CreativEval: Evaluating Creativity of LLM-Based Hardware Code Generation

Matthew DeLorenzo
Electrical and Computer Engineering
Texas A&M University
College Station, USA
matthewdelorenzo@tamu.edu

Vasudev Gohil
Electrical and Computer Engineering
Texas A&M University
College Station, USA
gohil.vasudev@tamu.edu

Jeyavijayan Rajendran
Electrical and Computer Engineering
Texas A&M University
College Station, USA
jv.rajendran@tamu.edu

2024 IEEE LLM Aided Design Workshop (LAD) | 979-8-3503-7608-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/LAD62341.2024.10691798

***Abstract*—Large Language Models (LLMs) have proved effective and efficient in generating code, leading to their utilization within the hardware design process. Prior works evaluating LLMs' abilities for register transfer level code generation solely focus on functional correctness. However, the creativity associated with these LLMs, or the ability to generate novel and unique solutions, is a metric not as well understood, in part due to the challenge of quantifying this quality.**

To address this research gap, we present `CreativEval`, a framework for evaluating the creativity of LLMs within the context of generating hardware designs. We quantify four creative sub-components, fluency, flexibility, originality, and elaboration, through various prompting and post-processing techniques. We then evaluate multiple popular LLMs (including GPT models, CodeLlama, and VeriGen) upon this creativity metric, with results indicating GPT-3.5 as the most creative model in generating hardware designs.

***Index Terms*—Hardware Design, LLM, Creativity**

I. Introduction

Recent advancements within artificial intelligence, machine learning, and computing performance have resulted in the development of LLMs, which have quickly proven to be a widely applicable and successful solution when applied to a variety of text-based tasks [1]. After extensive training on large quantities of text data, these transformer-based models [2] have demonstrated the ability to not only successfully interpret the contextual nuances of a provided text (or prompt), but also generate effective responses to a near human-like degree [3]. This can take the form of summarizing a document, answering and elaborating upon questions, and even generating code. The effectiveness and versatility of LLMs regarding textual understanding have resulted in their adoption within various applications, such as language translation [4], customer service chat-bots [5], and programming assistants [1].

Furthermore, the potential of LLM code generation has recently been explored within the integrated circuit (IC) design process [6], such as within the logic design stage. With chip designs continually growing in scale and complexity, efforts to increase the automation of this task through LLMs have been explored. This includes the evaluation of LLMs' ability to generate hardware design codes from English prompts, leading to promising initial results within various frameworks [7]–[10].

With the goal of further optimizing these LLMs to the level of an experienced hardware designer, many research efforts have focused on improving performance regarding of code functionality. This includes testing LLM fine-tuning strategies and prompting methods for domain-optimized performance, such as register transfer level (RTL) code generation.

However, another dimension to consider when evaluating the ability of a designer, absent from previous evaluations, is creativity. This term refers to the capacity to think innovatively—the ability to formulate new solutions or connections that are effective and unconventional [11]. When applied to hardware code generation, this can take the form of writing programs that are not only correct, but also novel, surprising, or valuable when compared to typical design approaches. This quality is essential to understanding the greater potential of LLMs as a tool for deriving new approaches to hardware design challenges, rather than simply a method to accelerate existing design practices. With a quantitative method of measuring this concept of creativity within LLM hardware generation, valuable insights could be derived, such as how performance could be further improved, or how LLMs can be best utilized within the hardware design process.

To address this absence within the analysis of LLM-based RTL code generation, we propose a comparative evaluation framework in which the creativity of LLMs can be effectively measured. This assessment is composed of four cognitive subcategories of creativity (fluency, flexibility, originality, and elaboration), which are quantified and evaluated within the context of generating functional Verilog modules. This approach utilizes various prompting structures, generation strategies, and post-processing methods, from which the quality and variations of responses are utilized to generate a metric for creativity. This work presents the following contributions:

- To the best of our knowledge, we propose the first framework from which a metric for creativity is defined for LLMs within the context of hardware design and code generation.
- We provide a comparative evaluation between state-of-the-art LLMs upon our creativity metric and its components, with GPT-3.5 achieving the highest result.
- To enable future research, we will open-source our framework codebase and datasets here: https://github.com/matthewdelorenzo/CreativEval/

II. Background and Related Work

A. LLMs for Code Generation and Hardware Design

Many state-of-the-art LLMs have demonstrated remarkable success in generating code when provided only with a natural language description, such as GPT-3.5/4 [12], BERT [13], and Claude [14], revolutionizing the software development process. These models demonstrate promising performance in code functionality, such as GPT-4 generating correct code for 67% of programming tasks in the HumanEval benchmark in a single response (pass@1) [15]–[17].

Therefore, the applications of LLMs within hardware design through RTL code generation are explored within various studies, such as DAVE [18] which utilized GPT-2 for this task. VeriGen [7] then demonstrated that fine-tuning smaller models (CodeGen) upon a curated RTL dataset can outperform larger models in RTL tests. VerilogEval [19] presents enhanced LLM hardware generation through supervised fine-tuning, and provides an RTL benchmark for evaluating functionality in RTL generation. ChipNeMo [9] applied fine-tuning upon open-source models (Llama2 7B/13B) for various hardware design tasks. RTLCoder [20] presents an automated method for expanding the RTL dataset used for fine-tuning, resulting in a 7B-parameter model that outperforms GPT-3.5 on RTL benchmarks. Other works, including RTLLM [21] and Chip-Chat [8], explore prompt engineering strategies to enhance the quality and scale of LLM-generated designs. Although there is a plethora of work on LLM-based RTL generation, none of these assess the creative component of LLMs in the hardware design process. We address this shortcoming in this work.

B. Evaluating Creativity

Prior cognitive science studies [22]–[25] have explored methods in which creative thinking can be effectively measured. A widely accepted creativity model [24] defines four primary cognitive dimensions from which divergent thinking, or the ability to generate creative ideas through exploring multiple possible solutions [26], can be measured—fluency, flexibility, originality, and elaboration.

- **Fluency.** The quantity of relevant and separate ideas able to be derived in response to a single given question.
- **Flexibility.** The ability to formulate alternative solutions to a given problem or example across a variety of categories.
- **Originality.** A measure of how unique or novel a given idea is, differing from typical responses or solutions.
- **Elaboration.** The ability to expand upon or refine a given idea. This can include the ability to construct complex solutions utilizing provided, basic concepts.

These subcategories have been widely in evaluating human creativity within educational research, including various studies of students [27]–[29] as a metric for effective learning. Furthermore, recent works explore the intersection between cognitive science and LLMs [30]–[32], in which the creativity of LLMs are evaluated within the context of natural language, demonstrating near-human like performance in many cases [31]. In particular, [33] utilizes the four creative subcategories to evaluate LLMs across multiple language-based cognitive tasks. However, this framework has not been adapted to LLMs within the context of generating hardware code. To this end, we devise our creativity evaluation framework for LLM-based hardware code generation.

```
1 //Create a full adder.
2 //A full adder adds three bits (including
  carry-in) and produces a sum and carry-out.
3
4 module top_module (
5     input a, b, cin,
6     output cout, sum );
```

Listing 1: Fluency/Originality prompt example

III. CreativEval Framework

Given a target LLM, our `CreativEval` framework, as shown in Fig. 1, seeks to evaluate the creativity associated with LLMs in hardware code generation. `CreativEval` evaluates the previously defined subcategories of creativity—fluency, flexibility, originality, and elaboration. To this end, we query the target LLM with different Verilog-based prompts, and analyze the responses through various methods of post-processing to calculate the desired metrics, as explained below.

A. Fluency

To capture the quantity of relevant and separate ideas in our context, we define fluency as the average number of unique Verilog solutions generated by the target LLM in response to a given prompt. Our prompts contain a brief English description of the module and the module's declaration, as shown in Listing 1. Each prompt is provided as input to the LLM, with the response intended to be the completed implementation of the module. As the inference process of LLMs contain variations in the generated responses, we generate t responses for each prompt to estimate the average performance.

Upon generating all responses, each response is then tested for functionality against the module's associated testbench. If all test cases pass, the module is considered functional. Then, for each prompt, the functional responses (if any) are collected and compared to identify if they are unique implementations.

This is done through GNN4IP [34], a tool utilized to assess the similarities between circuits. By representing two Verilog modules as a data-flow graph (DFG), GNN4IP generates a similarity score within [-1,1], with larger values indicating a higher similarity. Each correct generated solution from the LLM is input into GNN4IP, and compared to its ideal solution, or "golden response". Upon the generation of each similarity value for a given prompt, these results are then compared to determine how many unique values are in the response set, indicating the number of distinct solutions.

Given that there are a set of p total prompts in the dataset, the LLM generates t responses for each. After evaluating the functionality of these results, there is then a subset n prompts that contain at least one success (functional module generation). For each of these n prompts, there is a sub-total

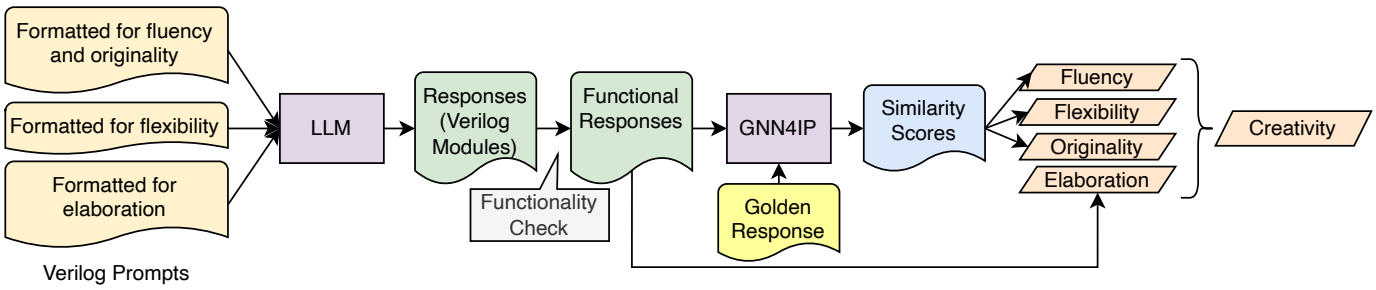

Fig. 1: Experimental Framework - calculating creativity of LLMs in Verilog code generation.

```
// You are a professional hardware designer that writes correct, fully functional Verilog modules.
// Given the fully implemented example of the Verilog module below:

module true_module(
    input a, b, cin,
    output cout, sum );
    assign sum = a ^ b ^ cin;
    assign cout = a & b | a & cin | b & cin;
endmodule

// Finish writing a different and unique implementation of the provided true_module in the module below, top_module.
module top_module (
    input a, b, cin,
    output cout, sum );
```

Listing 2: Flexibility prompt example

of the t responses that are functional, defined as m. Each of these m functional responses, r, are defined as the set $R = \{r_{1n}, ..., r_{mn}\}$. The GNN4IP similarity value is then found for each response in R, represented as the function S. The number of unique similarity values is then determined within the set, and normalized to total t responses. This process is repeated for all n successful prompts and averaged to define the fluency metric F below:

$$F = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|S(R_i)|}{t}\right) \tag{1}$$

B. Flexibility

Flexibility is quantified as the ability of the LLM to generate an alternative implementation of a Verilog module when provided with a solution. The prompts for this metric are constructed for a set of Verilog modules in which a correct solution (the golden response) is included (Listing 2). The LLM then rewrites the Verilog module, ideally resulting in a functional and unique implementation.

As before, t responses are generated for each of the p total prompts. After all responses are checked for functionality, n prompts have at least one functional response, each with m functional responses. These functional responses are compared directly with the golden response (through GNN4IP) to identify their similarity value. If the similarity value s is lower than a given threshold on the scale [-1,1], the response is considered an alternative solution, shown in Equation 2. For each successful prompt, the response with minimum similarity is found and evaluated against the threshold. Then, the total amount of n prompts with a response less than the threshold is determined, and normalized by the total prompts n. The final flexibility metric X is then defined below:

$$T(s) = \begin{cases} 1 & \text{if } s < 0 \\ 0 & \text{if } s \geq 0 \end{cases} \tag{2}$$

$$X = \frac{1}{n}\sum_{i=1}^{n}\Big(T[\min S(R_i)]\Big) \tag{3}$$

C. Originality

The originality metric is defined as the variance (uniqueness) of an LLM-generated Verilog module in comparison to a typical, fully functional implementation. This metric is derived from the similarity value (generated through GNN4IP) between successful generations and their golden response.

The originality experiment follows the same prompt structure and procedure as described in III-A. For each prompt, the response with the minimum similarity value is found. Then, the similarity values [-1, 1] are re-normalized to be on scale of [0, 1] with 1 indicating the least similarity (i.e. most original). These results are averaged over all n prompts, with the final originality metric O is described below:

$$O = \frac{1}{n}\sum_{i=1}^{n}\frac{(-\min S(R_i) + 1)}{2} \tag{4}$$

D. Elaboration

To measure an LLM's capacity for elaboration, the LLM is provided with multiple smaller Verilog modules in a prompt, and tasked with utilizing them to implement a larger, more complex module. As this metric requires multi-modular designs, a separate set of Verilog modules is utilized in constructing the prompts, as shown in Listing 3.

Multiple LLM responses are generated for each module, which are all then checked for functionality. For all given functional solutions, the responses are checked to see if the solution utilizes the smaller modules (as opposed to a single modular solution). If any of the responses for a given prompt are both functional and utilize the smaller modules, it is

TABLE I: Comparison of different LLMs in terms of creativity and its subcategories

LLM	Functionality	Fluency	Flexibility	Originality	Elaboration	Creativity
CodeLlama-7B [35]	0.2417	0.1483	0.0000	0.2926	0.2222	0.1658
CodeLlama-13B [36]	0.3167	0.1611	0.0260	**0.3021**	**0.3333**	0.2056
VeriGen-6B [37]	0.3667	0.1244	0.1000	0.2527	**0.3333**	0.2026
VeriGen-16B [38]	0.3250	0.1189	0.0556	0.2771	**0.3333**	0.1962
GPT-3.5 [39]	0.3083	0.1343	**0.1600**	0.2526	**0.3333**	**0.2201**
GPT-4 [40]	**0.3750**	**0.1644**	0.0795	0.2657	**0.3333**	0.2107

```
// You are given a module add16 that performs a 16-bit addition.
//Instantiate two of them to create a 32-bit adder.

module add16 (input[15:0] a, input[15:0] b, input cin, output[15:0] sum, output cout );

module top_module (
    input [31:0] a,
    input [31:0] b,
    output [31:0] sum
);
```

Listing 3: Elaboration prompt example

considered a positive instance of elaboration. Given p total Verilog prompts, of which n have at least one response that demonstrates elaboration, the metric is specified as:

$$E = \left(\frac{n}{p}\right) \tag{5}$$

E. Creativity: Putting It All Together

Given each of the subcategories associated with creativity defined above, the metrics are then combined to define the overall creativity of a given LLM in Verilog hardware design.

$$C = (0.25)F + (0.25)X + (0.25)O + (0.25)E \tag{6}$$

IV. Experimental Evaluation

A. Experimental Setup

We evaluate multiple LLMs using the `CreativEval` framework, including CodeLlama 7B [35] and 13B [36], VeriGen 6B [37] and 16B (8-bit quantized) [38], GPT-3.5 [39], and GPT-4 [40]. The inference process of the VeriGen and CodeLlama models is performed locally on an NVIDIA A100 GPU (80 GB), while GPT-3.5/4 are queried through the OpenAI API. All scripts are written in Python 3.10, with Icarus Verilog 10.3 as the simulator for evaluating functionality. The open-source GNN4IP repository is used to generate similarity scores. The HDLBits [41] prompt dataset (sourced from AutoChip [42]) utilized for functionality, fluency, and originality consists of 111 single-module prompts, with 9 multi-module prompts used for elaboration, each containing a solution and testbench. The functionality metric (pass@10) is measured on all 120 prompts. When generating responses, all LLMs use the following parameters: temperature=0.3; max_tokens=1024; top_k=10; top_p=0.95. All responses are trimmed to the first generated instance of "endmodule".

B. Results

Table I summarizes the results for all LLMs for each subcategory of creativity. In evaluating **fluency**, GPT-4 had the highest quantity of separate and correct Verilog solutions for a given module, with CodeLlama-13B achieving similar results. The VeriGen models comparatively struggled in this metric, partly due to repeated generations of similar implementations.

Regarding **flexibility**, GPT-3.5 had the highest rate of generating alternative solutions to provided modules across most models. The models that struggled (e.g., CodeLlama) produced results that were often direct copies of the provided module, indicating the ability to understand the prompt's natural language description as an important factor that determined flexibility.

As for **originality**, the GPT models had slightly worse performance than the others, with CodeLlama performing best. This means that the successful solutions provided with the GPT models were, on average, closer to the ideal solution. This could be due to its large size and training dataset, resulting in a more direct retrieval of existing solutions or coding practices.

Elaboration was largely similar for all modules, as the HDLBits dataset for this metric is comparatively small (9 modules). The models primarily excelled in correctly connecting the input and output parameters between separate modules, while struggling to generate the larger module solution.

Overall, the GPT models were the most **creative**, with GPT-3.5 as the best, and CodeLlama-7B as the least creative. Creativity is shown to slightly drop for the larger model sizes.

V. Conclusion

Recent studies on LLMs regarding their applications to hardware design have applied many optimization strategies to increase the performance in terms of functional correctness. However, these studies do not investigate the creativity associated with LLMs in their ability to generate solutions, largely due to the lack of an effective metric. In this work, we propose `CreativEval`, a framework to evaluate the creativity of LLMs in generating hardware code. By evaluating multiple popular LLMs, we perform a comparative analysis concluding that GPT-3.5 had the greatest creativity. Future research in this direction can evaluate more LLMs upon larger prompt sets.

Acknowledgment

The authors acknowledge the support from the Purdue Center for Secure Microelectronics Ecosystem – CSME#210205. This work was also partially supported by the National Science Foundation (NSF CNS–1822848 and NSF DGE–2039610).

REFERENCES

[1] Tim Keary, "12 Practical Large Language Model (LLM) Applications," https://www.techopedia.com/12-practical-large-language-model-llm-applications, 2023, [Online; last accessed 21-Nov-2023].

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[3] Z. G. Cai, X. Duan, D. A. Haslett, S. Wang, and M. J. Pickering, "Do large language models resemble humans in language use?" 2024.

[4] T. Kocmi and C. Federmann, "Large language models are state-of-the-art evaluators of translation quality," 2023.

[5] K. Pandya and M. Holia, "Automating customer service using langchain: Building custom open-source gpt chatbot for organizations," 2023.

[6] R. Zhong, X. Du, S. Kai, Z. Tang, S. Xu, H.-L. Zhen, J. Hao, Q. Xu, M. Yuan, and J. Yan, "Llm4eda: Emerging progress in large language models for electronic design automation," *arXiv preprint arXiv:2401.12224*, 2023.

[7] 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," 2023.

[8] 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, Sep. 2023. [Online]. Available: http://dx.doi.org/10.1109/MLCAD58807.2023.10299874

[9] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang *et al.*, "Chipnemo: Domain-adapted llms for chip design," 2024.

[10] M. DeLorenzo, A. B. Chowdhury, V. Gohil, S. Thakur, R. Karri, S. Garg, and J. Rajendran, "Make every move count: Llm-based high-quality rtl code generation using mcts," 2024.

[11] M. Runco and G. Jaeger, "The standard definition of creativity," *Creativity Research Journal - CREATIVITY RES J*, vol. 24, pp. 92–96, 01 2012.

[12] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman *et al.*, "Gpt-4 technical report," 2024.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[14] [Online]. Available: https://www.anthropic.com/news/claude-3-haiku

[15] Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang, "Wizardcoder: Empowering code large language models with evol-instruct," 2023.

[16] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan *et al.*, "Evaluating large language models trained on code," 2021.

[17] Y. Wang, H. Le, A. D. Gotmare, N. D. Q. Bui, J. Li, and S. C. H. Hoi, "Codet5+: Open code large language models for code understanding and generation," 2023.

[18] H. Pearce, B. Tan, and R. Karri, "Dave: Deriving automatically verilog from english," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, ser. MLCAD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 27–32. [Online]. Available: https://doi.org/10.1145/3380446.3430634

[19] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," 2023.

[20] 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," 2024.

[21] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," 2023.

[22] L. S. Almeida, L. P. Prieto, M. Ferrando, E. Oliveira, and C. Ferrándiz, "Torrance test of creative thinking: The question of its construct validity," *Thinking Skills and Creativity*, vol. 3, no. 1, pp. 53–58, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1871187108000072

[23] S. L. Doerr, "Conjugate lateral eye movement, cerebral dominance, and the figural creativity factors of fluency, flexibility, originality, and elaboration," *Studies in Art Education*, vol. 21, no. 3, pp. 5–11, 1980. [Online]. Available: http://www.jstor.org/stable/1319788

[24] J. P. Guilford, *The nature of human intelligence*. McGraw-Hill, 1971.

[25] E. P. Torrance, "Torrance tests of creative thinking," *Educational and psychological measurement*, 1966.

[26] M. Arefi, "Comparation of creativity dimensions (fluency, flexibility, elaboration, originality) between bilingual elementary students (azari language-kurdish language) in urmia city iran - the iafor research archive," Dec 2018. [Online]. Available: https://papers.iafor.org/submission22045/

[27] S. A. Handayani, Y. S. Rahayu, and R. Agustini, "Students' creative thinking skills in biology learning: fluency, flexibility, originality, and elaboration," *Journal of Physics: Conference Series*, vol. 1747, no. 1, p. 012040, feb 2021. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1747/1/012040

[28] F. Alacapinar, "Grade level and creativity," *Eurasian Journal of Educational Research (EJER)*, vol. 13, pp. 247–266, 01 2012.

[29] M. Arefi and N. Jalali, "Comparation of creativity dimensions (fluency, flexibility, elaboration, originality) between bilingual elementary students (azari language-kurdish language) in urmia city–iran," in *The IAFOR International Conference on Language Learning*, 2016.

[30] R. Shiffrin and M. Mitchell, Mar 2023. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.2300963120

[31] C. Stevenson, I. Smal, M. Baas, R. Grasman, and H. van der Maas, "Putting gpt-3's creativity to the (alternative uses) test," 2022.

[32] M. Binz and E. Schulz, "Using cognitive psychology to understand gpt-3," *Proceedings of the National Academy of Sciences*, vol. 120, no. 6, Feb. 2023. [Online]. Available: http://dx.doi.org/10.1073/pnas.2218523120

[33] Y. Zhao, R. Zhang, W. Li, D. Huang, J. Guo, S. Peng, Y. Hao, Y. Wen, X. Hu, Z. Du, Q. Guo, L. Li, and Y. Chen, "Assessing and understanding creativity in large language models," 2024.

[34] R. Yasaei, S.-Y. Yu, E. K. Naeini, and M. A. A. Faruque, "Gnn4ip: Graph neural network for hardware intellectual property piracy detection," 2021.

[35] "Hugging face." [Online]. Available: https://huggingface.co/codellama/CodeLlama-7b-hf

[36] "Hugging face." [Online]. Available: https://huggingface.co/codellama/CodeLlama-13b-hf

[37] "Hugging face." [Online]. Available: https://huggingface.co/shailja/fine-tuned-codegen-6B-Verilog

[38] "Hugging face." [Online]. Available: https://huggingface.co/shailja/fine-tuned-codegen-16B-Verilog

[39] "fine-tuning and api updates." [Online]. Available: https://openai.com/blog/gpt-3-5-turbo-fine-tuning-and-api-updates

[40] "fine-tuning and api updates." [Online]. Available: https://openai.com/research/gpt-4

[41] [Online]. Available: https://hdlbits.01xz.net/wiki/Main_Page

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