Automated Multi-Agent Workflows for RTL Design

Amulya Bhattaram[*], **Janani Ramamoorthy**[*], **Ranit Gupta**,
Diana Marculescu, Dimitrios Stamoulis[†]
Chandra Family Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX, USA 78712
{abhattaram, janani.ram, ranitgupta, dianam, dstamoulis}@utexas.edu

Abstract

The rise of agentic AI workflows unlocks novel opportunities for computer systems design and optimization. However, for specialized domains such as program synthesis, the relative scarcity of HDL and proprietary EDA resources online compared to more common programming tasks introduces challenges, often necessitating task-specific fine-tuning, high inference costs, and manually-crafted agent orchestration. In this work, we present *VeriMaAS*, a *multi-agent* framework designed to automatically compose agentic workflows for RTL code generation. Our key insight is to integrate formal verification feedback from HDL tools directly into workflow generation, reducing the cost of gradient-based updates or prolonged reasoning traces. Our method improves synthesis performance by 5–7% for pass@k over fine-tuned baselines, while requiring only a few hundred "training" examples, representing an order-of-magnitude reduction in supervision cost.

1 Introduction

Agentic AI presents exciting opportunities for system optimization and design [12, 5, 28, 18, 20, 30]. For program synthesis and hardware designs, numerous methods and benchmarks have emerged for register-transfer level (RTL) code generation [22, 13, 19, 15, 30, 23], electronic design automation (EDA) tool scripting [27], accelerator design [6], hardware design language (HDL) error debugging [9, 24], and post-synthesis metric estimation [1]. However, to mature into expert-level "HDL Copilots," agentic solutions must navigate disparate EDA workflows, nuanced design trade-offs, and complex synthesis pipelines [23, 30].

Recent approaches fine-tune large language models (LLM) on curated RTL/HDL benchmarks as pairs of natural language prompts (RTL design questions) and their corresponding hardware implementations [22, 13, 19, 30, 23]. Though these methods demonstrate strong performance [13, 30], they rely on costly fine-tuning [19], which requires substantial GPU budgets and might generalizes poorly to adjacent HDL tasks [1]. On the other hand, frontier Large Reasoning Models (LRMs), such as OpenAI's o4 [11], achieve robust results on RTL coding benchmarks [30] without fine-tuning, but shift the computational burden from training to inference [21].

In this work, we draw inspiration from the novel paradigm of automated multi-agent workflow generation [31, 17, 32, 10]. Recent methods, such as MaAS [31], Flow [17], and AFlow [32], improve task performance-cost trade-offs compared to monolithic LLM prompting with strong generalizability. However, as discussed in [31], these methods primarily focus on question-answering (QA) with wiki-trivia and math quizzes and coding tasks. Consequently, a gap might persist between "general knowledge" domains, where simple prompting operators such as Debate [4] yield robust performance on multiple-choice QA queries, versus domain-specific RTL design problems [30].

[*]Equal contribution; alphabetical order. [†]Corresponding author.

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025).

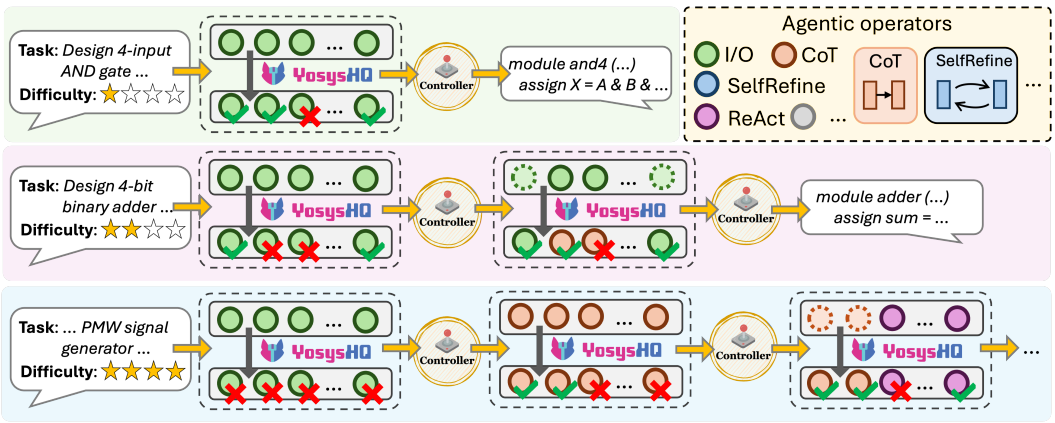

Figure 1: *VeriMaAS*: Given RTL tasks with varying difficulty, we adaptively sample agentic operators: at each step, the selected operators are evaluated against formal verification and synthesis EDA tools. The workflow controller receives synthesis logs to dynamically refine operator selection.

Our **key insight** is to integrate HDL verification checks directly into the workflow generation process: we dynamically provide agentic reasoning with design logs and error messages from RTL/EDA synthesis tools to guide workflow creation. This simple yet effective solution enables us to experiment with agentic reasoning across various design objectives, such as optimizing for post-synthesis goals through PPA (power, performance, and area)–aware prompting. Across state-of-the-art benchmarks [30, 13], our approach achieves accuracy on par or higher compared to prior methods by up to 7% *pass@k*, while requiring only a few hundred "training" examples for controller tuning, representing an **order-of-magnitude** less supervision than full fine-tuning.

2 Methodology

Figure 1 illustrates our approach. Given an RTL design task, *VeriMaAS* adaptively samples a set of reasoning operators based on the input query and task difficulty. In each state, the candidate Verilog designs are executed via a synthesis and verification pipeline using Yosys [26] and OpenSTA [2], and the resulting log-error messages are used as feedback to inform subsequent decision steps to refine the reasoning strategy or terminate the generation of code and return the current pool of samples.

Solution Space. We define solution space $\mathbb{O}$ as the set of available agentic operators: Zero-shot I/O, Chain-of-Thought (CoT) [25], ReAct [29], Self-Refine [16], and Debate [4]. We denote the multi-agent solution in $\mathbb{O}$ as $\mathcal{O} = \{O_i \in \mathbb{O} \mid i = 1, \ldots, K\}$. In fact, most existing prompting schemes can be viewed as single-solution operator sequences. For example, always using zero-shot CoT prompting corresponds to $\mathcal{O} = \{O_{\text{CoT}}\}$. Similarly, a Self-Refine setup that corrects a previous solution can be represented as a two-step sequence $\mathcal{O} = \{O_{\text{CoT}}, O_{\text{SelfRefine}}\}$. Given a user query, our goal is to find a composition of operators $\mathcal{O}$ that yields top RTL designs. For code generation, performance is typically assessed using *pass@k* metrics [3] over a fixed set of model outputs. As *pass@k* is typically computed over $K = 20$ samples, we set $|\mathcal{O}| = K = 20$.

Agentic Controller. The core of multi-agentic workflow automation lies in the *controller* module $\mathcal{C}$ which dynamically selects the prompting operators for each task: $\mathcal{C}$ takes as input the query q, the operators $\mathbb{O}$, and the answers $\mathcal{A}_{\mathcal{O}_{current}}$ from the current solution $\mathcal{O}_{current}$, and outputs an updated solution $\mathcal{O}_{new}$. We employ a *cascading controller* [31] that selects increasingly complex operators at each stage, following $\{O_{\text{I/O}} \rightarrow O_{\text{CoT}} \rightarrow O_{\text{ReAct}} \rightarrow O_{\text{SelfRefine}} \rightarrow O_{\text{Debate}}\}$. At each stage c, the controller $\mathcal{C}$ computes a confidence score s_c indicating whether to proceed to the next operator in the cascade. If s_c falls below a threshold τ_c, the controller returns the current solution vector $\mathcal{A}_{\mathcal{O}_c}$; otherwise, it adds a new operator O_{c+1} to the possible solution set and continues. We denote the set of per-stage thresholds as $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_C\}$.

In our implementation, we leverage the fact that formal verification and synthesis provide a strong signal of design complexity. Intuitively, if the majority of candidate solutions (*e.g.*, initially generated using I/O) fail to compile or pass Yosys checks [26], this suggests that the task may require more sophisticated reasoning operators. At each cascade stage, we run $K = 20$ Verilog candidates $\mathcal{A}_{\mathcal{O}_c}$

Table 1: *Pass@k vs.* Instruct LLMs, Reasoning LRMs, and fine-tuned RTL-coding models [14, 30].

Model	Method	VeriThoughts [30]		VerilogEval [19]	
		Pass@1	*Pass@10*	*Pass@1*	*Pass@10*
GPT 4o-mini	Instruct	80.64	90.87	50.26	61.02
GPT 4o-mini+**VeriMaAS**	Agent workflow	83.09 ↑2.45	92.85 ↑1.98	52.05 ↑1.79	64.02 ↑3.00
o4-mini	Reasoning	93.85	97.88	75.67	85.13
o4-mini+**VeriMaAS**	Agent workflow	94.09 ↑0.24	98.17 ↑0.29	76.15 ↑0.48	84.50 ↓0.63
Qwen2.5-7B	Instruct	44.90	82.33	22.92	51.47
RTLCoder-7B [14]	RTL Fine-tuned	–	–	34.60 ↑11.68	45.50 ↓5.97
Qwen2.5-7B+**VeriMaAS**	Agent workflow	56.62 ↑11.72	86.29 ↑3.96	29.10 ↑6.18	56.45 ↑4.98
Qwen2.5-14B	Instruct	67.89	94.13	33.78	62.04
VeriThoughts-14B [30]	RTL Fine-tuned	78.50 ↑10.61	92.10 ↓2.03	43.70 ↑9.92	55.14 ↓6.90
Qwen2.5-14B+**VeriMaAS**	Agent workflow	74.24 ↑6.35	95.78 ↑1.65	41.47 ↑7.69	62.48 ↑0.44
Qwen3-8B	Reasoning	84.11	98.82	58.21	74.64
RTLCoder-DeepSeek-7B [14]	RTL Fine-tuned	–	–	39.70	51.90
Qwen3-8B+**VeriMaAS**	Agent workflow	88.13 ↑4.02	99.05 ↑0.23	59.87 ↑1.66	74.18 ↓0.46
DeepSeek-R1-Qwen-14B [8]	Reasoning	46.20	89.10	38.70	69.00
Qwen3-14B	Reasoning	89.35	98.64	65.87	75.62
Qwen3-14B+**VeriMaAS**	Agent workflow	92.16 ↑2.81	98.75 ↑0.11	66.96 ↑1.09	75.71 ↑0.09

through Yosys for synthesis and area estimation, and through OpenSTA [2] for timing and power analysis. We compute $s_c = \texttt{Score}(\mathcal{A}_{\mathcal{O}_c})$ as the percentage of failing designs due to errors in verification, synthesis (area), runtime, or power analysis. If this percentage exceeds the stage-specific threshold $\tau_c \in [0, 1]$, the controller proceeds to the next cascade stage.

Problem formulation. The controller $\mathcal{C}$ is parameterized by the per-stage thresholds $\mathcal{T} = \{\tau_1, \ldots, \tau_C\}$. Let D denote a dataset consisting of queries q and their corresponding oracle Verilog solutions a. Our objective is to learn the thresholds $\mathcal{T}$ that generate high-quality solutions with minimal token cost [31]. Formulated as a multi-objective optimization problem, we write [31]:

$$\max_{\mathcal{T}} \mathbb{E}_{(q,a)\sim D} \left[U(\mathcal{T}; q, a, \mathbb{O}) - \lambda \cdot C(\mathcal{T}; q, a, \mathbb{O}) \right] \tag{1}$$

Here, $U(\cdot)$ and $C(\cdot)$ denote the utility and cost of solving the subset of query-answer pairs D under a threshold configuration $\mathcal{T}$ and operator set $\mathbb{O}$. We compute $U(\cdot)$ and $C(\cdot)$ as the *pass@k* score and the average number of tokens per query, respectively. We set $\lambda = 1e^{-3}$ as in [31].

Following [31], we randomly sample 500 datapoints from the VeriThoughts [30] *training* set. For each datapoint, we compute *pass@k* and token cost against the ground-truth design. Based on synthesis results for $K = 20$ candidates per query, we count how many fail to pass Yosys and OpenSTA checks. To determine thresholds $\mathcal{T}$, we aggregate failure counts across all samples and compute the 20th, 40th, 60th, and 80th percentiles (*i.e.*, corresponding to our five operators in $\mathbb{O}$). We note that this "tuning" procedure requires a few hundred datapoints, representing an order-of-magnitude reduction in training cost compared to the tens of thousands of samples needed for full fine-tuning [30].

3 Results

Experimental Setup. We evaluate performance on two state-of-the-art benchmarks, VerilogEval [13] and VeriThoughts [30], and report *pass@1* and *pass@10* scores over 20 samples following the VeriThoughts repo [30]. We use Yosys [26] for verification and area estimation, and OpenSTA [2] for timing and static power analysis. All designs are synthesized using the Skywater 130nm PDK [7], following the MetRex synthesis benchmark [1]. We evaluate proprietary and open-weight models, covering Instruct and Reasoning baselines, as well as existing fine-tuned RTL approaches [30, 9].

Main Results. Across both VeriThoughts and VerilogEval, VeriMaAS improves synthesis accuracy over strong single-agent prompting strategies and fine-tuned RTL models. On open-source LLMs, our framework yields up to 7–12% gains in pass@1 compared to existing fine-tuned baselines. *VeriMaAS* also consistently improves *pass@10* across both benchmarks on top of base LLMs, indicating that the framework not only raises top-1 accuracy but also expands the pool of valid candidate designs.

Table 2: *Pass@k* scores and token cost *vs.* single-agent solvers. We report the relative change *vs.* the Table 1 baselines. We **bold** the best results and underline the runner-ups.

Model	Prompting	VeriThoughts [30]			VerilogEval [19]		
		Pass@1	*Pass@10*	*Tokens (k)*	*Pass@1*	*Pass@10*	*Tokens (k)*
o4-mini	+ CoT [25]	94.11 ↑0.26	97.86 ↓0.02	1.10 1.09×	76.06 ↑0.39	84.35 ↓0.78	1.60 1.06×
	+ ReAct [29]	91.96 ↓1.89	98.04 ↑0.16	1.70 1.68×	74.33 ↓1.34	84.10 ↓1.03	2.14 1.42×
	+ SelfRefine [16]	**94.31** ↑0.46	**98.57** ↑0.69	2.24 2.22×	75.71 ↑0.04	84.05 ↓1.08	3.23 2.14×
	+**VeriMaAS**	94.09 ↑0.24	98.17 ↑0.29	1.21 1.20×	**76.15** ↑0.48	**84.50** ↓0.63	1.71 1.13×
GPT 4o-mini	+ CoT [25]	82.25 ↑1.61	92.05 ↑1.18	0.71 1.42×	51.25 ↑0.99	62.07 ↑1.05	0.77 1.33×
	+ReAct [29]	82.77 ↑2.13	**93.10** ↑2.23	1.33 2.66×	**54.81** ↑4.55	**67.31** ↑6.29	1.36 2.34×
	+SelfRefine [16]	83.02 ↑2.38	92.48 ↑1.61	1.45 2.90×	51.47 ↑1.21	61.75 ↑0.73	1.59 2.74×
	+**VeriMaAS**	**83.09** ↑2.45	92.85 ↑1.98	1.26 2.52×	52.05 ↑1.79	64.02 ↑3.00	0.85 1.47×
Qwen2.5-14B	+ CoT [16]	68.11 ↑0.22	95.09 ↑0.96	0.70 1.08×	37.56 ↑3.78	**65.04** ↑3.00	0.91 1.07×
	+ ReAct [29]	62.53 ↓5.36	93.38 ↓0.75	1.09 1.68×	24.01 ↓9.77	57.57 ↓4.47	1.17 1.38×
	+ SelfRefine [16]	**74.71** ↑6.82	**95.96** ↑1.83	1.49 2.29×	41.67 ↑7.89	63.55 ↑1.51	1.85 2.18×
	+**VeriMaAS**	74.24 ↑6.35	95.78 ↑1.65	1.21 1.86×	41.47 ↑7.69	62.48 ↑0.44	1.61 1.89×

Table 3: Post-synthesis delta with and without PPA-aware optimization. We report the relative change in PPA metrics; *Pass@10* deltas are on entire benchmark *vs.* +**VeriMaAS** baselines (Table 1).

Model	VeriThoughts-PPA-Tiny [30]				VerilogEval-PPA-Tiny [19]			
	Pass@10	Δ*Area%*	Δ*Power%*	Δ*Delay%*	*Pass@10*	Δ*Area%*	Δ*Power%*	Δ*Delay%*
GPT-4o-mini	92.46 ↓0.39	9.18↓	1.6↑	10.32↓	62.93 ↓1.09	18.83↓	3.26↓	19.47↓
o4-mini	98.06 ↓0.11	14.86↓	0.00↓	15.87↓	84.18 ↓0.32	12.22↓	1.70↑	3.52↓
Qwen2.5-7B	86.33 ↑0.04	13.44↓	8.67↓	13.91↓	56.45 ↑0.00	28.79↓	4.07↑	24.58↓
Qwen2.5-14B	95.72 ↓0.06	16.8↓	14.57↓	21.39↓	62.33 ↓0.15	16.17↓	5.22↑	15.53↓
Qwen3-8B	99.04 ↓0.01	22.81↓	3.68↓	20.14↓	74.06 ↓0.12	9.98↓	6.04↓	9.03↓
Qwen3-14B	98.75 ↑0.00	9.99↓	2.12↑	9.94↓	75.64 ↓0.07	11.66↓	7.85↓	11.39↓

Closed-source models see smaller but consistent gains, reflecting that multi-agent orchestration adds value even when base performance is already high. As summarized in Table 2, *VeriMaAS* achieves these improvements with moderate token overhead, staying close to lightweight CoT [25] prompting and below iterative strategies like Self-Refine [16]. Table 2 also demonstrates that these gains hold for both *pass@1* and *pass@10* (with the exception of VerilogEval *pass@10* which sees marginal drop across all operators).

PPA-Aware Optimization Unlike fine-tuning that entangles performance objectives into model weights, our controller can be flexibly re-optimized for different goals. As a proof of concept, we set the cost term (Eq. 1) to the Yosys-reported area $C = \textbf{Area}(\mathcal{T}; q, a, \mathbb{O})$. We note that some benchmark tasks (*e.g.*, NAND gates) might offer little room for PPA optimization, so we use OpenAI o4 as pseudo-oracle to recommend the top 20 designs where RTL changes are more likely to affect downstream PPA metrics. We refer to these subsets as `-PPA-Tiny`. Table 3 reports the deltas in area, power, and delay between the standard and PPA-optimized *VeriMaAS* solutions. As expected, we observe area and runtime reductions by up to 28.79%. On the other hand, power and *pass@10* deltas suggest a trade-off: while many models show improvements, some (especially for VerilogEval) see slight power increases or marginal *pass@10* decrease.

4 Conclusion and Future Work

We introduced VeriMaAS, a multi-agent framework that integrates formal verification feedback into RTL code generation, alleviating reliance on costly fine-tuning and long reasoning traces. To support broader collaboration and development, we maintain our prototype as an open-source WiP repository [2]. Looking ahead, our aim is to further enhance the controller formulation by incorporating tree-search or RL-based policies following the latest workflow automation techniques from the broader AI community [10, 32, 17]. Moreover, we motivate expanding our orchestration signals to commercial EDA tools, and integrating (commercial) PDKs towards a comprehensive synthesis and PPA optimization beyond the current proof-of-concept.

[2]Link: https://github.com/dstamoulis/maas/tree/verimaas/verithoughts: documentation-migration in progress

References

[1] Manar Abdelatty, Jingxiao Ma, and Sherief Reda. Metrex: A benchmark for verilog code metric reasoning using llms. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pages 995–1001, 2025.

[2] Tutu Ajayi, Vidya A Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, et al. Toward an open-source digital flow: First learnings from the openroad project. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–4, 2019.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[4] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2023.

[5] Michael Fore, Simranjit Singh, and Dimitrios Stamoulis. Geckopt: Llm system efficiency via intent-based tool selection. In *Proceedings of the Great Lakes Symposium on VLSI 2024*, pages 353–354, 2024.

[6] Yonggan Fu, Yongan Zhang, Zhongzhi Yu, Sixu Li, Zhifan Ye, Chaojian Li, Cheng Wan, and Yingyan Celine Lin. Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.

[7] Google. Skywater pdk. `https://github.com/google/skywater-pdk`, 2020.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *preprint arXiv:2501.12948*, 2025.

[9] Raghu Vamshi Hemadri, Jitendra Bhandari, Johann Knechtel, Badri P Gopalan, Ramesh Narayanaswamy, Ramesh Karri, and Siddharth Garg. Veriloc: Line-of-code level prediction of hardware design quality from verilog code. *arXiv preprint arXiv:2506.07239*, 2025.

[10] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025.

[11] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[12] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[13] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Verilogeval: Evaluating large language models for verilog code generation. In *International Conference on Computer Aided Design*, pages 1–8. IEEE, 2023.

[14] Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pages 1–5, 2024.

[15] 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.

[16] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2023.

[17] Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: A modular approach to automated agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[18] Varatheepan Paramanayakam, Andreas Karatzas, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. Less is more: Optimizing function calling for llm execution on edge devices. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE, 2025.

[19] Nathaniel Pinckney, Christopher Batten, Mingjie Liu, Haoxing Ren, and Brucek Khailany. Revisiting verilogeval: Newer llms, in-context learning, and specification-to-rtl tasks. *arXiv preprint arXiv:2408.11053*, 2024.

[20] Simranjit Singh, Michael Fore, Andreas Karatzas, Chaehong Lee, Yanan Jian, Longfei Shangguan, Fuxun Yu, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. Llm-dcache: improving tool-augmented llms with gpt-driven localized data caching. In *2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4. IEEE, 2024.

[21] Dimitrios Stamoulis and Diana Marculescu. Geo-olm: Enabling sustainable earth observation studies with cost-efficient open language models & state-driven workflows. In *Proceedings of the ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies*, 2025.

[22] Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Benchmarking large language models for automated verilog rtl code generation. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2023.

[23] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

[24] YunDa Tsai, Mingjie Liu, and Haoxing Ren. Rtlfixer: Automatically fixing rtl syntax errors with large language model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

[25] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[26] Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, 2013.

[27] Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[28] Mengwei Xu, Dongqi Cai, Wangsong Yin, Shangguang Wang, Xin Jin, and Xuanzhe Liu. Resource-efficient algorithms and systems of foundation models: A survey. *ACM Computing Surveys*, 57(5):1–39, 2025.

[29] 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.

[30] Patrick Yubeaton, Andre Nakkab, Weihua Xiao, Luca Collini, Ramesh Karri, Chinmay Hegde, and Siddharth Garg. Verithoughts: Enabling automated verilog code generation using reasoning and formal verification. *arXiv:2505.20302*, 2025.

[31] Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Proceedings of the 42nd International Conference on Machine Learning*, pages 1–12, 2025.

[32] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025.