HDLEval Benchmarking LLMs for multiple HDLs

Farzaneh Rabiei Kashanaki Mark Zakharov, Jose Renau
Dept. of Computer Science and Engineering
University of California, Santa Cruz
Email: {frabieik, mzakharo, renau}@ucsc.edu

***Abstract*—**
Large Language Models (LLMs) are transforming code generation and documentation processes across programming languages, including hardware description languages (HDLs). However, existing benchmarks primarily focus on individual HDLs, limiting comprehensive evaluations. To address this, we introduce HDLEval, a versatile benchmarking system that evaluates LLM performance across multiple HDLs, enabling meaningful comparisons. By sharing standardized test benches, HDLEval ensures consistent performance assessments between HDLs, offering insights into language-agnostic LLM capabilities.

This system employs formal verification to verify generated code and leverages prompt engineering techniques to overcome syntactic differences between HDLs. HDLEval supports scalable evaluation, making it adaptable for current and future HDLs. Our experiments demonstrate how HDLEval can identify strengths and weaknesses in LLM-generated logic, providing a framework for improving HDL programming with LLMs.

***Index Terms*—LLM, Verilog**

I. Introduction

Recent advancements in Large Language Models (LLMs) such as OpenAI's GPT, Google's BARD/PALM, and Meta's Llama are transforming the programming landscape by enhancing code generation, documentation, and assisting newcomers in navigating complex coding frameworks. However, the integration of LLMs with Hardware Description Languages (HDLs) faces significant challenges due to the specialized syntax and limited documentation within the HDL community, hindering their broader application in hardware design.

Hardware Description Languages are crucial for designing microchips and other embedded systems but often cater to a niche community. The distinct characteristics and non-Von Neumann architecture of HDLs add layers of complexity in code generation for LLMs. Given the potential of LLMs to reduce entry barriers and streamline hardware design, adapting these models for more effective interaction with various HDLs is essential.

This work introduces **HDLEval**, a novel benchmarking system that evaluates LLMs across multiple HDLs. Unlike existing benchmarks like HumanEval [2] and VerilogEval, which are confined to popular programming languages or specific HDLs such as Verilog, HDLEval adopts a language-agnostic approach. It enables the same set of problems, formulated in plain English, to be tested across different HDLs, facilitating direct comparisons of performance and adaptability.

HDLEval not only expands the application scope of language models in hardware design but also incorporates formal verification over traditional unit tests. This change caters to the need for rigorous validation of code generated by LLMs across varied languages and platforms. By categorizing the benchmark problems into combinational and pipelined tests, HDLEval distinctly outlines the capabilities of LLMs in managing different complexities within HDLs—the combinational tests tackle straightforward logic synthesis, while the pipelined tests present challenges with sequential logic and timing-based constraints, vital for real-world applications.

Furthermore, HDLEval's adaptable architecture permits future expansions to accommodate emerging HDLs and evolving language models, ensuring its long-term utility in advancing both fields.

LLMs are typically evaluated using benchmarks like HumanEval to quantify their coding capacity. While HumanEval tests Python, the recently proposed HumanEval-X [11] extends coverage to multiple languages. However, it neglects HDLs, focusing only on popular languages. VerilogEval and RTLLM follow the HumanEval model but use Verilog tests instead of Python. HDLEval, however, extends these ideas with significant modifications to support multiple HDLs through tests written exclusively in English, enabling an equitable assessment across any HDL and dividing the problems into combinational and pipelining tests to better delineate the current limitations of LLMs with HDLs.

Moreover, HDLEval utilizes formal verification, instead of unit tests, to validate code generation, allowing for inter-language translation testing—a complex task in popular languages but facilitated in HDLs by tools that check for logic equivalence in Verilog, which all HDLs ultimately generate.

II. Related Work

HDLBits [1] is a website with problems and tests to teach students the basics of Verilog. HDLEval and VerilogEval have several tests derived from HDLBits. In HDLEval, many of the HDLEval-Human tests are from HDLBits. For generated HDLBits solutions to do a logical equivalent check versus a golden model, we either use solutions found on GitHub [10] that do not have a restrictive license or we create solutions ourselves.

HumanEval tests have the same coverage problem as VerilogEval. HDLEval uses Logic-Equivalent-Check (LEC) to formally check equivalence, but other works [3] have observed the same limitation and extended the HumanEval tests to improve coverage.

VerilogEval [4] and RTLLM [5] propose a test set to evaluate Verilog designs. VerilogEval has two sets of problems:

Human and Machine. Human consists of human-generated tests, while Machine is GPT-3.5 Turbo tests translated to English from existing Verilog code. RTLLM has a different set of problems divided into arithmetic and logic. Both use simulation for testing correctness and evaluate only Verilog code. RTLLM claims that the tests could be used for languages like Chisel, but the paper lacks basic explanations, like how to address issues with matching Chisel generated IOs.

III. HDLEval

HDLEval aims to develop a comprehensive suite of tests applicable to various HDLs, providing insights into the challenges associated with LLMs and HDLs. The authors intend to continuously release new HDLEval versions and make it available as an open-source resource.

A. Multi-HDL Tests

HDLEval distinguishes itself as an innovative, language-agnostic benchmarking framework that supports various HDLs without bias towards any specific syntax. Unlike VerilogEval, which focuses solely on Verilog, HDLEval offers a unified testing methodology that enables benchmarking of multiple HDLs, addressing challenges that arise due to syntax-specific constructs. For instance, VerilogEval includes a D latch implementation using an "always" block, a construct that is not universally supported across all HDLs. HDLEval removes this bias by providing language-neutral English problem descriptions.

To illustrate this approach further, consider a traditional VerilogEval test that requires implementing a NOR gate using Verilog syntax. The Verilog-specific problem statement typically concludes with a module declaration that specifies input and output ports directly: "module m2014_q4e (input in1, input in2, output logic out);". However, many HDLs have distinct syntax conventions, such as struct-like or interface-based input-output definitions, making VerilogEval's specific syntax problematic for non-Verilog HDLs. HDLEval circumvents this by focusing on purely English problem descriptions, ensuring compatibility with a wide range of HDLs.

To facilitate seamless input-output (IO) matching, HDLEval employs heuristics for mapping inputs and outputs across different languages. For example, a wrapper is used to ensure that single-output problems have consistent naming conventions. Furthermore, matching is done solely by name rather than relying on order, which minimizes the risk of incompatibilities. This flexibility allows HDLEval to maintain neutrality and test the LLMs' adaptability.

HDLEval also recognizes the limitations inherent in the capabilities of different HDLs. Verilog is often referred to as the "assembly language" of HDLs due to its comprehensive feature set, but other HDLs, such as DSLX, may lack specific features like arbitrary pipelining. DSLX uses actor-like abstractions, which complicate the use of traditional pipelining constructs. Hence, pipelining tests would typically fail in DSLX due to this fundamental limitation, not because of flaws in the LLM or HDL.

To address these challenges, HDLEval categorizes its tests based on functional attributes like "parameters," "combinational," and "pipelining." This ensures that only compatible tests are assigned to an LLM based on the specific HDL. Such categorization, along with input-output heuristics, makes HDLEval a comprehensive framework for evaluating the adaptability of LLMs to multiple HDLs. The aim is to extend the testing suite to accommodate more complex problems, gradually increasing test complexity and verifying functionality through logic equivalence checking (LEC).

In summary, HDLEval provides a unique and indispensable benchmarking framework that mitigates language bias and increases testing adaptability, making it suitable for assessing and improving LLMs across diverse HDLs. This focus on language neutrality and comprehensive evaluation enables LLMs to overcome language-specific challenges, learn from broader patterns, and ultimately deliver reliable HDL code generation. Further improvements could involve expanding the benchmark suite to more complex designs while maintaining the integrity of language-agnostic benchmarking.

B. Test Source

We derive HDLEval tests from three principal sources: HDLBits, custom tests, and Efabless competitions.

HDLBits serves as a pedagogical platform for introducing students to Verilog. A significant portion of our VerilogEval tests originate from this website. We selectively incorporate these tests, excluding those that are exclusively pertinent to Verilog syntax. In designing tests, we ensure compatibility across multiple hardware description languages (HDLs), including Verilog, Chisel, pyRTL, and DSLX, by maintaining uniform problem descriptions and interface definitions.

Custom tests, developed by our research group's students, address topics not encompassed by HDLBits and incorporate more complex design elements. For instance, one custom test requests is a 8-bit floating point addition unit.

Efabless competitions constitute another source of HDLEval tests. We adapt published prompts into new tests, often refining the problem through iterative question adjustments aimed at resolving specific issues. Given HDLEval's HDL-agnostic nature, we remove intermediate Verilog solutions and modify the prompts to reach an equivalent code. Though subjective, this approach facilitates the creation of tests that push the capabilities of current language models.

C. Test Classification

LLMs with specific HDL expertise generally handle combinational logic well but struggle with pipelining tests due to their complexity. Hence, HDLEval has two categories: Combinational (HDLEval-Comb) and Pipelining (HDLEval-Pipe). The former comprises combinational logic tests, while the latter includes more challenging pipelining problems, reflecting the real-world scenarios where data must be processed in stages.

For example, the shift register test involves creating a module with a synchronous reset that shifts data towards the

most significant bit, showcasing the model's ability to handle data movement across stages—a critical aspect of pipeline design. Another example is the JK flip-flop, which highlights state retention across clock cycles, an essential feature in designing complex pipelines that depend on previous states for decision-making processes.

Further illustrating the importance of pipelining in HDLs, the up-down counter demonstrates how LLMs can manage conditional operations based on multiple inputs to control a count sequence dynamically. This test challenges the models to understand and generate HDL code that not only counts but also intelligently decides when and how to count based on external controls.

The accumulator-based microcontroller example extends this by integrating an Arithmetic Logic Unit (ALU) that performs multiple operations based on opcode instructions. This setup tests the LLM's capability to handle various operations and maintain state across different stages of data processing, crucial for implementing functional pipelines in processor design.

Each of these examples serves to evaluate how well an LLM can understand, interpret, and generate HDLs that not only meet the functional requirements but also adhere to the nuanced demands of pipelined architectures.

D. Innovations and Future Directions

HDLEval represents a significant advancement in benchmarking for HDLs by establishing a framework that supports a variety of languages without being limited to any specific syntax. This approach not only facilitates broader testing capabilities but also ensures that the benchmarks are adaptable to new HDLs as they emerge in the field.

While the current scope of HDLEval primarily covers commonly used HDLs in educational and industrial contexts, we acknowledge the dynamic nature of hardware description languages and the continuous evolution of their ecosystems. Our framework is designed to be scalable and flexible, allowing for the inclusion of additional HDLs as required by future developments in the field.

Moreover, the integration of formal verification through logic-equivalence-checking (LEC) enhances the robustness of our testing methodology. Although LEC is currently optimized for less complex designs due to computational limitations, we are actively researching ways to extend this capability to support more intricate and larger-scale designs. This expansion will not only deepen the testing scenarios but will also ensure that HDLEval remains a relevant and powerful tool as both the complexity of HDLs and the capabilities of LLMs progress.

In summary, HDLEval is poised to play a pivotal role in the development of LLMs for HDL generation, providing a comprehensive, adaptable, and rigorous benchmarking tool. As we continue to refine and expand this framework, it will serve as an invaluable resource for both researchers and practitioners aiming to push the boundaries of automated hardware design.

E. Incorporation of Advanced Pipelining Tests

In addition to the aforementioned tests, HDLEval includes advanced pipelining scenarios such as a vector coprocessor with complex control logic and multiple operational stages, which challenge the LLMs to optimize data throughput and operational efficiency under varied computational loads. This not only tests the syntactical generation capabilities but also the semantic understanding of intricate hardware operations.

These pipelining tests are crucial for assessing the readiness of LLMs for real-world applications where efficiency and accuracy in sequential and parallel data processing are paramount. By systematically increasing the complexity of these tests, HDLEval aims to push the boundaries of what LLMs can achieve in the domain of HDL synthesis.

In addition to English to HDL tests, HDLEval can be used as a translation test. Each problem has an equivalent Verilog implementation. This means asking the LLM to translate the given Verilog to another HDL is possible. Then, since the HDL generates its own Verilog, we can check the correctness of the translation. Translation tests allow HDLEval to measure different characteristics of LLMs.

F. Test Evaluation

A novelty from HDLEval is the use of logic-equivalence-check (LEC) instead of a testbench. This is not a problem in HDLs because all the existing HDLs allow to generate Verilog, and existing open-source tools like Yosys [9] can perform LEC between modules. HumanEval also has this problem. Recent evalplus [3] shows that adding bugs to HumanEval is not always capture by the HumanEval tests. A LEC step is a formal equivalence that a small set of tests can not prove.

A challenge in using LEC is its requirement for convergence to affirm the equivalence of two modules. This issue did not happen in any VerilogEval and HDLEval combinational designs. However, in our examination of VerilogEval tests, we encountered an instance with pipelining where the LEC failed to complete its analysis. This occurred in a test involving a state machine designed to calculate population count.

For some pipeline tests, the LEC step can not prove equivalence or failure, but we find that this is not a problem. The reality is that LEC tries many combinations of values and none fails. From a LEC point of view, there are potential values to cover and hence it can not prove correctness. This is in-fact more powerful tests that most HumanEval assertion tests. This rare scenario occurss only once in 156 tests. In HDLEval, we consider passing the test unless LEC reports a missmatch.

IV. Setup

Table I lists all the languages used in the evaluation and the compiler versions used by this paper. When a date is provided it corresponds to the top-of-tree version at that given month.

Table II shows the LLMs used. The OpenAI account for GPTs used a Tier-5 account, which is the highest throughput. This is important to account for potential throttling.

Many LLMs, including GPT-3.5, are not deterministic. It has produced differing outcomes for the same example under

TABLE I
LANGUAGE TOOLS AND VERSIONS

Language	Tool	Version
Verilog	Yosys	0.35
Chisel	FIRRTL	3.5
pyRTL	pyRTL compiler	9/2023
DSLX	XLS	10/2023

TABLE II
LLMS USED IN THE EVALUATION

LLM	Version	Updated	Context
GPT4	gpt-4-1106-preview	4/2023	128000
GPT3	gpt-3.5-turbo-1106	9/2021	16385

identical prompt conditions. OpenAI recently proposed a new API to address this issue, providing a seed, but this solution still needs to be fully implemented across all LLMs. For fair evaluation, we avoid the deterministic settings and perform 1, 5, or 10 runs depending of the pass@k parameter.

Agents [12] iterate through LLMs to improve the performance of the LLMs. Agents leverage self-reflection, memory, and grounding. The agent used in the evaluation resembles AutoChip [6] and RTLFixer [7]. For self-reflection uses Chain-of-Thought (CoT) [8], for grounding it uses error messages like RTLFixer but uses compiler errors.

V. EVALUATION

A. HDLEval Insights

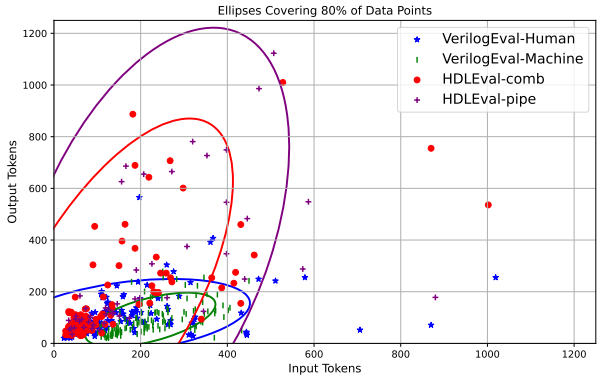

Fig. 1. HDLEval has a larger set of problems than VerilogEval.

HDLEval makes two key contributions: it allows users to use the same test across multiple HDLs and highlights the distinct behaviors of combinational and pipelined logic. It emphasizes the need for LLMs to address the challenges of HDLEval-Comb and HDLEval-Pipe.

To understand HDLEval and compare it against VerilogEval, Figure 1 compares the input and output code sizes, measured in tokens excluding comments. The plot ellipses encompass 80% of the data points for each benchmark option. Most VerilogEval outputs contain fewer than 200 tokens, with an average of around 130 output tokens, equating to approximately just 14 lines of Verilog lines of code per test. In comparison, HDLEval has an average input is comparable but the output has 27 Verilog lines of code or nearly double. HDLEval share many tests with VerilogEval because both HDLBits. The extra size in input and output mostly comes from the efabless and custom tests. This is show in the standard deviation (Std. Dev) in Table III.

TABLE III
BENCHMARKS USED IN EVALUATION

Dataset	Num. of	LoC		Avg Tokens	
	Tests	Avg	Std. Dev.	Input	Output
HDLEval-comb	138	19.1	41.6	129.8	186.6
HDLEval-pipe	53	47.6	57.6	220.9	371.5
Verilog-Human	138	14.1	7.2	172.3	102.8
Verilog-Machine	132	13.4	6.3	222.4	91.7

Table III presents several statistics for the benchmarks like the average sizes for tokens and lines of code. The table also shows an increased standard deviation for HDLEval due to the efabless tests. HDLEval exhibits larger input text specifications and output code than VerilogEval, which is expected as HDLEval incorporates the shared HDLBits [1] problems but also Efabless open-source competition questions. While HDLBits are questions to learn Verilog, Efabless tests were from chip design competitions.

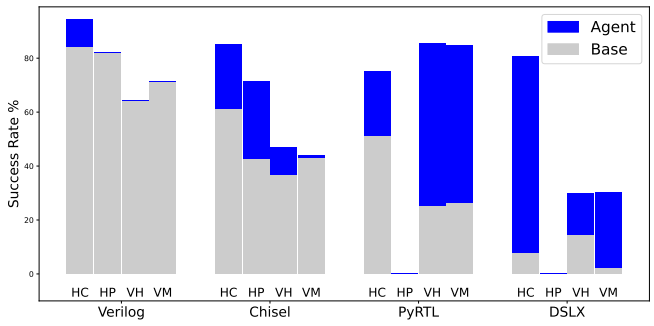

Fig. 2. HDLEval and Verilog performance across HDLs for GPT4.

Figures 2- 3 shows the HDLEval (HC and HP) can be applied to multiple HDLs while VerilogEval even generates significantly worse results for other HDLs because several questions assume Verilog Syntax creating unnecessary complications for the new HDL.

A comparative analysis of HDLEval-Comb and HDLEval-pipe reveals striking performance disparities across various HDLs, which persist across multiple LLMs. Even an advanced LLM like GPT4 only passes more than half of the HDLEval-Pipe tests with the aid of an in-house agent when targeting Chisel. This discrepancy between combinational and pipeline underscores a significant gap in current LLMs' understanding of pipelining, which is one of the key contributions of HDLEval. Figures 2- 3 illustrate that this challenge is prevalent across

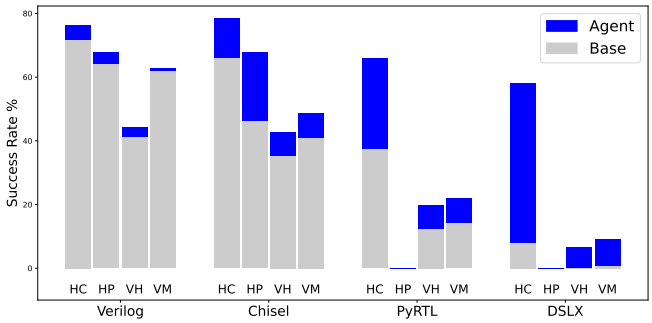

Fig. 3. HDLEval and Verilog performance across HDLs for GPT3.

all HDLs, with particularly pronounced effects in Chisel and PyRTL.

VI. Conclusions and Future Work

HDLEval proposes a new benchmark for HDLs that allows to evaluate multiple hardware description languages with the same testbench.

The benchmark is divided in two Pipelining and Combinational. There are two main reasons, not all the HDLs allow generic pipelining like Verilog, and the LLM performance is very different between combinational and pipelining. In the future, it would be interesting to create a set of "elastic" pipelining because several HDLs have this option.

Besides the new tests, HDL also proposes to use LEC instead of unit testing to capture errors this has the potential to avoid generating incorrect code and assume it correct.

Although not evaluated, HDLEval allows for additional tests like translation because it includes a correct Verilog module used for testing.

We thinkg that HDLEval is an important contribution to the community, and we plan to release it. To avoid using it for training, we plan to release it with a GPL license and to encrypt it so that web crawlers do not use it as a training data set of LLMs. The license will explicitly prohibit to release the code un-encrypted. This avoids the problem of having to continuously improve the tests because the LLMs have trained with them.

References

[1] HDLBits - Verilog Practice. website, November 2017.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

[3] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, 2023.

[4] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Verilogeval: Evaluating large language models for verilog code generation. *arXiv preprint arXiv:2309.07544*, 2023.

[5] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model, 2023.

[6] Shailja Thakur, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri. Autochip: Automating hdl generation using llm feedback, 2023.

[7] Yun-Da Tsai, Mingjie Liu, and Haoxing Ren. Rtlfixer: Automatically fixing rtl syntax errors with large language models, 2024.

[8] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[9] Clifford Wolf. Yosys Open SYnthesis Suite. https://github.com/YosysHQ/yosys, 2022. Online; accessed on December 2022.

[10] Xiaopi. HDLBits solutions. https://github.com/xiaop1/Verilog-Practice, 2023.

[11] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x, 2023.

[12] Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. Agents: An open-source framework for autonomous language agents, 2023.

Appendix

HDLEval is part of a new publicly available repository and can be accessed at https://github.com/masc-ucsc/hdeval.

To protect the integrity and intended use of the HDLEval benchmark, several measures have been implemented to prevent LLM to use the repository for training:

- **License Restriction**: The HDLEval uses General Public License (GPL). This license ensures that any use of the benchmark must comply with GPL terms, effectively restricting its use by private LLMs for training purposes.
- **Encrypted Repository**: The repository is available in an "encrypted" mode to add an extra layer of protection. The "encryption" is a gzip with uuencode which is portable but enough to avoid scrapping.
- **Usage Restrictions**: Explicit terms within the repository forbid its use for LLM training. The benchmark is strictly intended for evaluation purposes only.

To facilitate the evolution and extension of the HDLEval, a standardized JSON structure and directory format have been established.

- **JSON Structure**: The JSON files within the HDLEval repository follow a consistent schema, which includes metadata about the benchmarks, test parameters, and expected results.
- **Directory Format**: The directory layout of the HDLEval repository is organized to allow multiple benchmarks and versions.