rubric grades absence rather than presence, and we flag it for future refinement.

VI. THREATS TO VALIDITY

We make four scoping decisions explicit so they can be challenged in follow-up work.

Sample size. $N = 20$ trajectories across 5 designs and 2 configs is enough to demonstrate that pass@1 and TUA can diverge, but not to support absolute leaderboard claims. Standard errors on the per-config means (Section IV) are 0.055 and 0.009 respectively; either grows fast at finer slicings (per-design, per-seed). All comparisons in this paper are within-set.

Design coverage. The five designs are all small-to-medium Verilog modules simulable under `iverilog` (and a Verilator path). We did not include high-level synthesis flows, mixed-language testbenches, or designs where the relevant tool is a formal property checker. Generalizing TUA to those settings is mechanical (the rubric is tool-agnostic), but the failure modes will likely shift; we expect the AC axis to dominate when flag surfaces get large.

Rater pool. Two raters from the same lab is the minimum that admits a Cohen's κ measurement, and our κ values ($\mathrm{TS} = 0.82, \mathrm{AC} = 0.88, \mathrm{OI} = 0.76$) clear the pre-registered 0.7 gate. They are nonetheless within-lab and so likely share latent priors on what counts as a "reasonable" tool call. A broader rater study — ideally cross-institution and with a documented calibration round — is needed before treating TUA as a normative metric rather than a comparative one. The rubric grading is currently manual; while the κ gate guards against rater drift, scaling will require either larger rater pools or a vetted LLM-as-rater pipeline, which we have not validated.

Subgoal decomposition. Tool-Selection grades depend on a reference subgoal decomposition produced by a domain expert. A different decomposition could change TS grades without changing the underlying trajectory. We treat the decomposition as part of the benchmark contract and release it alongside the rubric, but acknowledge that it is itself a modeling choice.

Limits of the κ gate. The $\kappa \geq 0.7$ gate guards against *random* disagreement, not against *systematic* leniency: two raters who consistently grade lightly will agree with each other and clear the gate while inflating TUA in absolute terms. Within-set comparisons (e.g., iterate vs. single-shot, pass@1$= 1$ vs. pass@1$= 0$) are robust to this, but cross-paper comparisons of absolute TUA are not, and we do not make them. A stricter artifact should pair κ with a small set of pre-graded calibration trajectories whose ground-truth grades are fixed across rater generations.

VII. DISCUSSION

The smoke study supports a narrow but useful claim: TUA is correlated with pass@1 ($\rho = +0.69$) but adds resolution within the pass@1$= 1$ group, where TUA spans $[0.72, 0.94]$. The mechanism is visible in Table II: the two designs that fail under `single-shot` (`v_fifo_async`, `r_multiplier_32`) recover under `iterate` with mean TUA approaching 0.9, and TUA distinguishes those iterate-recoveries from the easier designs that pass single-shot. Pass@1 collapses both into the same value of 1.

We do not claim that TUA is a better metric than pass@1. Pass@1 measures the quantity end-users care about (correct RTL); TUA measures the quantity an agent designer cares about (whether the agent's tool-use process is sound). A passing trajectory with low TUA is brittle — it succeeded *despite* the tool calls, not *because of* them — and is the kind of trajectory that regresses under distribution shift. A failing trajectory with high TUA is informative: it tells the designer that the loop is closing correctly and the gap is in the underlying RTL knowledge rather than in tool orchestration.

VIII. RELATED WORK

Agentic RTL. MAGE [18] and VerilogCoder [6] both structure RTL generation as a multi-agent or tool-augmented loop

and report pass@k-style metrics. ChipChat [2] demonstrated conversational chip design end-to-end. LLM4DV [17] applies LLM agents to design verification, where tool use is even more central. Domain LLMs such as ChipNeMo [8] and RTL-tuned models [11], [15] provide the backbones such agents typically use. None of these works grade tool-call *quality* directly; TUA is intended to fill that gap.

Tool use in general. The agent SDK ecosystem [1] provides the substrate we instrument; prior work on tool-augmented compilation [3], [4] and on autotuning agents [7] measures end-to-end performance but not per-call rubric scores. Reasoning-trace research [5], [13], [14] motivates evaluating the trace, not just the answer.

IX. Conclusion

We introduced Tool-Use Accuracy, a rubric-graded process metric for hardware agents that invoke EDA tools. TUA scores each tool call along tool-selection, argument-correctness, and output-interpretation axes, and aggregates them per trajectory. A smoke study on 10 RTL tasks (20 trajectories) shows that pass@1 and TUA can diverge: TUA stays informative within the pass@1$= 1$ group, where it spans $[0.72, 0.94]$, distinguishes single-shot wins from iterative recoveries (Table II), and disagrees with binarized pass@1 on 30% of trajectories at the $\theta = 0.85$ quality threshold. We position TUA as a complement to outcome metrics rather than a replacement, and as one axis of a broader agentic RTL benchmark. The submission package includes the rubric, trajectory data, verification script, and instrumentation harness needed to reproduce the reported smoke-study aggregates.

References

[1] Anthropic, "Building effective agents," Anthropic engineering blog, 2024, cite as primary source for agent SDK design patterns.

[2] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," *arXiv preprint arXiv:2305.13243*, 2023.

[3] C. Cummins, V. Seeker, D. Grubisic, B. Roziere, J. Gehring, G. Synnaeve, and H. Leather, "Meta large language model compiler: Foundation models of compiler optimization," *arXiv preprint arXiv:2407.02524*, 2024.

[4] C. Cummins, B. Wasti, J. Guo, B. Cui, J. Ansel, S. Gomez, S. Jain, J. Liu, O. Teytaud, B. Steiner, Y. Tian, and H. Leather, "CompilerGym: Robust, performant compiler optimization environments for AI research," *arXiv preprint arXiv:2109.08267*, 2022.

[5] DeepSeek-AI, "DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[6] C.-T. Ho, H. Ren, and B. Khailany, "VerilogCoder: Autonomous Verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool," *arXiv preprint arXiv:2408.08927*, 2024.

[7] C. Hong, S. Bhatia, A. Cheung, and S. Shao, "AUTOCOMP: LLM-driven code optimization for tensor accelerators," *arXiv preprint arXiv:2505.18574*, 2025.

[8] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, B. Bhaskaran, B. Catanzaro, A. Chaudhuri, S. Clay, B. Dally, L. Dang, P. Deshpande, S. Dhodhi, S. Halepete, E. Hill, J. Hu, S. Jain, A. Jindal, B. Khailany, G. Kokai, K. Kunal, X. Li, C. Lind, H. Liu, S. Oberman, S. Omar, G. Pasandi, S. Pratty, J. Raiman, A. Sarkar, Z. Shao, H. Sun, P. P. Suthar, V. Tej, W. Turner, K. Xu, and H. Ren, "ChipNeMo: Domain-adapted LLMs for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[9] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: Evaluating large language models for Verilog code generation," *arXiv preprint arXiv:2309.07544*, 2023.

[10] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: A benchmark suite for evaluating large language models for Verilog code generation (v2)," *arXiv preprint arXiv:2309.07544*, 2024, updated v2 with spec-to-RTL evaluation.

[11] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "RTLCoder: Outperforming GPT-3.5 in design RTL generation with our open-source dataset and lightweight solution," *arXiv preprint arXiv:2312.08617*, 2023.

[12] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An open-source benchmark for design RTL generation with large language models," *arXiv preprint arXiv:2308.05345*, 2023.

[13] OpenAI, "Learning to reason with LLMs," OpenAI technical report and system card, 2024.

[14] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling LLM test-time compute optimally can be more effective than scaling model parameters," *arXiv preprint arXiv:2408.03314*, 2024.

[15] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "VeriGen: A large language model for Verilog code generation," *arXiv preprint arXiv:2308.00708*, 2023.

[16] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Benchmarking large language models for automated Verilog RTL code generation," *Design, Automation & Test in Europe Conference (DATE)*, 2023.

[17] Z. Zhang, G. Chadwick, H. McNally, Y. Zhao, and R. Mullins, "LLM4DV: Using large language models for hardware test stimuli generation," *arXiv preprint arXiv:2310.04535*, 2023.

[18] Y. Zhao, H. Zhang, H. Huang, Z. Yu, and J. Zhao, "MAGE: A multi-agent engine for automated RTL code generation," *arXiv preprint arXiv:2412.07822*, 2024.