(Special Session)

OpenROAD and CircuitOps: Infrastructure for ML EDA Research and Education

Vidya A. Chhabria[1], Wenjing Jiang[2], Andrew B. Kahng[3], Rongjian Liang[4],
Haoxing Ren[4], Sachin S. Sapatnekar[2], and *Bing-Yue Wu[1]
[1]Arizona State University; [2]University of Minnesota; [3]University of California, San Diego; [4]NVIDIA Corporation

***Abstract*—Traditional electronic design automation (EDA) techniques struggle to fulfill the stringent efficiency and quick turnaround demands of complex integrated systems. Machine learning (ML) strategies for EDA ("ML EDA") are pivotal in transforming EDA to address these challenges. However, they encounter significant obstacles due to inadequate infrastructure, ranging from datasets to software interfaces. This paper demonstrates a software infrastructure for ML EDA built on two key technologies: (i) OpenROAD's Python APIs, and (ii) NVIDIA's CircuitOps, an EDA data representation format tailored for ML, facilitating ML EDA applications. The paper illustrates three ML EDA examples that utilize the established OpenROAD and CircuitOps infrastructure.**

I. Introduction

Traditional electronic design automation (EDA) methods rely on numerical algorithms and discrete optimization. However, they are increasingly inadequate for developing future systems that face tight market deadlines and must meet rigorous performance and power requirements. The adoption of machine learning (ML) in EDA is poised to revolutionize this field by significantly accelerating EDA processes [1]–[3]. By enabling faster ML-powered analyses, it becomes feasible to handle more complex systems and implement more intricate, possibly ML-driven, optimizations. This results in not only better circuit performance but also reduced time frames for design processes. The integration of ML into EDA, known as "ML EDA," is experiencing rapid growth. By combining ML techniques with domain expertise, it is possible to extract and reuse knowledge from data with unmatched efficiency. This trend is evident in both academia and industry, with a significant portion of research papers at major conferences focusing on ML-based EDA approaches; leading EDA tool providers are quickly developing and releasing new ML-powered tools for design, verification, and testing, exemplified by products such as Cerebrus and DSO.ai.

However, despite its promise, ML EDA faces tremendous barriers to adoption due to limited infrastructure. An ideal infrastructure will include access to EDA tools, PDKs, datasets, trained ML models, and software libraries that enable easy integration between existing EDA tools and ML frameworks. Today, we are far from such an infrastructure due to IP-related challenges and the closed culture of the chip design industry. There have been several efforts in the past that are trying to develop such an infrastructure; these can be categorized into initiatives for creating datasets [4]–[7], data representation formats for ML EDA [8], [9], open-source EDA tool flows and tools, a concept of a one-stop-shop for ML EDA applications [10], and several individual open-source ML EDA algorithms [11], [12]. While these are crucial components of an infrastructure, there has been no prior work that develops a software infrastructure to enable integration between ML frameworks and EDA tools.

In this paper, we demonstrate an ML EDA software infrastructure that builds on two key technologies: (i) OpenROAD's Python interpreter [13] for physical design and (ii) NVIDIA's CircuitOps [14], [15] for representing physical design data in an ML-friendly format. Our infrastructure enables a new research platform for EDA researchers, with three key elements as shown in Fig. 1.

1) Python APIs in OpenROAD that wrap the underlying C++ APIs of EDA engines, to enable faster data generation compared to using commercial tools' Tcl interfaces.
2) An ML-ready data format, CircuitOps, from NVIDIA that leverages OpenROAD to model chip data as labeled property graphs, and pandas data frames.
3) Additional Python APIs in OpenROAD that integrate ML inference results back into EDA tools, giving a feedback path from ML algorithms into the OpenROAD platform.

Our paper illustrates three ML EDA applications using the infrastructure. The first leverages the Python interpreter to train a model for IR drop prediction ***within*** OpenROAD and perform inference for IR drop ***within*** OpenROAD. The second leverages the Python interpreter to train a graph convolutional network (GCN), using a reinforcement learning (RL) framework for logic gate sizing ***within*** OpenROAD. These two ML EDA applications demonstrate a true "ML in EDA tool" framework. The third application uses timing prediction as an example to demonstrate how the CircuitOps data representation format enables easy data collection. The example applications presented in this paper using the proposed infrastructure are available on GitHub [16].

II. OpenROAD Python Interface

Although the EDA industry provides the chip design community with sophisticated tools, these tools must be operated using a complex, low-bandwidth tool command language

*Primary author.

979-8-3503-6378-4/24$31.00 ©2024 IEEE

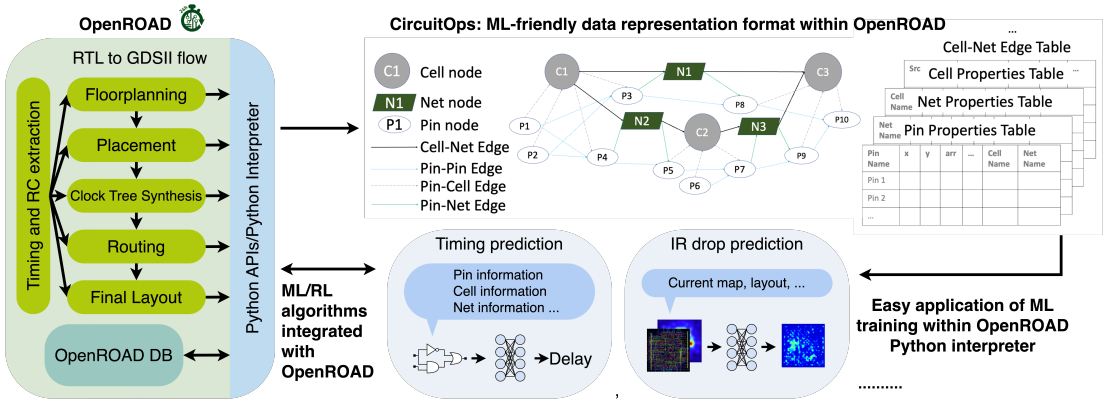

Fig. 1. A software infrastructure for the integration of ML frameworks and EDA tools.

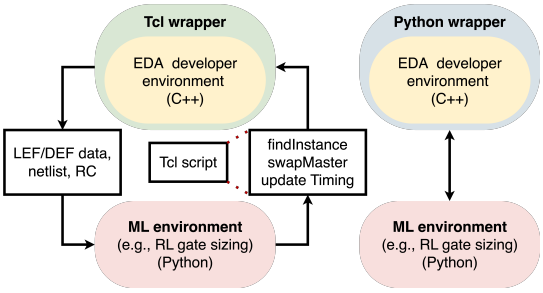

Fig. 2. Interaction between an ML environment and an EDA tool with Tcl APIs (left) and Python APIs (right).

TABLE I
RUNTIME COMPARISONS BETWEEN PYTHON AND TCL APIS FOR DATA EXCHANGE BETWEEN ML AND EDA TOOL ENVIRONMENTS.

Design	#cells	#nets	Runtime (s)	
			Python APIs	Tcl w/ file IO
aes	30,202	17,812	12.6	31.5
bp_be	141,468	58,464	26.4	103.6
bp_fe	96,150	36,388	17.5	62.9

(Tcl). A significant drawback of Tcl is its limited selection of data structures. For ML EDA applications, Tcl limits (i) the efficiency of collecting large training datasets from EDA tools due to scalability issues and (ii) the ability of ML algorithms to interact with an EDA tool due to the lack of flexibility to communicate with ML environments (Python).

Fig. 2 shows the interaction of EDA tools with ML environments using Tcl (left) and Python APIs (right). The Tcl-based flow relies on file I/Os as the primary means of data exchange between ML environments and EDA tools, making any iterative loop between ML frameworks and existing EDA tools prohibitively slow. Table I lists the runtime of a framework that exchanges data between the ML environment for three small benchmarks. The Tcl API-based flow is slow and does not scale to a larger number of iterations and larger testcases.

Our work highlights the Python APIs around OpenROAD that enable easy interaction between the OpenROAD database and ML environments. These APIs enable feature extraction and label annotation into the database (DB) without leaving the OpenROAD environment. We illustrate two categories of APIs (flow APIs and DB query APIs) and highlight ML EDA applications that these APIs enable.

Flow APIs enable the execution of different stages of the physical design flow through a Python shell. Examples include performing floorplan, placement, and routing. For instance, Listings 1 and 2 show code snippets that respectively read in traditional EDA tool files and perform floorplanning. The flow API capability streamlines the development of ML EDA applications, particularly those that predict information of the next physical design stage at the current stage [17], [18].

```
import openroad as ord
from openroad import Tech, Design
# Read files
tech = Tech()
tech.readLiberty("lib_file")
tech.readLef("tech_file")
tech.readLef("lef_file")
design = Design(tech)
design.readVerilog("verilog_file")
design.link("design_top_level_name")
```

Listing 1. File reading through OpenROAD Python API.

```
floorplan = design.getFloorplan()
# Set the floorplan utilization to 50%
floorplan_utilization = 50
# Set the aspect ratio of the design (height/width) as 0.5
floorplan_aspect_ratio = 0.5
# Set the spacing between die and core as 10 microns
floorplan_core_spacing = [design.micronToDBU(10) for i in
  range(4)]
floorplan.initFloorplan(floorplan_utilization,
  floorplan_aspect_ratio, floorplan_core_spacing[0],
  floorplan_core_spacing[1], floorplan_core_spacing[2],
  floorplan_core_spacing[3])
floorplan.makeTracks()
# Place IO pins
params = design.getIOPlacer().getParameters()
params.setRandSeed(42)
params.setMinDistanceInTracks(False)
params.setMinDistance(design.micronToDBU(0))
params.setCornerAvoidance(design.micronToDBU(0))
# Place the pins on M7 and M8
design.getIOPlacer().addHorLayer(
  design.getTech().getDB().getTech().findLayer("M7"))
design.getIOPlacer().addVerLayer(
  design.getTech().getDB().getTech().findLayer("M8"))
IOPlacer_random_mode = True
design.getIOPlacer().run(IOPlacer_random_mode)
```

Listing 2. Floorplanning using OpenROAD Python API.

DB Query APIs act as helpers to interact with OpenDB (OpenROAD's database). Therefore, they play a crucial role in feature extraction for ML EDA applications and label annotation. Listings 3 and 4 illustrate Python APIs to extract cell, net and pin-related timing properties from the database. The ability to query such properties directly within a Python

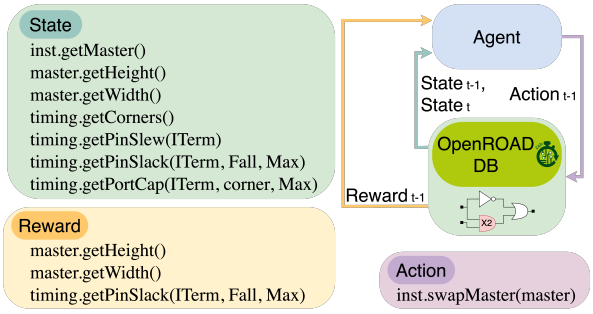

Fig. 3. RL-based gate sizing enabled via OpenROAD Python APIs.

shell, enables easy feature extraction for both training and inference of several ML EDA algorithms. Using these APIs, we demonstrate two example ML EDA applications. The first uses image-like data for IR drop prediction using convolutional neural networks, and the second uses graphs and graph convolutional neural networks trained using reinforcement learning (RL) for logic gate sizing.

```
block = design.getBlock()
corner = timing.getCorners()[0]
insts = block.getInsts()
for inst in insts:
  inst_static_power = timing.staticPower(inst, corner)
  inst_dynamic_power = timing.dynamicPower(inst, corner)
  inst_name = inst.getName()
  libcell_name = inst.getMaster().getName()
  inst_x0 = inst.getBBox().xMin()
  inst_y0 = inst.getBBox().yMin()
  inst_x1 = inst.getBBox().xMax()
  inst_y1 = inst.getBBox().yMax()
nets = block.getNets()
for net in nets:
  pin_and_wire_cap = timing.getNetCap(net, corner,
    timing.Max)
  net_name = net.getName()
  net_type = net.getSigType()
```

Listing 3. Querying cell and net properties using OpenROAD Python API.

```
for inst in insts:
  inst_ITerms = inst.getITerms()
  for pin in inst_ITerms:
    if design.isInSupply(pin):
      continue
    pin_name = design.getITermName(pin)
    pin_rise_arrival_time = timing.getPinArrival(pin,
     timing.Rise)
    pin_fall_arrival_time = timing.getPinArrival(pin,
     timing.Fall)
    pin_rise_slack = timing.getPinSlack(pin, timing.Fall,
      timing.Max)
    pin_fall_slack = timing.getPinSlack(pin, timing.Rise,
      timing.Max)
    pin_slew = timing.getPinSlew(pin)
```

Listing 4. Querying pin properties using OpenROAD Python API.

```
db = ord.get_db()
inst = block.findInst("buffer_1")
target_master = db.findMaster("BUF_X1")
inst.swapMaster(target_master)
```

Listing 5. Performing gate sizing using OpenROAD Python API.

(1) IR drop prediction A large body of work has used ML for IR drop prediction [11], [12], [19], [20]. These works map the IR drop prediction problem into an image-based ML task. We demonstrate how our APIs enable IR drop prediction using these techniques in [16]. We use the Python APIs to query instance locations and instance power to create power maps. Similarly, we create golden IR drop maps. The power maps serve as features, and the IR drop maps serve as labels. These maps are available in the same Python shell as the EDA database, eliminating the need for file I/Os. Similarly, we can perform ML inference within the same Python shell, enabling a true "ML inside EDA tool" framework.

(2) RL-based logic gate sizing [21], [22] A typical RL framework involves an iterative flow where an agent from a particular environment state explores a solution space by performing actions, and estimates a reward for an action performed from that state. The action results in the agent transitioning from one state to a next state. In the context of logic gate sizing, the state is the current set of logic gate sizes within the netlist, the action is a change in the size of a particular logic gate, and the reward is the reduction in the weighted sum of the slack, power, and area [21]. Without the Python APIs, a framework such as this would require the iterative exchange of data (action, state, and reward) between the Tcl APIs and the RL environment in Python via file I/Os (as shown in Fig. 2). However, with the enablement of the Python APIs in OpenROAD, we can train the agent within the same Python shell of the EDA tool, allowing incremental timing updates for reward calculation, updates to the database for the action, and feature extraction for the state and next state transition. The APIs used for the RL framework are shown in Fig. 3. Listing 5 shows how the swapMaster Python API can modify the netlist for gate sizing.

III. CircuitOps

One of the critical bottlenecks for ML EDA development is the lack of publicly available ML-friendly datasets due to intellectual property concerns. Additionally, current EDA data collection methods face several challenges. (i) Most EDA tools require knowledge of the underlying data structure to interact with them, either in Python or Tcl, which creates a barrier to entry for non-expert users. (ii) Querying EDA information requires iterative looping (even with the Python APIs described in Section II), resulting in low parallelizability, extensive runtimes and poor scalability. (iii) Many tailor-made datasets must be extracted from the EDA platform to facilitate ML EDA research across a range of problems in EDA. Each dataset building requires custom scripts for data extraction, resulting in engineering overhead. However, some of the EDA information is shared across different ML EDA problems, e.g., the location of cell placement is used in a number of image-based ML EDA applications.

CircuitOps [14] serves as a common data representation format for ML EDA. It provides a low user barrier, as users can access the EDA information through CircuitOps using popular Python-based ML packages such as pandas, NumPy and PyTorch. This helps bypass the need for EDA tool knowledge or the Python APIs to operate EDA platforms such as OpenROAD. These ML packages further enable parallelization capability for ML EDA dataset generation. Furthermore, users can manipulate the metadata using popular Python-based graph libraries such as graph_tool and DGL. The above-

mentioned Python-based libraries are already widely used in ML EDA applications, making CircuitOps compatible with ML EDA development. CircuitOps streamlines the engineering effort required to build datasets for various ML EDA applications by storing the metadata for VLSI designs in a graph-based data representation format called labeled property graphs (LPGs), where each node in the graph is backed by an intermediate representation (IR) table entry (Fig. 1). LPGs and IR tables are the two main components of CircuitOps.

(1) LPG is a graph-based representation format that stores the relationships between nodes, i.e., pins and cells, and edges.

(2) IR tables store the corresponding node properties presented in LPG, i.e., library cell information, instance information, pin timing information, net information, etc.

The generic CircuitOps data representation format enables easy dataset generation for a variety of ML EDA applications: by applying different filters to the graph, different sub-datasets can be instantly generated. These filtering operations and data queries are performed by in-built Python libraries. We demonstrate CircuitOps using the work in [17] for a timing prediction problem. Listing 6 shows an example of building a dataset for this problem from an underlying LPG and IR table instance. The listing distills the properties and generates a dataset consisting of only pins, i.e., pin slack, pin rising arrival time, and pin falling arrival time, and only preserving the edges between pins by applying the filter using graph_tool. This is completely free of any EDA tool knowledge as the data is stored in general graph format (an instance of graph_tool) and the features are queried using graph_tool APIs.

The graph-based data representation provided by CircuitOps perfectly matches the use cases for netlist-based ML EDA applications, such as ML algorithms for gate-sizing and buffering. For geometry-based ML EDA applications, the parallelizability provided by CircuitOps increases the efficiency of dataset building as well. OpenROAD has enabled CircuitOps to convert EDA file data into the CircuitOps data representation format with the scripts and flows available in [16].

```
from graph_tool.all import *
# Generate graph
g = Graph()
# Add vertices to the graph
# Vertices include pins, cells, and nets
g.add_vertex("#_vertices_in_the_design")
v_type = g.new_vp("int")
v_type.a[0:"#_pin"] = 0 # pin
v_type.a["#_pin":"#_pin"+"#_cell"] = 1 # cell
v_type.a["#_pin"+"#_cell":] = 2 # net
# Add edges to graph
# edge_df is the pandas.DataFrame format of the
    Intermediate Representation (IR) Table
edge_df["e_type"] = 0 # pin_pin
edge_df.loc["#_pin_pin" : "#_pin_pin"+"#_cell_pin", ["
    e_type"]] = 1 # cell_pin
edge_df.loc["#_pin_pin"+"#_cell_pin" : "#_pin_pin"+"#
    _cell_pin"+"#_net_pin", ["e_type"]] = 2 # net_pin
edge_df.loc["#_pin_pin"+"#_cell_pin"+"#_net_pin" : "#
    _pin_pin"+"#_cell_pin"+"#_net_pin"+"#_net_cell", ["
    e_type"]] = 3 # net_cell
edge_df.loc["#_pin_pin"+"#_cell_pin"+"#_net_pin"+"#
    _net_cell" : "#_pin_pin"+"#_cell_pin"+"#_net_pin"+"#
    _net_cell"+"#_cell_cell", ["e_type"]] = 4 # cell_cell
e_type = g.new_ep("int")
g.add_edge_list(edge_df.values.tolist(), eprops=[e_type])
# Add pin features to the graph
# pin_df is the pin properties IR table
v_slack = g.new_vp("float")
v_risearr = g.new_vp("float")
v_fallarr = g.new_vp("float")
v_slack.a[0:"#_pin"] = pin_df["slack"].to_numpy()
v_risearr.a[0:"#_pin"] = pin_df["risearr"].to_numpy()
v_fallarr.a[0:"#_pin"] = pin_df["fallarr"].to_numpy()

# Generate pin-pin graph ###
g_pin = GraphView(g, vfilt=(v_type.a==0), efilt=(e_type.a
    ==0))
```

Listing 6. Distilling pin properties (slack and arrival times) from the LPG.

IV. Conclusion

Our work showcases an ML EDA infrastructure employing OpenROAD Python APIs and the CircuitOps data representation format. It highlights how these technologies facilitate the integration of ML frameworks with EDA tools via Python APIs and simplify dataset generation for ML EDA applications through CircuitOps. The example ML EDA applications that use this infrastructure have been demonstrated at ASP-DAC 2024 as a tutorial and are accessible to the community in [16].

References

[1] G. Huang, *et al.*, "Machine Learning for Electronic Design Automation: A Survey," *ACM TODAES*, vol. 26, no. 5, pp. 1–46, 2021.

[2] H. Ren and J. Hu, *Machine Learning Applications in Electronic Design Automation*. Cham, Switzerland: Springer, 2022.

[3] B.-Y. Wu, *et al.*, "SpeedER: A Supervised Encoder-Decoder Driven Engine for Effective Resistance Estimation of Power Delivery Networks," in *Proc. MLCAD*, 2022.

[4] J. Jung, *et al.*, "METRICS2.1 and Flow Tuning in the IEEE CEDA Robust Design Flow and OpenROAD ICCAD Special Session Paper," in *Proc. ICCAD*, 2021.

[5] Z. Chai, *et al.*, "CircuitNet: An Open-Source Dataset for Machine Learning in VLSI CAD Applications with Improved Domain-Specific Evaluation Metric and Learning Strategies," *IEEE TCAD*, vol. 42, no. 12, pp. 5034–5047, 2023.

[6] V. A. Chhabria, *et al.*, "BeGAN: Power Grid Benchmark Generation Using a Process-portable GAN-based Methodology," in *Proc. ICCAD*, 2021.

[7] "VerilogGeneration." https://github.com/shailja-thakur/VGen, 2024.

[8] P. Shrestha and I. Savidis, "EDA-ML: Graph Representation Learning Framework for Digital IC Design Automation," in *Proc. ISQED*, 2024.

[9] P. Shrestha, *et al.*, "EDA-schema: A Graph Datamodel Schema and Open Dataset for Digital Design Automation," in *Proc. GLSVLSI*, 2024.

[10] "SLICE." https://slice-ml-eda.github.io/, 2024.

[11] V. A. Chhabria, *et al.*, "Thermal and IR Drop Analysis Using Convolutional Encoder-Decoder Networks," in *Proc. ASP-DAC*, 2021.

[12] V. A. Chhabria, *et al.*, "MAVIREC: ML-aided Vectored IR-drop Estimation and Classification," in *Proc. DATE*, 2021.

[13] "OpenROAD." https://github.com/The-OpenROAD-Project/OpenROAD, 2022.

[14] R. Liang, *et al.*, "CircuitOps: An ML Infrastructure Enabling Generative AI for VLSI Circuit Optimization," in *Proc. ICCAD*, 2023.

[15] "CircuitOps." https://github.com/NVlabs/CircuitOps, 2022.

[16] "ASP-DAC24-Tutorial." https://github.com/ASU-VDA-Lab/ASP-DAC24-Tutorial, 2024.

[17] E. C. Barboza, *et al.*, "Machine Learning-Based Pre-Routing Timing Prediction with Reduced Pessimism," in *Proc. DAC*, 2019.

[18] V. A. Chhabria, *et al.*, "A Machine Learning Approach to Improving Timing Consistency between Global Route and Detailed Route," *ACM TODAES*, vol. 29, no. 1, 2023.

[19] G. S. P. Kadagala and V. A. Chhabria, "2023 ICCAD CAD Contest Problem C: Static IR Drop Estimation Using Machine Learning," in *Proc. ICCAD*, 2023.

[20] Y. Zhong and M. D. F. Wong, "Fast Algorithms for IR drop Analysis in Large Power Grid," in *Proc. ICCAD*, 2005.

[21] V. A. Chhabria, *et al.*, "IR-Aware ECO Timing Optimization Using Reinforcement Learning," in *arXiv preprint arXiv:2402.07781*, 2024.

[22] Y.-C. Lu, *et al.*, "RL-Sizer: VLSI Gate Sizing for Timing Optimization using Deep Reinforcement Learning," in *Proc. DAC*, 2021.