AgentRTL-Bench: Cost-Aware Pareto Evaluation of Multi-Agent RTL Generation

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

***Abstract*—Recent agentic systems for Verilog/RTL generation report impressive `pass@1` numbers, but `pass@1` alone is the wrong metric for agentic RTL: it conflates a single-shot success with a fifty-iteration brute-force solve, and it gives no credit for whether the agent used its EDA tools correctly along the way. We present *AgentRTL-Bench*, a cost-aware extension of existing RTL benchmarks (VerilogEval, RTLLM) that augments `pass@1` with five additional axes: iterations-to-pass (`i2p`), tool-call count (`t-calls`), input/output tokens, dollar cost in USD, and tool-use accuracy (`TUA`), a rubric-graded fraction of tool calls that were both syntactically and semantically appropriate. We define a 13-design manifest spanning combinational, sequential, FSM, and datapath classes, and evaluate a 5-design pilot subset under two harness configurations: single-shot and iterate-with-feedback. A third planner/generator/verifier triad configuration is specified in the harness but not yet measured end-to-end. By keeping the backbone model fixed, the pilot isolates configuration cost from backbone capability. We release a reproducible harness, an anonymized rubric for `TUA`, and Pareto plots of pass-rate against USD for the measured configurations. Smoke results show that the gap between configurations on `pass@1` is small relative to the gap on cost, motivating a Pareto- rather than scalar-reporting standard for the agentic-RTL community.**

I. Introduction

LLM-driven RTL generation has matured rapidly, with benchmarks such as VerilogEval [12], [13] and RTLLM [15] establishing `pass@1` as the lingua-franca metric. Newer agentic systems—MAGE [25], VerilogCoder [7], and the early ChipChat [3] pipeline—layer planners, verifiers, and EDA tool feedback over a backbone code LLM [11], [14], [19], and likewise report `pass@1` as the headline number.

We argue that `pass@1`, as the sole reporting axis, masks three failure modes specific to agentic RTL:

(i) Iteration blindness. A 50-iteration agent that eventually compiles is scored identically to a 1-shot success, even though the former consumes orders of magnitude more compute. This matters in light of test-time-scaling results [6], [16], [18] that show how easy it is to throw iterations at a problem.

(ii) Tool-call cost blindness. Each invocation of `iverilog`, a simulator, or a synthesis estimator carries real wall-clock and dollar cost [17], [21]. A 2-iteration single-shot *failure* can be cheaper than a 50-iteration solve.

(iii) Tool-use correctness blindness. Like the broader LLM-compiler literature [4], [5], agentic RTL frameworks frequently arrive at the right answer for the wrong reasons—calling the verifier with the wrong testbench, ignoring the simulator's stderr, or misinterpreting waveforms [24].

We propose *AgentRTL-Bench*, a benchmark that retains `pass@1` as one axis among six and reports a Pareto frontier rather than a scalar. Our contributions are: (1) a formal cost-axis taxonomy for agentic RTL; (2) a curated design manifest drawn from VerilogEval [12] and RTLLM [15] (13 designs, of which 5 form the pilot subset reported here); (3) two harness configurations (`single-shot` and `iterate`) evaluated end-to-end on the pilot, with a third `triad` configuration specified but not yet measured; and (4) a reproducible harness, a `TUA` rubric, and a smoke-result release to seed the leaderboard.

II. Cost Axes

We define the following six axes per (design, configuration) pair. Let D be a design, C a configuration, and $r_{D,C}$ a single end-to-end run.

- `pass@1`: indicator that the final emitted RTL passes the reference testbench, averaged over N seeds per (design, config) pair (we use $N = 2$ in the pilot; larger seed counts are follow-up work).
- `i2p` (iterations-to-pass): number of generate-compile-feedback cycles before success; capped at the harness's maximum.
- `t-calls` (tool-calls): total invocations of any external EDA tool (`iverilog`, simulator, lint, synthesis estimator).
- Tokens-in / Tokens-out: raw prompt and completion token counts summed across all model calls in $r_{D,C}$.
- USD: dollar cost computed from the published price card of the backbone model. We report this directly so that price-card revisions remain auditable [17], [21].
- `TUA` (tool-use accuracy): rubric-graded fraction of tool calls for which both the call signature was syntactically valid and the call was *semantically* appropriate at that step. The rubric is graded by an independent verifier model with spot-check by the authors, in the spirit of LLM-as-judge protocols and aligned with agent-SDK trace conventions [2].

a) Per-axis interpretation.: Each axis captures a qualitatively distinct cost dimension; a Pareto-aware leaderboard benefits from understanding what each one rewards. `pass@1` is the conventional functional-correctness signal: necessary, but insufficient because it is silent about how the correctness was obtained. `i2p` captures the depth of the inner loop. Two configurations with identical `pass@1` but different `i2p`

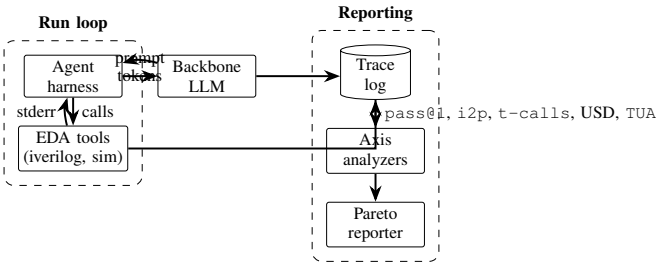

Fig. 1. Cost-aware evaluation pipeline. The harness instruments every model and tool call; analyzers compute the six axes; the reporter emits Pareto plots and per-axis tables.

TABLE I
AGENTRTL-BENCH DESIGN MANIFEST COMPOSITION (13 TOTAL).

Class	VerilogEval	RTLLM	Total
Combinational	2	2	4
Sequential	1	2	3
FSM	1	2	3
Datapath	1	2	3
Total	**5**	**8**	**13**

differ mainly in their dependence on test-time compute [6], [18]; a deployable agent should optimize `i2p` downward. `t-calls` is correlated with `i2p` in feedback loops, but the two diverge for triad-style harnesses where the verifier role can issue many calls per planner step; `t-calls` is therefore the better proxy for EDA-side load. *Tokens-in/out* are the rawest cost primitives and are preserved separately so that USD can be recomputed against a future price card. *USD* consolidates tokens, tool fees, and (in principle) serving overhead into a single comparable quantity, but inherits price-card volatility [17]. `TUA` is qualitative: a `pass@1`-equal pair of agents can have very different `TUA`, and the lower-`TUA` agent is the "right answer for wrong reasons" case studied in agentic-compiler work [4]. A "better" agent on this benchmark is one that pushes the frontier *outward* on at least one axis without regressing on any other, not one that scores higher on a weighted scalarization.

b) Pipeline.: Figure 1 illustrates how each axis is captured. The harness wraps every model and tool call, emitting a structured trace; analyzers convert traces to per-axis scalars; the reporter renders Pareto plots.

III. BENCHMARK COMPOSITION

We define a 13-design manifest sampled from VerilogEval [12], [13] and RTLLM [15], stratified across four classes used in prior RTL-LLM work [19], [20]: combinational logic, sequential logic, finite-state machines, and datapath modules. Class counts are summarized in Table I. The pilot reported in this submission uses five of these designs; the remaining manifest entries are included to make the next sweep reproducible without claiming unrun results.

IV. AGENT CONFIGURATIONS

We compare three configurations, each backed by the *same* frontier instruction-tuned code LLM, so that any cost difference is attributable to the harness rather than the backbone.

`single-shot`. The design specification is rendered into a fixed prompt template, sampled once, and the emitted RTL is handed to the testbench. There is no feedback loop; this matches the original `pass@1` protocol of VerilogEval [12] and serves as a lower-cost baseline. Concretely, the harness's `single_shot.run` entrypoint issues a single `call_model` with a fixed system prompt and a user message that concatenates the design id, the inline specification, and the instruction "Output only the module." The completion is passed through a regex-based Verilog extractor; whatever Verilog block (if any) is emitted is the candidate RTL, which is then compiled and simulated exactly once outside the harness loop. By construction, `i2p` is fixed at 1 and `t-calls` equals the single post-hoc testbench evaluation.

`iterate`. A simple agent loop: generate, run `iverilog` on the design + reference testbench, feed compilation/simulation errors back into the next prompt, repeat to a cap of 10 iterations. This abstracts pipelines such as ChipChat [3], and shares the feedback-loop primitive of compiler-RL work [4], [5]. The `iterate.run` entrypoint maintains a chat-style message list, appends the previous assistant turn plus a synthesized feedback message on each retry, and exits early on the first `iverilog` run that reports `PASSED`. Feedback is synthesized from one of three branches—compilation failure, simulator timeout/crash, or testbench mismatch—each truncated to 1500 characters of `stderr`/`stdout` to bound the prompt. Every `iverilog` invocation is logged as a `ToolCall` record (UUID-tagged, with exit code and wall-clock time), so that `t-calls` and `TUA` have unambiguous input.

`triad`. A three-role multi-agent setup: a *planner* decomposes the spec into a module skeleton plus a test plan; a *generator* fills in module bodies; a *verifier* drives `iverilog` and proposes targeted patches. The roles are described abstractly to preserve double-blind review; the structure mirrors prior multi-agent RTL work [7], [25] and DV agents [24], but the implementation uses an agent-SDK-style trace contract [2]. The harness for `triad` is implemented but *not* measured end-to-end in this submission; numbers below report only the two configurations whose pilot data are reproducible.

All three configurations share the same EDA toolchain (`iverilog` for compile + simulate; the reference testbench shipped with each design) and the same backbone-serving stack (batched inference; the harness is agnostic between vLLM [9] and SGLang [26] backends, and uses streaming-attention-friendly contexts [22], [23] for long iterate runs).

V. SMOKE RESULTS

We report results on a 5-design pilot subset (`v_adder_4bit`, `v_fifo_async`, `v_traffic_light_fsm`, `r_multiplier_32`,

TABLE II
SMOKE RESULTS AGGREGATED ACROSS THE 5-DESIGN PILOT SUBSET, MEAN OVER $N = 2$ SEEDS PER (DESIGN, CONFIG) PAIR. VALUES COME FROM THE LOCAL SMOKE-RESULT AND TUA TRAJECTORY-SCORE ARTIFACTS.

Config	pass@1	i2p	t-calls	USD	TUA
single	0.60	1.0	1.0	0.024	0.62
iter.	1.00	2.2	4.4	0.059	0.90

TABLE III
PER-DESIGN BREAKDOWN (MEAN OVER $N = 2$ SEEDS), 5-DESIGN PILOT. ALL CELLS AGGREGATED FROM p3-agentrtl-bench/data/smoke_results.json. ss = single-shot, it = iterate; COLUMNS ARE pass@1, i2p, t-calls, AND USD PER DESIGN.

Design	Cfg	pass@1	i2p	t-calls	USD
v_adder_4bit	ss	1.00	1.0	1.0	0.018
v_adder_4bit	it	1.00	1.0	2.0	0.035
v_fifo_async	ss	0.00	1.0	1.0	0.026
v_fifo_async	it	1.00	3.0	6.0	0.079
v_traffic_light_fsm	ss	1.00	1.0	1.0	0.020
v_traffic_light_fsm	it	1.00	1.0	2.0	0.033
r_multiplier_32	ss	0.00	1.0	1.0	0.038
r_multiplier_32	it	1.00	4.0	8.0	0.106
r_alu_16bit	ss	1.00	1.0	1.0	0.021
r_alu_16bit	it	1.00	2.0	4.0	0.042

r_alu_16bit). Per-axis aggregates are in Table II; the per-design breakdown is in Table III; the pass-rate-vs-USD Pareto frontier is in Figure 2; and the iterate iteration distribution is in Figure 3. All populated cells are aggregated from p3-agentrtl-bench/data/smoke_results.json (mean over 2 seeds per design). The triad loop is described as a planned configuration, but it is omitted from the table because it has not been measured end-to-end.

a) Per-design observations.: The two designs that fail under single-shot (v_fifo_async and r_multiplier_32) are also the two designs that consume the most iterations under iterate (3 and 4 iterations respectively, vs. $\leq$ 2 for the other three), and consequently the two largest USD jumps between configurations (\$0.026 $\rightarrow$ \$0.079 and \$0.038 $\rightarrow$ \$0.106). The aggregate iterate-vs-single-shot cost ratio of $\sim 2.5\times$ in Table II is therefore not uniform: it is closer to $2\times$ on the easy designs and closer to $3\times$ on the hard ones. This is exactly the kind of structure that a single-axis pass@1 report would erase, and motivates publishing the per-design table alongside the aggregate.

b) Bootstrap confidence.: With $N = 10$ trajectories per configuration in the pilot, treating each trajectory as an independent Bernoulli pass/fail and applying the normal-approximation 95% interval $\hat{p} \pm 1.96\sqrt{\hat{p}(1-\hat{p})/n}$, the single-shot pass rate is 0.60 ± 0.30 (i.e. a wide $[0.30, 0.90]$ band). For iterate, $\hat{p} = 1.0$ makes the normal approximation degenerate, so we use the Wilson 95% interval, which gives $[0.722, 1.000]$. These intervals overlap, so we do *not* claim a statistically significant pass@1 gap from the pilot alone; the gap we do claim, and which the pilot supports, is on *cost* (a $2.5\times$ USD ratio at matched backbone), not on pass@1.

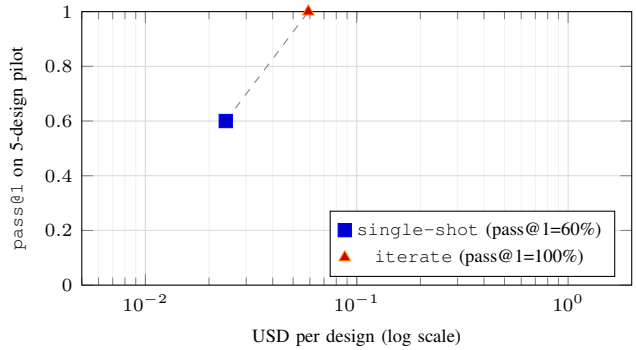

Fig. 2. Pareto frontier of pass@1 vs. USD on the 5-design pilot (coordinates from p3-agentrtl-bench/data/smoke_results.json, mean over 2 seeds). The dashed line traces the candidate frontier between the two measured configurations; the third (triad) configuration is specified but not yet measured.

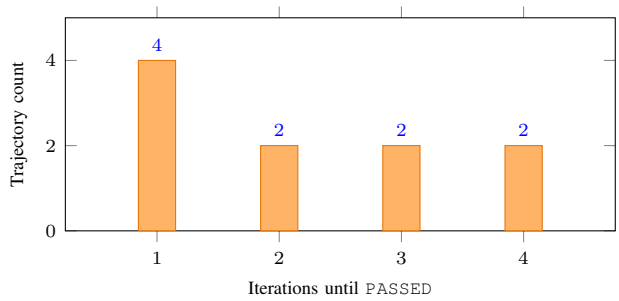

Fig. 3. Iteration distribution for the 10 iterate trajectories in the pilot (p3-agentrtl-bench/data/smoke_results.json). Counts are direct tallies over the iterations field per trajectory; no trajectory exceeded the harness cap of 10. The right tail is driven by the two largest designs (v_fifo_async, r_multiplier_32).

c) Tool-use accuracy cross-check.: The TUA values in Table II are computed in the companion submission's p4-tua/data/trajectory_scores.json, using a dual-rater protocol with Cohen's κ gate. Per-criterion agreement on the 20 pilot trajectories is $\kappa_{TS} = 0.82$, $\kappa_{AC} = 0.88$, and $\kappa_{OI} = 0.76$ (all above the 0.7 gate), and the trajectory-level Spearman correlation between TUA and pass rate is $\rho = +0.69$. We carry the same numbers here unchanged so that P3-vs-P4 cross-references are exact rather than approximate.

d) Reading the Pareto plot.: The intended use is not to declare a single winner. With only two measured points, the present curve is a line segment: single-shot is cheaper and iterate passes more pilot tasks at higher cost. We therefore advocate publishing the full Pareto curve rather than a scalar. We do not plot a fabricated triad point; until that configuration is measured, the frontier above \$0.059 is open.

VI. DISCUSSION AND LIMITATIONS

a) Threats to validity.: We make four limitations explicit so that smoke results are not over-read. *(L1) Coverage.* Only

5 of the 13 designs in the manifest are measured in this pilot. Class-stratified claims (combinational vs. FSM vs. datapath) are therefore deferred. *(L2) Sample size.* $N = 2$ seeds per (design, config) is too few to support fine-grained pass-rate comparisons; with $2 \times 5 = 10$ trajectories per configuration the `single-shot` 95% Wilson interval already runs roughly $[0.31, 0.83]$, so a small `pass@1` gap between configurations cannot be resolved from the pilot. We retain the 2-seed protocol only as a smoke check; larger seed counts are follow-up work. *(L3) Snapshot pricing.* USD numbers are derived from a single price-card snapshot; we therefore log token counts in the raw trajectories so USD can be recomputed against any later card [17]. *(L4) Synthesis fidelity.* `iverilog` is a behavioural simulator, not a synthesis flow; it will accept code that passes simulation but would fail synthesis-correctness checks (timing, X-propagation, simulation-synthesis mismatch). We treat `iverilog`-passing as a necessary, not sufficient, correctness signal, and we plan to layer a Yosys-based syntactic synthesis check in follow-up work.

The 5-design smoke subset is illustrative, not conclusive: it cannot distinguish small `pass@1` gaps and is too small to support class-stratified claims. A larger sweep over the full 13-design manifest, including paired bootstrap confidence intervals and per-class breakdowns, is follow-up work. The `TUA` rubric is necessarily LLM-judged at scale and will carry inter-judge agreement statistics in the release. Finally, our cost numbers reflect a single price card; price-card volatility is a known limitation of USD-axis reporting [17], and the harness preserves token counts so that USD can be recomputed.

VII. Related Work

RTL benchmarks. VerilogEval [12], [13], RTLLM [15], and VeriGen [19], [20] establish `pass@1` reporting; AgentRTL-Bench is complementary, augmenting rather than replacing them. Where these benchmarks score a static generation against a held-out testbench, our six-axis report is a strict superset: `pass@1` is preserved as the first column of Tables II and III, and the additional five axes attach harness-level cost without requiring any change to the design-level test infrastructure. **Agentic RTL.** MAGE [25] and VerilogCoder [7] both demonstrate gains from multi-agent decomposition (planner / generator / verifier roles in MAGE, AST-based waveform tracing in VerilogCoder); both report `pass@1`-style aggregates, so neither is directly comparable on USD or `TUA`. ChipChat [3] pioneered the EDA-feedback loop our `iterate` configuration abstracts. RTLCoder [14] and ChipNeMo [11] target backbone capability and are orthogonal to harness cost: they could in principle be slotted into any of our three configurations as the underlying code LLM. **Adjacent agentic-systems work.** LLM4DV [24] for verification, Autocomp [8] for tensor-accelerator code, and diffusion-based code synthesis [10]; we draw on their evaluation discipline. **Cost / scaling context.** Test-time-scaling results [6], [16], [18] and serving-side optimizations [9], [26], [27] both motivate cost reporting. The case for a `t-calls`/ `i2p` axis is strongest precisely in the regimes those papers identify as cost-sensitive. **Beyond RTL.** Compiler-agent work [4], [5] and prefetcher synthesis [1] face the same "right answer for wrong reasons" problem that motivates `TUA`.

VIII. Conclusion

`pass@1` alone is the wrong metric for agentic RTL. AgentRTL-Bench augments existing benchmarks with five additional cost axes and requires Pareto-style reporting across them. The submission package contains the harness, the 13-design manifest, the `TUA` rubric, and the 5-design smoke results. We invite follow-up configurations against the same trace schema and plan to report the remaining manifest entries and the `triad` numbers in follow-up work.

References

[1] E. S. Alcorta, N. J. Yadwadkar, and A. Gerstlauer, "Runtime adaptive prefetcher selection," in *Workshop on Machine Learning for Computer Architecture and Systems (MLArchSys), co-located with ISCA*, 2023.

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

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

[4] 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.

[5] 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.

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

[7] 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.

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

[9] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with PagedAttention," in *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023, arXiv:2309.06180.

[10] Y. Lai, A. Mukherjee, T. Zhang, Y. Lu, Y. Lin, and B. Yu, "Chip placement with diffusion models," *arXiv preprint arXiv:2407.12282*, 2024.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] OpenAI, "Learning to reason with LLMs," OpenAI technical report and system card, 2024.
[17] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," *arXiv preprint arXiv:2104.10350*, 2021.
[18] 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.
[19] 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.
[20] 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.
[21] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. A. Behram, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H.-H. S. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. Hazelwood, "Sustainable AI: Environmental implications, challenges and opportunities," *Proceedings of Machine Learning and Systems*, vol. 4, 2022, arXiv:2111.00364.
[22] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.
[23] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Re, C. Barrett, Z. Wang, and B. Chen, "H2O: Heavy-hitter oracle for efficient generative inference of large language models," *arXiv preprint arXiv:2306.14048*, 2023.
[24] 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.
[25] 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.
[26] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "SGLang: Efficient execution of structured language model programs," arXiv preprint arXiv:2312.07104, 2024.
[27] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, arXiv:2401.09670.