arXiv:2405.06676v1 [cs.CL] 4 May 2024

EDA Corpus: A Large Language Model Dataset for Enhanced Interaction with OpenROAD

Bing-Yue Wu[1], Utsav Sharma[2], Sai Rahul Dhanvi Kankipati[1], Ajay Yadav[1], Bintu Kappil George[1],
Sai Ritish Guntupalli[1], Austin Rovinski[2], and Vidya A. Chhabria[1]
[1]Arizona State University; [2]New York University

Abstract—**Large language models (LLMs) serve as powerful tools for design, providing capabilities for both task automation and design assistance. Recent advancements have shown tremendous potential for facilitating LLM integration into the chip design process; however, many of these works rely on data which are not publicly available and/or not permissively licensed for use in LLM training and distribution. In this paper, we present a solution aimed at bridging this gap by introducing an open-source dataset tailored for OpenROAD, a widely adopted open-source EDA toolchain. The dataset features over 1000 data points and is structured in two formats: (i) a pairwise set comprised of question prompts with prose answers, and (ii) a pairwise set comprised of code prompts and their corresponding OpenROAD scripts. By providing this dataset, we aim to facilitate LLM-focused research within the EDA domain. The dataset is available at https://github.com/OpenROAD-Assistant/EDA-Corpus.**

I. INTRODUCTION

Chip design is a complex process which requires deep domain expertise, both in foundational knowledge and in the electronic design automation (EDA) tools used in creating chips. This expertise creates a barrier not only for newcomers to chip design, but even for experts switching to different tools or sub-domains within chip design, lowering their productivity. Recent advancements in LLMs have demonstrated tremendous potential in task automation and comprehension of esoteric topics. Works like ChatGPT [1], Copilot [2], and others have shown high performance on a wide variety of tasks; however, these LLMs are powered by training on extremely large data corpora. General-purpose LLMs are known to experience trouble extending to esoteric tasks when their training corpora lack sufficient coverage of the target domain.

Recent works on domain-specific LLMs for EDA have shown that tailored data sets and fine-tuning mechanisms can significantly improve performance over foundation models [3]–[5]. While models which relate to frontend RTL design have significant data available due to open-source languages such as Verilog, models related to backend physical design have scarcely any data available due to a heavy reliance on proprietary APIs, tools, and data.

To date, LLM methodologies in physical design EDA face significant challenges due to the absence of accessible, open infrastructure. The high cost of licenses for access to commercial EDA tools, documentation, and tutorials places them out of reach for the wider community. Moreover, these tools are bound by strict end-user license agreements which often restrict activities such as benchmarking, training AI with their documentation, and freely exchanging user scripts.

To foster research in LLM-assisted physical design, we introduce EDA Corpus, a curated dataset for physical design automation tasks. EDA Corpus is based on OpenROAD [6], a widely utilized open-source EDA tool for automated place and route tasks. Leveraging OpenROAD mitigates obstacles associated with proprietary EDA tools, enabling the public release of our dataset and facilitating its use with LLMs without licensing constraints. EDA Corpus consists of two types of data: (i) question and answer pairs, and (ii) prompt and script pairs. The question-answer dataset contains pairs of prose questions about OpenROAD and the corresponding prose answer. The prompt-script dataset contains pairs of prose requests to execute actions and the corresponding Python script which executes the actions using the OpenROAD API.

Our key contributions include the following:

- Release of a question-answer dataset to train LLMs on answering questions about physical design methods using OpenROAD. Each data point was provided by or verified as accurate by OpenROAD experts.
- Release of a prompt-script dataset to train LLMs for script generation for physical design tasks in OpenROAD. Each data point was verified through execution in OpenROAD.
- Demonstration of improvement over state-of-the-art LLMs by fine-tuning ChatGPT [1] with EDA Corpus.

To the best of our knowledge, this is the first publicly released [7], permissively licensed dataset to drive training of LLMs for physical design tasks. By releasing this dataset, we aim to seed efforts for training LLM assistants for physical design. LLMs have the capability to dramatically increase physical design accessibility for both new and seasoned chip designers alike. Furthermore, it sets an exemplar for continued research in EDA and physical design.

II. BACKGROUND

A. LLMs for Chip Design and EDA

Recently, there have been several works on using LLMs for hardware design that span high-level synthesis [8], hardware description language (HDL) generation, place and route script generation [3], and more. LLM4EDA [4] establishes a taxonomy for three classes of EDA LLMs:

- **Chatbot assistants** for learning the details of an EDA tool flow and process to design chips [5]
- **Code generation and evaluation** for generating high-quality HDL code [9]–[13] or EDA tool scripts [3]

- **HDL verification and analysis** for code summarizing and bug-finding [5], [14]–[16]

To the best of our knowledge, prior art using LLMs for physical design is extremely limited. ChatEDA [3] conducts fine-tuning using 1,500 self-generated data points; however, there is no public dataset that has been released as a part of this work. To fill this gap, EDA Corpus provides datasets for training LLMs on physical design tasks within OpenROAD. It aims to train chatbots to assist in both understanding physical design in the context of OpenROAD as well as converting user intent into physical design actions. The corpus encompasses two of the three categories listed above: ***assistant chatbot*** and ***script generation***, which are described further in Section III.

B. Relevance of OpenROAD for training LLMs

Open-source physical design platforms such as OpenROAD [6] and iEDA [17] have been widely used for education and research. OpenROAD, is an open-source platform for RTL to GDS and is composed of several different tools tightly integrated together into a single application via a database, OpenDB [18]. Unlike commercial tools, which work with Tcl interfaces, OpenROAD has Python interfaces [19] which are crucial in training LLMs for script generation.

Although OpenROAD has been widely used, it does pose challenges for first time users including installation-related issues and understanding how the underlying EDA tools work. Even though there are Python-based user interfaces which are simpler to use compared to Tcl, understanding each API and writing scripts using these APIs to query/modify the database and perform physical design tasks is challenging due to the limited documentation available and the requirement to be familiar with OpenROAD source code. We aim to overcome the above challenges by providing a dataset that can be used to train LLMs to serve as chatbots and EDA tool script generators to help users of OpenROAD improve productivity.

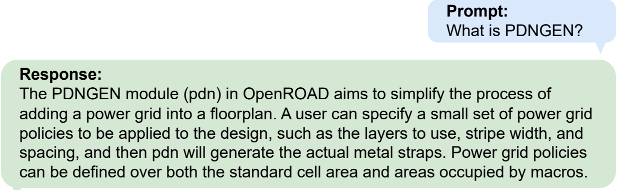
III. DATASET DESCRIPTION

This paper introduces a dataset which consists of prompt-response pair data, encompassing (i) a prose dataset which includes knowledge-based question-answer pairs, as shown in Fig. 1, and (ii) a script dataset which includes operation-based prompt-script pairs, as shown in Fig. 2.

A. Question-Answer Dataset

The question-answer dataset contains pairs of question prompts and prose answers which are collected from the OpenROAD GitHub issues, discussions, and documentation. Data from the issues and discussions were sampled from real user questions with answers provided by OpenROAD experts. Data from the documentation was created by generating questions corresponding to answers present in the OpenROAD documentation. The datapoints are categorized into three categories: OpenROAD general, OpenROAD tool, and OpenROAD flow.

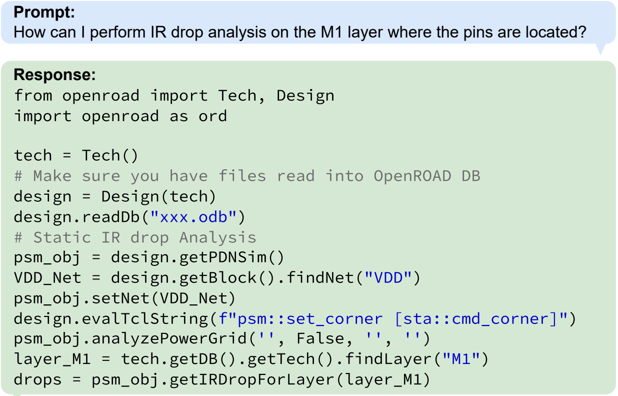
General: Questions in this category relate to general capabilities of OpenROAD, such as high-level questions, common debugging questions, or design strategy questions.



Prompt:
What is PDNGEN?

Response:
The PDNGEN module (pdn) in OpenROAD aims to simplify the process of adding a power grid into a floorplan. A user can specify a small set of power grid policies to be applied to the design, such as the layers to use, stripe width, and spacing, and then pdn will generate the actual metal straps. Power grid policies can be defined over both the standard cell area and areas occupied by macros.

Fig. 1. Sample knowledge-based question-answer datapoint.



Prompt:
How can I perform IR drop analysis on the M1 layer where the pins are located?

Response:

```
from openroad import Tech, Design
import openroad as ord

tech = Tech()
# Make sure you have files read into OpenROAD DB
design = Design(tech)
design.readDb("xxx.odb")
# Static IR drop Analysis
psm_obj = design.getPDNSim()
VDD_Net = design.getBlock().findNet("VDD")
psm_obj.setNet(VDD_Net)
design.evalTclString(f"psm::set_corner [sta::cmd_corner]")
psm_obj.analyzePowerGrid('', False, '', '')
layer_M1 = tech.getDB().getTech().findLayer("M1")
drops = psm_obj.getIRDropForLayer(layer_M1)
```

Fig. 2. Sample flow-based prompt-script datapoint for IR drop analysis.

Tool: Questions in this category relate to specific OpenROAD tools, including algorithm questions, command argument questions, and result interpretation questions.

Flow: Questions in this category relate specifically to the OpenROAD Flow, which is the user-facing RTL-to-GDS flow maintained by The OpenROAD Project.

B. Prompt-Script Dataset

The prompt-script dataset is comprised of prompts and OpenROAD Python code pairs. While Tcl is the normal interface for OpenROAD, leveraging Python allows the reuse of pretrained LLMs for Python code generation. It is worth noting that there are currently no publicly available LLMs trained specifically for generating Tcl scripts, hence the focus on Python-based scripts. Within our prompt-script dataset, data points are categorized into two main types: flow-based scripts and database (DB)-based scripts.

Flow-based: This category of our script dataset includes those datapoints that perform physical design tasks, such as floorplanning, placement, routing, and essential OpenROAD-supported procedures, e.g., reading and writing for files and database representation of the design.

DB-based: This category of our script dataset includes those data points that directly interact with the DB to either query the DB for information or modify the netlist and layout. These datapoints utilize the “get” and “set” helper functions of the DB. Examples include extracting design information, such as cell, net, and pin properties, netlist modification APIs, such as gate sizing and buffering, and incremental changes to layout, such as flipping a cell or rotating it. We further subcategorize this into query-based datapoints (which utilize several ”get” helper functions in OpenDB) and modification-based datapoints that incrementally change the database [7].

TABLE I
Question-answer and prompt-script dataset statistics

Dataset	Category	Unique	Augmented	Total
Question-answer	General	61	120	181
	Tools	64	126	190
	Flow	73	146	219
Total question-answer		198	392	590
Prompt-script	Flow	138	235	373
	Database	258	312	570
Total prompt-script		395	545	943

TABLE II
Comparison between ChatGPT3.5/4 and finetuned ChatGPT3.5

Model	Prompt-script		Question-answer		
	True	False	True	Partial	False
ChatGPT3.5	0	50	4	17	9
ChatGPT4	3	47	6	17	7
ChatGPT3.5 fine-tuned with prompt-script	22	28	–	–	–
ChatGPT3.5 fine-tuned with question-answer	–	–	26	3	1

IV. Dataset evaluation: Quality and statistics

A. *Curating the Dataset*

Question-answer dataset The most significant challenge in creating this dataset, and one of our major contributions, is curating this data to filter out noise. Many questions and answers in the source data (i) do not have an answer clearly marked on GitHub, (ii) were resolved through OpenROAD bugfixes (and therefore are not relevant), or (iii) contain interspersed irrelevant conversation. The question-answer pairs in this dataset were curated to provide concise questions with direct, generalized answers and remove spurious data pairs. Prior LLM works have shown that even small amounts of high-quality training data can produce high-quality models through fine-tuning foundation models [20]. Given the low availability of question-answer data for EDA, providing a curated, high-quality dataset is essential to facilitate future research.

Table I identifies the distribution of data points based on the categories mentioned in Section III. The dataset features 198 unique data points as question-answer pairs. In addition, this dataset is supplemented by augmented data pairs formed through paraphrasing questions and answers. Between the unique and augmented data pairs, the dataset comprises nearly 600 data points.

Prompt-script dataset Table I presents the distribution of datapoints across categories in our prompt-script dataset. The table distinguishes between unique-functioned datapoints and augmented datapoints. While each datapoint is unique, in the augmented dataset some datapoints may perform similar functions with parameter variations in the script. For instance, the augmented has few instances of sizing a gate and the gate sized can be different between datapoints. Although these datapoints are distinct, they are categorized as "augmented" due to their similar functionalities but differing script parameters. Two methods are applied to enrich the existing data and categorize them as "Augmented" in Table I:

1) ***Paraphrasing prompts***: The work by Chang et al. [10] indicates that different prompts will affect the quality of the generated code by LLMs. Therefore, we have paraphrased prompts to link the same script to multiple semantic meanings to mitigate such effects.
2) ***Variable and parameter changes***: The balance of datapoints across the different categories will highly affect the training process of data-driven algorithms. To keep a balanced dataset where each unique function has a sufficient representation, we augment the dataset by changing variable names in the script and input parameters.

Some of the flow-based data are generated using automated scripting techniques. For instance, different contextual prompts are generated for each physical design stage, and domain experts generate different settings and implementations of each physical design stage. With permuting different stages in the flow, unique physical design flow scripts are created. We also use the same augmentation techniques to enrich the flow-based dataset. The DB query and design modification datasets are created using a specific design as an example and are based on manually designed prompts and corresponding scripts. These prompts and scripts are created by six individuals, inherently creating different semantic representations of the prompts based on individual preference and encapsulating different scripting styles of each individual

B. *Validating the Dataset*

Question-answer dataset To ensure the quality of the collected data, data samples are only taken from responses provided by OpenROAD experts, validated as correct by OpenROAD experts, or sampled from official OpenROAD documentation. Further, prompts and responses were proofread and adjusted to correct grammar, although some prompts were not adjusted to represent realistic user prompts better.

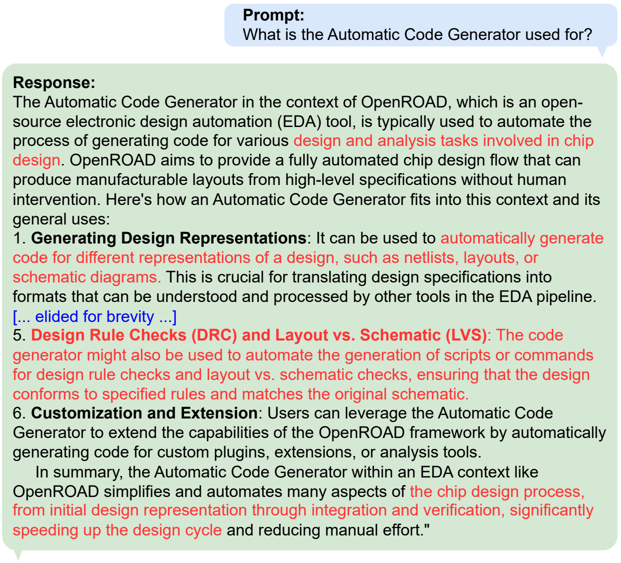
Prompt-script dataset Each datapoint in the prompt-script dataset undergoes validation by a domain expert, who executes the script within OpenROAD and evaluates the correctness of the output against the specified prompt. This validation process mirrors the methodology employed in unit testing. Given that the prompts are manually crafted and the scripts are tailored for each prompt within the OpenROAD application, every datapoint is inherently correct by construction.

V. Impact of the Corpus

To demonstrate the impact of the corpus, we run a simple experiment by comparing different versions of ChatGPT, shown in Table II. For each of the datasets, we separate the dataset into 95% training and 5% validation. Then, we use the training set to fine-tune ChatGPT3.5 using the "auto" fine-tuning settings. Lastly, we evaluate each model with the validation set to determine the performance on each task. We perform fine-tuning and validation separately for each dataset.

A. *Question-Answer Dataset*

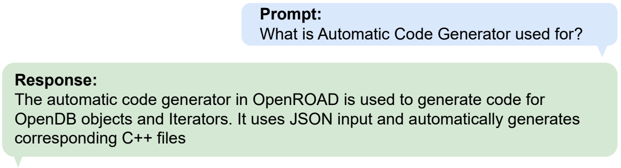
For the question-answer dataset, the responses are marked as true if the response contains only factual information and



Prompt:
What is the Automatic Code Generator used for?

Response:
The Automatic Code Generator in the context of OpenROAD, which is an open-source electronic design automation (EDA) tool, is typically used to automate the process of generating code for various design and analysis tasks involved in chip design. OpenROAD aims to provide a fully automated chip design flow that can produce manufacturable layouts from high-level specifications without human intervention. Here's how an Automatic Code Generator fits into this context and its general uses:
1. **Generating Design Representations**: It can be used to automatically generate code for different representations of a design, such as netlists, layouts, or schematic diagrams. This is crucial for translating design specifications into formats that can be understood and processed by other tools in the EDA pipeline.
[... elided for brevity ...]
5. **Design Rule Checks (DRC) and Layout vs. Schematic (LVS)**: The code generator might also be used to automate the generation of scripts or commands for design rule checks and layout vs. schematic checks, ensuring that the design conforms to specified rules and matches the original schematic.
6. **Customization and Extension**: Users can leverage the Automatic Code Generator to extend the capabilities of the OpenROAD framework by automatically generating code for custom plugins, extensions, or analysis tools.
In summary, the Automatic Code Generator within an EDA context like OpenROAD simplifies and automates many aspects of the chip design process, from initial design representation through integration and verification, significantly speeding up the design cycle and reducing manual effort."

Fig. 3. ChatGPT4 for question answering. Red text is factually incorrect.



Prompt:
What is Automatic Code Generator used for?

Response:
The automatic code generator in OpenROAD is used to generate code for OpenDB objects and Iterators. It uses JSON input and automatically generates corresponding C++ files

Fig. 4. Example using fine-tuned ChatGPT3.5 for question answering.

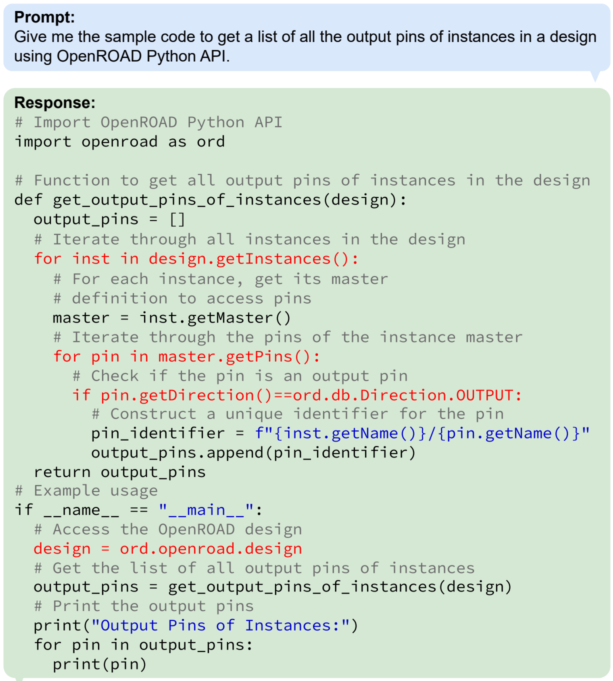
answers the question, partially true if the response answers the question correctly but also contains false information, and false if the response does not answer the question or provides only false information. These answers are judged by a consensus of OpenROAD experts. Fig. 3 shows a False example where ChatGPT4 produces a response with factually incorrect information present and fails to answer the question correctly. Fig. 4 shows an example of a True response where the question is answered and contains factually correct information.

Table II demonstrates that on the question-answer task, ChatGPT3.5/4 does not do well. However, it is able to provide many partially correct answers by relying on general EDA knowledge from the foundation model. The fine-tuned ChatGPT3.5 performs much better by offering concise answers and answering the question directly. In this case, the experiment demonstrates that while foundation models can provide general answers, fine-tuning with the dataset can improve EDA-specific answers.

B. Prompt-Script Dataset

For the prompt-script dataset, the responses are marked as true only if the model generates a script that returns the correct result when executed in OpenROAD. All other responses are marked as false. Fig. 5 shows a False example where ChatGPT4 produces code with incorrect syntax or hallucinated APIs (red lines). Fig. 6 shows a True example from a fine-tuned ChatGPT3.5 model where the generated script produces the correct output when executed in OpenROAD.

On the prompt-script task, ChatGPT3.5/4 do exceedingly poorly on the task with almost no correct responses. After fine-tuning, however, the accuracy increases to nearly half as



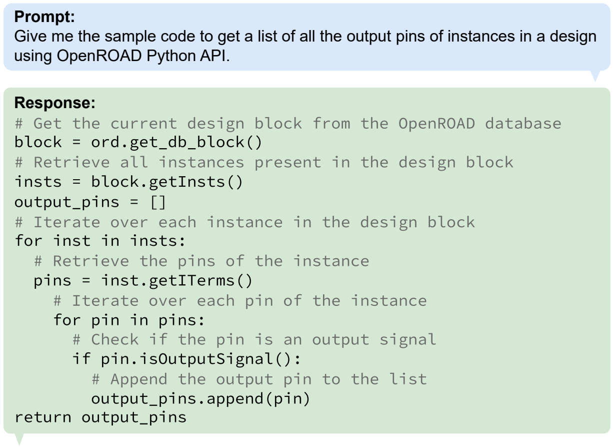
Prompt:
Give me the sample code to get a list of all the output pins of instances in a design using OpenROAD Python API.

Response:

```
# Import OpenROAD Python API
import openroad as ord

# Function to get all output pins of instances in the design
def get_output_pins_of_instances(design):
  output_pins = []
  # Iterate through all instances in the design
  for inst in design.getInstances():
    # For each instance, get its master
    # definition to access pins
    master = inst.getMaster()
    # Iterate through the pins of the instance master
    for pin in master.getPins():
      # Check if the pin is an output pin
      if pin.getDirection()==ord.db.Direction.OUTPUT:
        # Construct a unique identifier for the pin
        pin_identifier = f"{inst.getName()}/{pin.getName()}"
        output_pins.append(pin_identifier)
  return output_pins
# Example usage
if __name__ == "__main__":
  # Access the OpenROAD design
  design = ord.openroad.design
  # Get the list of all output pins of instances
  output_pins = get_output_pins_of_instances(design)
  # Print the output pins
  print("Output Pins of Instances:")
  for pin in output_pins:
    print(pin)
```

Fig. 5. Example using ChatGPT4 for script generation. Red lines are incorrect.



Prompt:
Give me the sample code to get a list of all the output pins of instances in a design using OpenROAD Python API.

Response:

```
# Get the current design block from the OpenROAD database
block = ord.get_db_block()
# Retrieve all instances present in the design block
insts = block.getInsts()
output_pins = []
# Iterate over each instance in the design block
for inst in insts:
  # Retrieve the pins of the instance
  pins = inst.getITerms()
    # Iterate over each pin of the instance
    for pin in pins:
      # Check if the pin is an output signal
      if pin.isOutputSignal():
        # Append the output pin to the list
        output_pins.append(pin)
return output_pins
```

Fig. 6. Example using fine-tuned ChatGPT3.5 for script generation.

shown in Table II. This result demonstrates that the prompt-script dataset can greatly improve performance on EDA script generation tasks. However, further development and research are warranted to improve accuracy.

VI. CONCLUSION

This paper introduces EDA Corpus, a pioneering open-source dataset designed to facilitate the integration of LLMs into physical design through the OpenROAD framework. By offering over a thousand data points across question-answer and prompt-script formats, EDA Corpus marks a significant step towards filling the gap in domain-specific data for LLM training, promoting research in LLM-assisted physical design. Our evaluation shows that fine-tuning LLMs with EDA Corpus leads to improved performance on physical design-specific tasks (script generation or question and answer), highlighting the critical role of tailored datasets.

REFERENCES

[1] "ChatGPT." https://openai.com/chatgpt, 2023.

[2] "GitHub Copilot." https://github.com/features/copilot, 2021.

[3] H. Wu, *et al.*, "ChatEDA: A Large Language Model Powered Autonomous Agent for EDA," *IEEE T. Comput. Aid. D.*, 2024.

[4] R. Zhong, *et al.*, "LLM4EDA: Emerging Progress in Large Language Models for Electronic Design Automation," *arXiv preprint arXiv:2401.12224*, 2023.

[5] M. Liu, *et al.*, "ChipNeMo: Domain-Adapted LLMs for Chip Design," *arXiv preprint arXiv:2311.00176*, 2024.

[6] T. Ajayi, *et al.*, "INVITED: Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project," in *Proc. DAC*, pp. 1–4, 2019.

[7] "EDA Corpus." https://github.com/OpenROAD-Assistant/EDA-Corpus.

[8] L. J. Wan, *et al.*, "Software/Hardware Co-design for LLM and Its Application for Design Verification," in *Proc. ASP-DAC*, 2024.

[9] K. Thorat, *et al.*, "Advanced Large Language Model (LLM)-Driven Verilog Development: Enhancing Power, Performance, and Area Optimization in Code Synthesis," *arXiv preprint arXiv:2312.01022*, 2024.

[10] K. Chang, *et al.*, "ChipGPT: How Far Are We from Natural Language Hardware Design," *arXiv preprint arXiv:2305.14019*, 2023.

[11] S. Thakur, *et al.*, "AutoChip: Automating HDL Generation Using LLM Feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[12] M. Liu, *et al.*, "VerilogEval: Evaluating Large Language Models for Verilog Code Generation," in *Proc. ICCAD*, 2023.

[13] Y. Fu, *et al.*, "GPT4AIGChip: Towards Next-Generation AI Accelerator Design Automation via Large Language Models," in *Proc. ICCAD*, IEEE, 2023.

[14] Y. Tsai, *et al.*, "RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models," *arXiv preprint arXiv:2311.16543*, 2024.

[15] M. Orenes-Vera, *et al.*, "From RTL to SVA: LLM-assisted Generation of Formal Verification Testbenches," *arXiv preprint arXiv:2309.09437*, 2023.

[16] X. Meng, *et al.*, "Unlocking Hardware Security Assurance: The Potential of LLMs," *arXiv preprint arXiv:2308.11042*, 2023.

[17] X. Li, *et al.*, "iEDA: An Open-Source Intelligent Physical Implementation Toolkit and Library," *arXiv preprint arXiv:2308.01857*, 2023.

[18] "OpenDB." https://github.com/The-OpenROAD-Project/OpenROAD/tree/master/src/odb, 2023.

[19] V. A. Chhabria, *et al.*, "OpenROAD and CircuitOps: Infrastructure for ML EDA Research and Education," in *Proc. VTS*, 2024.

[20] C. Zhou, *et al.*, "Lima: Less is More for Alignment," *Advances in Neural Information Processing Systems*, vol. 36, 2024.