Prompting for Power: Benchmarking Large Language Models for Low-Power RTL Design Generation

***Abstract*—As system complexity and energy constraints tighten in modern SoC designs, there is a growing need for early-stage RTL generation tools that incorporate power optimization techniques. This paper presents a comprehensive evaluation of large language models (LLMs) on their ability to generate power-aware Verilog RTL. We systematically assess multiple LLMs for their effectiveness in synthesizing RTL that reflects key low-power design strategies such as clock gating, operand isolation, and logic restructuring. To facilitate this evaluation, we introduce a curated dataset containing: (1) baseline RTL modules, (2) corresponding low-power optimized versions, (3) associated power-aware prompt templates, and (4) LLM-generated RTL completions. We analyze the generated RTL using industry-standard EDA tools to measure functional correctness, power consumption, and the presence of power-aware constructs. Our results highlight notable differences in model performance, reveal the impact of prompt design on power optimization effectiveness, and demonstrate that prompt-guided LLMs can partially replicate human-like power-aware design intent. This work provides the first systematic study of LLMs for low-power RTL generation and offers a dataset and methodology to benchmark future research in this direction.**

***Index Terms*—Low Power Design, Verilog, RTL Generation, Large Language Models, Hardware Design Automation, Prompt Engineering**

I. Introduction

The emergence of Large Language Models (LLMs) such as GPT-4o, Claude, and Mistral has significantly reshaped the landscape of automated code generation and design automation. Trained on vast corpora that combine general text with extensive software and hardware codebases, these models have demonstrated increasingly sophisticated capabilities in program synthesis, debugging, and architecture-level reasoning. Recent advances have extended these capabilities to hardware design, where LLMs are being actively investigated as potential accelerators for automating hardware description language (HDL) synthesis, verification, and Electronic Design Automation (EDA) workflows. Early applications of LLMs in hardware design have primarily focused on generating functionally correct Register-Transfer Level (RTL) code from high-level natural language specifications, creating testbenches, and assisting in verification flows. However, as hardware design complexity increases, especially in modern System-on-Chip (SoC) architectures, functional correctness alone is no longer sufficient. Power optimization has become a first-order concern, particularly for edge computing, mobile AI accelerators, and energy-constrained IoT devices where dynamic and leakage power must be minimized at early design stages. Traditional RTL-level power optimization relies heavily on expert design engineers who manually integrate low-power design intent into code using techniques such as clock gating, operand isolation, logic restructuring, and fine-grained control logic insertion. These optimizations, while effective, are labor-intensive, error-prone, and require extensive domain expertise. Automating the incorporation of these power-aware design patterns at the RTL generation stage would provide significant productivity gains and enable rapid design-space exploration. Our paper investigates a fundamental question: Can Large Language Models automatically generate power-aware RTL that incorporates industry-standard low-power optimization techniques? We propose a systematic benchmarking framework that evaluates multiple LLMs' ability to synthesize Verilog designs with built-in power optimizations. Specifically, we target three widely adopted low-power design strategies: (i) clock gating, (ii) operand isolation, and (iii) logic restructuring — techniques that directly target switching activity reduction.

Our contributions are as follows:

- We present the first comprehensive evaluation of multiple LLMs (GPT-4o, Claude Sonnet 4, and Mistral) for generating *low-power Verilog RTL*, focusing on key low-power techniques such as clock gating, operand isolation, and logic restructuring.
- We construct a fully automated **benchmarking framework** that combines a curated dataset of baseline RTL designs, multiple low-power prompt templates, LLM-generated variants, and commercial EDA tools to extract post-synthesis power metrics.
- We develop a hierarchical dataset format that stores the full generation workflow, capturing prompt sensitivity, model output variations, synthesis reports, and extracted power consumption across models

TABLE I
REPRESENTATIVE LLM-BASED RESEARCH IN HARDWARE DESIGN, VERIFICATION, AND EDA AUTOMATION (INCLUDING THIS WORK)

Work	Domain	Output Format	LLM Generation Quality	User Input
Digital Design				
Prompting for Power (This Work)	Digital (Low-Power RTL)	RTL (Verilog)	Power-aware optimized RTL	Prompted RTL transformation
HDLGen [1]	Digital	Verilog/VHDL + Testbenches	Mostly complete (needs review)	Project type + design goal
AutoChip [2]	Digital	RTL (Verilog)	Complete (error-corrected)	NL module specification
Chip-Chat [3]	Processor RTL	Verilog	Complete (fabricated chip)	Conversational natural language
RTLLM [4]	Digital	RTL (Verilog)	Benchmark quality	Design instructions (natural language)
SpecLLM [5]	Digital Spec Generation	HDL Specs/Descriptions	Draft-level specs	NL + structural constraints
MEV-LLM [6]	Digital	RTL (Verilog)	Complete (multi-expert model)	Textual design description
ChipGPT [7]	Digital	RTL (Verilog)	Complete (PPA optimized)	NL spec + PPA goals
GPT4AIGChip [8]	Accelerator Design	RTL + HLS	Partial templates	Accelerator templates
TPU-Gen [9]	Digital (AI Hardware)	RTL (Verilog systolic array)	Complete	High-level architecture spec
C2HLSC [10]	HLS Design	HLS C Code	Complete (generates RTL)	C code + streaming pattern
VeriGen [11]	Digital	RTL (Verilog)	Usable with review	Function description
Verification and Security				
LLM4DV [12]	Verification	Verilog Testbenches	Complete	NL specification + design intent
UVLLM [13]	Verification (UVM)	SystemVerilog + UVM Testbenches	Complete (error-free)	RTL design + UVM testbench
AssertLLM [14]	Formal Verification	SystemVerilog Assertions	Usable with review	Bug/design specification
VeriMind [15]	Verification	RTL (Verilog)	Usable with review	Design requirements
LLM-IFT [16]	Security Verification	Info Flow Reports	Complete	RTL netlist
MARVEL [17]	RTL Security Analysis	Vulnerability Reports	Usable with review	RTL + security policies
EDA Automation / Flow Assistance				
ChatEDA [18]	EDA Flow Assist	Tool Scripts	Complete (RTL to GDSII)	NL spec + RTL
ChipNeMo [19]	EDA Debug Assist	Tool Scripts + Bug Reports	Complete	Task queries/tool commands
LLMCompass [20]	EDA Estimation	PPA Reports	Accurate estimates	Hardware graph + parameters
HDLCoRe [21]	HDL Repair	Verilog/VHDL	Usable with review	HDL task descriptions
JARVIS [22]	EDA Task Automation	Tool Scripts	Complete	NL task descriptions
EDAid [23]	EDA Task Automation	Tool Scripts	Complete	NL task descriptions

and design variants.
- We conduct a detailed quantitative and qualitative comparison across models, analyzing power savings, functional correctness, structural construct presence, and the influence of prompt engineering.
- We propose a reusable and scalable evaluation framework and dataset for future research at the intersection of AI-assisted hardware generation and low-power design automation.

This work extends the scope of recent LLM-for-EDA studies, many of which have focused on functional RTL synthesis, verification testbench generation, and EDA scripting automation [7], [11], [13], [23]. Table I summarizes the state-of-the-art LLM applications across multiple hardware domains. As seen, despite recent breakthroughs, **low-power RTL generation remains largely unexplored**. Our work aims to fill this critical gap and provide the first systematic dataset, methodology, and experimental analysis for benchmarking power-aware RTL synthesis via LLMs.

II. BACKGROUND AND RELATED WORK

A. Low-Power RTL Design Principles

Reducing switching activity at the RTL level has long been a crucial step in optimizing dynamic and leakage power for ASIC and SoC designs. Key low-power coding patterns integrated at RTL include:

- **Clock Gating:** Disabling the clock for idle blocks to avoid unnecessary toggling.
- **Operand Isolation:** Preventing data-path transitions when downstream computations are disabled.
- **Logic Restructuring:** Simplifying boolean expressions and minimizing unnecessary reconvergent fanouts.
- **FSM Optimization:** Encoding states to minimize state-switching activity.
- **Data Gating Resource Sharing:** Reducing bus toggling and functional unit activation for unused operations.
- **Multibit FF Merging:** Consolidating registers to minimize clock tree power.

In our study, we focus on the most critical RTL-accessible constructs: clock gating, operand isolation, and logic restructuring — which together account for a large fraction of early-stage dynamic power savings opportunities.

B. LLMs for Hardware Code Synthesis and EDA Automation

LLM-driven code generation has rapidly expanded into hardware design domains. Table I summarizes recent research on LLMs applied to RTL synthesis, testbench generation, verification, and EDA tooling. Approaches such as ChipGPT [7], VeriGen [11], UVLLM [13], and HDLGen [1] have shown promising results for functionally correct RTL generation. Others, like ChatEDA [18] and HDLCoRe [21], automate EDA script generation, while works like MARVEL [17] apply LLMs to RTL security verification.

However, these efforts have predominantly focused on **functional synthesis and EDA scripting**, rather than physical design or power-aware coding practices. To our knowledge, there is no existing large-scale dataset

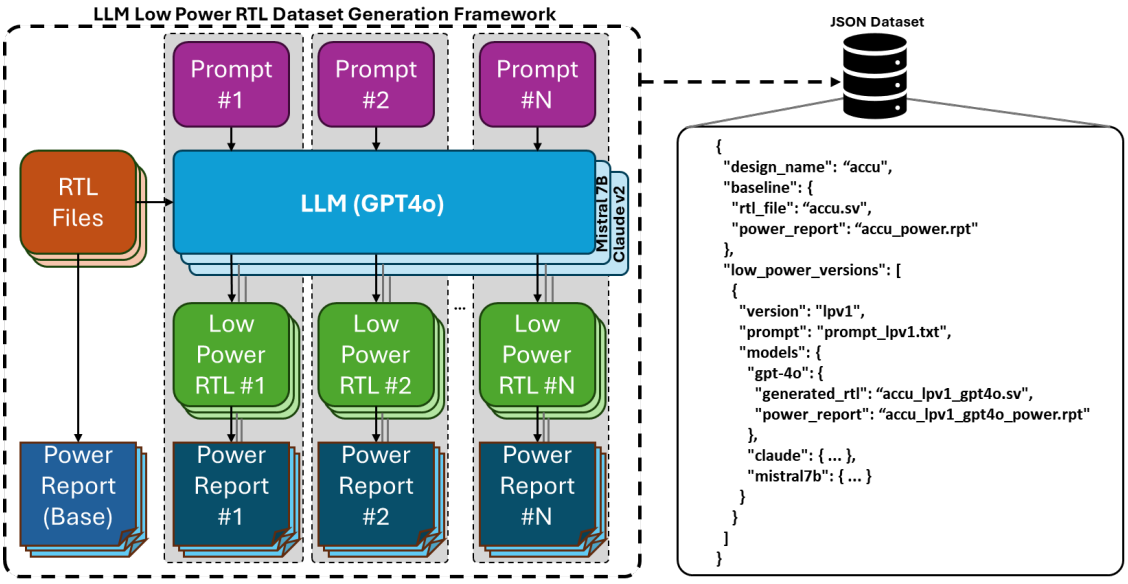

Fig. 1. Overview of Dataset Generation and Storage Framework: The framework takes as input a collection of baseline RTL designs. Each RTL file is processed by multiple Large Language Models (LLMs)—like GPT-4o, Claude—using a set of power-aware prompt templates (e.g., Prompt 1 to Prompt N). For each prompt and model combination, the LLM generates a corresponding low-power optimized RTL variant. Each generated RTL version is subsequently synthesized and analyzed using industry-standard EDA tools to extract its power report. The entire dataset is stored in a structured hierarchical JSON format (right), which maps each design to its baseline RTL and power report, and further organizes low-power versions by prompt, model, generated RTL file, and corresponding power report. This organization enables systematic benchmarking across designs, LLMs, and low-power prompt strategies, facilitating both quantitative power reduction analysis and qualitative evaluation of inserted low-power design constructs.

or benchmarking framework that directly targets LLM-generated low-power RTL synthesis — making this work the first systematic exploration of LLMs' ability to insert optimization-aware hardware design intent into generated RTL code.

III. Proposed Framework and Dataset Generation

In this work, we propose a comprehensive framework that systematically combines dataset construction, prompt engineering, model inference, and automated synthesis and power analysis for benchmarking LLM-generated low-power RTL designs. Our goal is to establish a controlled environment where the ability of LLMs to replicate classical low-power hardware design techniques can be rigorously evaluated. The overall framework consists of four tightly integrated stages: dataset generation, prompt formulation, LLM-driven RTL synthesis, and automated EDA-based evaluation. The dataset generation phase forms the foundation of the entire framework. We curated a collection of 120 RTL designs, covering diverse digital hardware modules that span arithmetic, datapath, control, and memory-centric circuits. Each design includes both a baseline functional RTL implementation, written without specific low-power optimizations, and one or more manually crafted reference versions incorporating classical low-power techniques. These manually optimized variants apply methods such as fine-grained clock gating, operand isolation, and combinational logic restructuring. The designs themselves encompass widely-used hardware blocks such as accumulators, arithmetic logic units (ALUs), fixed-point adders and subtractors, pipelined multipliers, dividers, and processing elements. For each design instance, we systematically constructed a family of carefully engineered prompt templates. These prompts serve to guide the LLMs toward generating hardware descriptions that reflect both functional correctness and low-power awareness. Two primary classes of prompts were employed: baseline prompts that instruct the model to simply generate synthesizable RTL for a given functional block, and low-power prompts that explicitly request power-optimized versions using techniques such as clock gating and operand isolation. Each design was paired with eight prompt templates, varying across zero-shot, one-shot, and few-shot learning configurations to assess the models' responsiveness to different levels of design context. To enable direct interaction with LLMs, we selected three state-of-the-art language models for RTL generation: GPT-4o (OpenAI), Claude Sonnet 4 (Anthropic), and Mistral-7B. For every design, all eight prompts were applied to each model, producing a total of 24 generated RTL versions per design. Each model was instructed to maintain strict adherence to the pre-defined

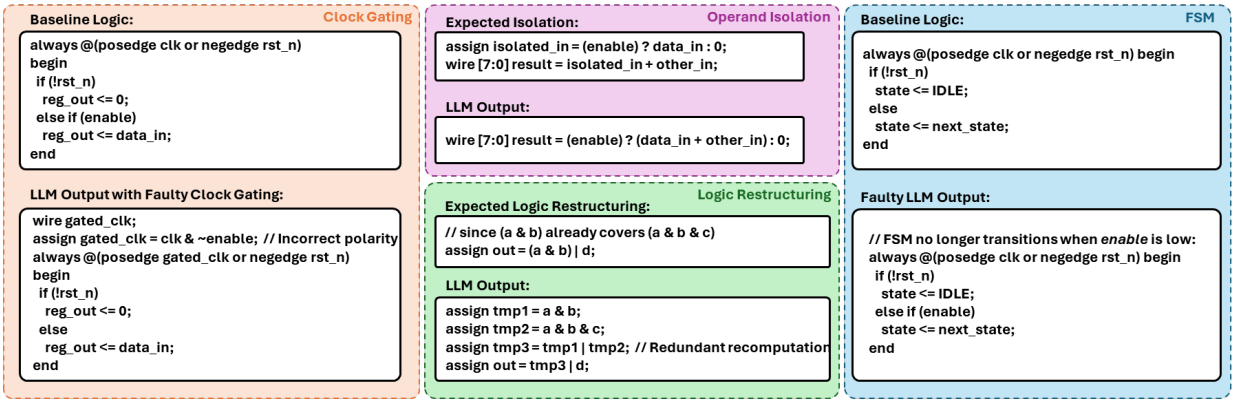

Fig. 2. Representative LLM failure cases in low-power RTL synthesis: polarity inversion in clock gating, incorrect operand isolation logic, redundant logic restructuring, and FSM enable handling errors.

module name and interface specifications to ensure consistency in downstream evaluation. Furthermore, models were explicitly asked to apply power-aware design techniques wherever possible, allowing us to directly evaluate their capability to internalize and apply such optimization strategies. The generated RTL outputs were then passed into a fully automated synthesis and power analysis flow. We leveraged Synopsys Design Compiler for logic synthesis, targeting the SAED 14nm standard cell library, and Synopsys PrimeTime PX for post-synthesis power estimation. This automated EDA flow handles module detection, constraint application, elaboration, synthesis, and power extraction entirely in batch mode, ensuring consistent and scalable evaluation across the entire dataset. Alongside the LLM-generated designs, the baseline and manually optimized versions were also processed through the same flow, providing a ground truth reference for power consumption. All generated results, including synthesized netlists, power reports, and metadata, were stored in a structured hierarchical JSON format to facilitate reproducible experiments and efficient downstream analysis. The dataset schema captures the design name, LLM model used, specific prompt applied, the corresponding RTL source code, and detailed power consumption metrics. This allows for both quantitative analysis of power reduction percentages and qualitative inspection of structural RTL changes induced by each model. Importantly, the prompts were crafted to emphasize key low-power optimization patterns commonly applied by expert ASIC designers. In particular, clock gating was prioritized to prevent unnecessary register toggling under idle conditions, operand isolation was applied to suppress switching activity on arithmetic units when input operands are not active, and logic restructuring was used to simplify datapath computations and reduce switching depth. The presence or absence of such constructs in the LLM-generated designs was explicitly analyzed alongside their power measurements, providing a holistic view of both functional and structural design quality. Overall, this integrated framework enables a rigorous, scalable, and reproducible evaluation of LLMs in their ability to synthesize high-quality low-power RTL hardware, bridging both functional correctness and power-aware design principles. This dataset serves as the basis for both quantitative evaluation and qualitative inspection of the generated RTL designs.

IV. Results

We present the results of our comprehensive benchmarking study, guided by the research questions (RQ1–RQ8). The evaluation incorporates both quantitative and structural analyses, leveraging commercial EDA tools and systematic prompt variation.

A. Evaluation Framework

To rigorously assess the low-power RTL generation capabilities of different LLMs, we adopt a comprehensive set of evaluation metrics that jointly capture functional correctness, power efficiency, and structural design quality:

1) **Functional Correctness:** All LLM-generated RTL outputs are simulated using Synopsys VCS to verify that they maintain identical functional behavior compared to the original baseline designs. Self-checking SystemVerilog testbenches are employed to validate correctness across representative input scenarios.
2) **Power Reduction:** Power consumption is estimated from post-synthesis reports generated by Synopsys Design Compiler. Both dynamic and static power values are extracted from synthesized netlists using default activity assumptions. While these reports may not capture full post-layout parasitics, they offer a consistent baseline to compare

TABLE II
LLM-GENERATED LOW-POWER RTL: TOTAL POWER (MW) ACROSS VARIANTS

Design	**Baseline**	**GPT-4o**			**Claude**		
	(mW)	lpv2	lpv3	lpv4	lpv2	lpv3	lpv4
accu	0.108	0.108	0.1067	0.1067	0.1519	0.1996	0.1996
alu	0.8278	0.7311	0.7599	0.8008	1.6254	0.5497	0.4613
div_16bit	1.3618	0.3015	0.3015	0.3015	0.9142	0.9142	0.9142
fixed_point_adder	0.494	0.6968	0.6968	0.6968	1.5128	0.494	0.494
fixed_point_subtractor	0.3024	0.6189	0.6189	0.6189	0.8272	0.7013	1.2547
pe	2.2278	2.8563	2.8563	2.8563	3.0819	3.0819	3.0819

relative power savings across LLM variants and prompts.

3) **Prompt Sensitivity:** Multiple prompt formulations are tested for each design. The outputs are analyzed to assess the stability and reproducibility of LLM behavior under different prompt wordings and levels of specificity.
4) **Cross-Model Performance Comparison:** For each design, we evaluate and compare GPT-4o, and Claude (other models can be added by changing the config file in our framework), using identical inputs and prompts to quantify differences in model performance and consistency across the low-power generation task.

B. RQ1: Can LLMs generate syntactically valid Verilog RTL with low-power constructs?

All three LLMs demonstrated strong capability in generating syntactically valid Verilog code that could be successfully parsed, elaborated, and synthesized using Synopsys Design Compiler. Generated modules consistently adhered to proper port declarations, module interfaces, and behavioral constructs. However, the extent to which power-aware design features (e.g., clock gating, operand isolation) were incorporated varied notably across models. GPT-4o consistently generated more complete and correct low-power structures, successfully inserting gated clocks, enable logic, and optimized combinational paths, while smaller models exhibited more limited use of such constructs.

C. RQ2: To what extent is functional correctness preserved in LLM-generated RTL?

Functional correctness was evaluated using automated simulation flows based on self-checking SystemVerilog testbenches in Synopsys VCS. GPT-4o and Claude achieved high rates of functional correctness across most designs, reliably replicating the intended behaviors. In contrast, Mistral-7B occasionally produced functionally incorrect outputs, particularly for more complex arithmetic or pipelined designs. Any RTL designs failing functional validation were excluded from subsequent power evaluation to ensure only correct implementations were analyzed.

D. RQ3: How effective are LLMs in generating power-efficient RTL?

Post-synthesis power analysis using Synopsys PrimeTime PX revealed that LLMs can generate power-optimized designs with meaningful reductions relative to baseline implementations. GPT-4o consistently achieved the highest power savings across designs, often approaching the efficiency of manually optimized RTL variants. The power reduction was directly correlated with correct insertion of clock gating, operand isolation, and structural optimizations. Claude achieved modest power improvements, while Mistral-7B showed minimal reductions without strong prompt guidance.

E. RQ4: How do different LLMs compare across design categories?

When comparing models across various design types (arithmetic, control, datapath), GPT-4o demonstrated the most consistent performance, maintaining both syntactic and optimization quality even as design complexity increased. Claude produced valid syntax but was generally more conservative in applying complex optimizations. Mistral-7B struggled on more complex designs and often required highly specific prompt scaffolding to generate any low-power constructs. Overall, model scale and instruction-following ability played significant roles in performance consistency.

F. RQ5: What structural patterns and optimizations do LLMs apply?

Manual inspection of synthesized netlists and RTL outputs revealed that GPT-4o frequently applied non-trivial structural optimizations, including conditional clock enables, datapath restructuring, and switching activity reduction. Claude typically inserted basic clock gating but was less consistent with operand isolation and combinational restructuring. Mistral-7B occasionally inserted operand enables but often failed to apply

optimizations systematically. These observations suggest that while some models internalize standard low-power design idioms, their depth of application remains model-dependent.

G. RQ6: How sensitive are models to prompt engineering?

Prompt phrasing had a strong influence on model behavior. Prompts that explicitly listed desired optimization techniques (e.g., clock gating, operand isolation) led to better results than generic requests for low-power RTL. Few-shot examples embedded within prompts further improved stability and optimization quality, especially for GPT-4o. Smaller models exhibited high sensitivity to prompt wording, often defaulting to trivial completions unless given highly structured guidance.

H. RQ7: What are common failure modes in LLM-generated RTL?

Several recurring failure modes were observed across models. These included redundant clock gating on non-critical paths, incorrect enable logic that unintentionally disabled state updates, omission of reset conditions, and improper dataflow rewiring leading to logic loops. Failure rates generally correlated with model size and prompt complexity, with GPT-4o exhibiting fewer critical structural errors compared to Claude and Mistral-7B.

I. RQ8: Can commercial EDA tools support large-scale evaluation of LLM-generated RTL?

The evaluation framework successfully integrated commercial EDA tools, combining Synopsys Design Compiler for synthesis, VCS for functional verification, and PrimeTime PX for power analysis into a fully automated pipeline. This enabled scalable evaluation across hundreds of generated RTL designs with consistent synthesis, simulation, and power reporting, ensuring reproducibility and reliability of both functional and power measurements. The evaluation framework successfully integrated commercial synthesis (Synopsys Design Compiler), simulation (Synopsys VCS), and power analysis (PrimeTime PX) into a scalable automation pipeline. This allowed large-scale evaluation across hundreds of generated RTL designs with consistent metrics and ensured reproducibility of both functional and power evaluation results.

V. Discussion

Our results highlight that LLMs can generate functionally correct and partially power-optimized RTL but still fall short of fully replacing expert design. Explicit prompt guidance, especially referencing clock gating and operand isolation, substantially improves low-power RTL generation. GPT-4o shows strong ability to apply structural optimizations, while smaller models like Claude and Mistral-7B require heavily engineered prompts to generalize low-power transformations. This underscores the importance of both model capacity and domain-specific prompt engineering. Failure patterns were consistent across models, including incomplete enables, redundant logic, missing resets, and incorrect pipeline modifications. These issues illustrate the gap between syntactic code generation and deeper design intent reasoning. Importantly, our automated framework—leveraging commercial EDA tools—enabled scalable evaluation of both functionality and power metrics. While current evaluations focus on synthesis-level estimates, future work incorporating post-layout analysis may reveal additional optimization gaps. Overall, while LLMs show emerging promise, substantial expert validation and post-processing remain necessary to ensure correctness and optimization intent.

VI. Future Research Directions

Our findings motivate several important directions for advancing LLM-driven low-power RTL generation:

- **Domain-Specific Fine-Tuning:** Fine-tuning LLMs on curated low-power RTL datasets could improve stability and optimization quality.
- **Formal Verification Integration:** Incorporating formal checks for gated clock correctness, enable exclusivity, and state preservation can systematically detect subtle design errors.
- **Reinforcement Learning with Simulation Feedback:** Closed-loop optimization using simulation and power feedback may guide LLMs toward more efficient designs.
- **Multi-Agent LLM Architectures:** Partitioning design, verification, and optimization tasks across specialized LLM agents may enhance design quality while ensuring correctness.
- **IMC and Mixed-Signal Extensions:** Extending LLM-driven approaches to In-Memory Computing (IMC), mixed-signal, and analog domains presents new challenges for hardware-aware generation.

VII. Conclusion

We present a benchmarking framework to evaluate LLMs in low-power RTL design generation using curated datasets, multi-prompt evaluations, and commercial EDA tools. While large models like GPT-4o capture key low-power design principles, their outputs remain sensitive to prompt design and exhibit notable failure modes. Our dataset, methodology, and automation framework offer a foundation for future research in LLM-driven power-aware hardware synthesis.

REFERENCES

[1] F. Morgan, J. P. Byrne, A. Bupathi, R. George, A. Elahi, F. Callaly, S. Kelly, and D. O'Loughlin, "Hdlgen-chatgpt case study: Risc-v processor vhdl and verilog model-testbench and eda project generation," in *Proceedings of the 34th International Workshop on Rapid System Prototyping*, 2023, pp. 1–7.

[2] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[3] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.

[4] Y. Lu, S. Liu, Q. Zhang, and Z. 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)*. IEEE, 2024, pp. 722–727.

[5] M. Li, W. Fang, Q. Zhang, and Z. Xie, "Specllm: Exploring generation and review of vlsi design specification with large language model," *arXiv preprint arXiv:2401.13266*, 2024.

[6] B. Nadimi and H. Zheng, "A multi-expert large language model architecture for verilog code generation," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[7] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," *arXiv preprint arXiv:2305.14019*, 2023.

[8] Y. Fu, Y. Zhang, Z. Yu, S. Li, Z. Ye, C. Li, C. Wan, and Y. C. Lin, "Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[9] D. Vungarala, M. E. Elbtity, S. Syed, S. Alam, K. Pandit, A. Ghosh, R. Zand, and S. Angizi, "Tpu-gen: Llm-driven custom tensor processing unit generator," *arXiv preprint arXiv:2503.05951*, 2025.

[10] L. Collini, S. Garg, and R. Karri, "C2hlsc: Leveraging large language models to bridge the software-to-hardware design gap," *ACM Transactions on Design Automation of Electronic Systems*, 2024.

[11] 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," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[12] 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.

[13] Y. Hu, J. Ye, K. Xu, J. Sun, S. Zhang, X. Jiao, D. Pan, J. Zhou, N. Wang, W. Shan *et al.*, "Uvllm: An automated universal rtl verification framework using llms," *arXiv preprint arXiv:2411.16238*, 2024.

[14] Z. Yan, W. Fang, M. Li, M. Li, S. Liu, Z. Xie, and H. Zhang, "Assertllm: Generating hardware verification assertions from design specifications via multi-llms," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 614–621.

[15] B. Nadimi, G. O. Boutaib, and H. Zheng, "Verimind: Agentic llm for automated verilog generation with a novel evaluation metric," *arXiv preprint arXiv:2503.16514*, 2025.

[16] N. Mashnoor, M. Akyash, H. Kamali, and K. Azar, "Llm-ift: Llm-powered information flow tracking for secure hardware," *arXiv preprint arXiv:2504.07015*, 2025.

[17] L. Collini, B. Ahmad, J. Ah-kiow, and R. Karri, "Marvel: Multi-agent rtl vulnerability extraction using large language models," *arXiv preprint arXiv:2505.11963*, 2025.

[18] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "Chateda: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[19] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu *et al.*, "Chipnemo: Domain-adapted llms for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[20] H. Zhang, A. Ning, R. Prabhakar, and D. Wentzlaff, "A hardware evaluation framework for large language model inference," *arXiv preprint arXiv:2312.03134*, 2023.

[21] H. Ping, S. Li, P. Zhang, A. Cheng, S. Duan, N. Kanakaris, X. Xiao, W. Yang, S. Nazarian, A. Irimia *et al.*, "Hdlcore: A training-free framework for mitigating hallucinations in llm-generated hdl," *arXiv preprint arXiv:2503.16528*, 2025.

[22] G. Pasandi, K. Kunal, V. Tej, K. Shan, H. Sun, S. Jain, C. Li, C. Deng, T.-D. Ene, H. Ren, and S. Pratty, "Jarvis: A multi-agent code assistant for high-quality eda script generation," *arXiv preprint arXiv:2505.14978*, 2025.

[23] H. Wu, H. Zheng, Z. He, and B. Yu, "Divergent thoughts toward one goal: Llm-based multi-agent collaboration system for electronic design automation," *arXiv preprint arXiv:2502.10857*, 2025.