A Collaborative Data Model for AI/ML in EDA

August 19, 2020

Kerim Kalafala, IBM
Veeravanallur Parthasarathy, AMD
Norman Chang, ANSYS
Akhilesh Kumar, ANSYS
Elias Fallon, Cadence Design Systems
Sriram Madhavan, GLOBALFOUNDRIES
Prateek Bhansali, Intel Corporation
Srinivas Bodapati, Intel Corporation
Chandramouli Kashyap, Intel Corporation
James Masters, Intel Corporation
Ramy Iskander, Intento Design
Larg Weiland, PDF Solutions
Karthik Aadithya, Sandia National Laboratory
Boon-Siang Cheah, Synopsys
Mengdi He, Synopsys
Leigh Anne Clevenger, Si2

Published by
Silicon Integration Initiative, Inc. (Si2™)
12335 Hymeadow Dr, Suite 450
Austin, TX 78750

A Collaborative Data Model for AI/ML in EDA

***Abstract*—A standard, common method for classification and structure of machine learning training and inference data for interoperability is critical to enable and accelerate the use of artificial intelligence and machine learning in semiconductor electronic design automation. Subject matter experts from across the semiconductor and EDA industry highlight the differences and common threads in developing industry standards for AI/ML in EDA application data for design areas including digital, analog, shapes-based and IP development. The authors conclude that in order to accelerate AI/ML applications for EDA, a collaborative and coordinated approach is needed. A prerequisite for this approach is establishing the best process for organizing, leveraging and sharing data. Si2 industry survey results show a gap in the availability and organization for AI/ML data in EDA. A common data model would address the data organization gap for chip developers, EDA tool developers, IP providers and researchers by first supporting the high interest EDA areas, design data and derived data.**

Keywords— artificial intelligence, classification, EDA, machine learning, standards

I. Introduction

What is needed to enable and accelerate the use of artificial intelligence and machine learning in electronic design automation? Subject matter experts from across the semiconductor and EDA industries describe in this work that a standard, common method for classification and structure of ML training and inference data for EDA interoperability is critical. Their views on design areas including digital, analog, shapes-based and IP development highlight the differences and common threads in developing industry standards for AI/ML in EDA application data.

In the following sections, this work provides background and motivation for machine learning and IC design, describes EDA design data for ML, defines derived data, discusses IP protection for ML data, presents modes of inferencing models, discusses use case applications for a common data model, and draws conclusions on AI/ML in EDA data model opportunities.

II. Background and Motivation

Integrated Circuit (IC) design is a complex process, during which billions of nanoscale transistor devices are fabricated on a silicon die and connected via intricate metal layers. The final product is an IC which powers much of our life today.

Given the intricacy of modern-day ICs, as shown in Fig. 1, Electronic Design Automation (EDA) plays a critical role in the design process. Even in the presence of EDA tools, however, a single IC can take months or years to design. Accelerating this timeline would reduce cost and time to market, benefitting all industry stakeholders. Machine Learning (ML) and Artificial Intelligence (AI) may be the solution. Important to the motivation of this study are Machine Learning and IC Design, the Demand for Data, and the Structure of the Data Model.

A. Machine Learning and IC Design

Before going further, it is important to understand the kinds of problems ML is intended to solve. AI/ML is best suited for applications where a great deal of data exists for training and evaluation, patterns for leveraging past experiences are common, and the problem either cannot be solved deterministically using physical principles or is exponentially complex. The first criterion is easily satisfied for many problems in IC design. Data from EDA tools is plentiful in the forms of placement, routing, netlists, characterized libraries, simulation outputs, masks, layout and extraction files. One can leverage this data, and the patterns embedded within, to generate new data using ML [1]. For example, the topology of the design, hierarchy in the design, the symmetry in layouts, repeating patterns in waveforms, etc. The second criterion is also satisfied for many problems in EDA design. While some issues can be mathematically solved using physical principles—such as circuit simulation, which combines device physics equations with numerical techniques to solve Differential Algebraic Equations (DAEs)—there are many applications in which NP-hard problems abound and heuristics are used to solve them. Examples of the latter include finding the optimal routing in a design given an initial placement of logic gates or transistors [2], [3], determining the optimal setting of parameters for logic and physical synthesis, analog circuit optimization and transistor sizing [4], etc. ML can also be applied to the compute infrastructure and scheduling for solving CAD problems, such as identifying the correct memory [5] and core configuration of machine to launch a CAD job in the cloud.

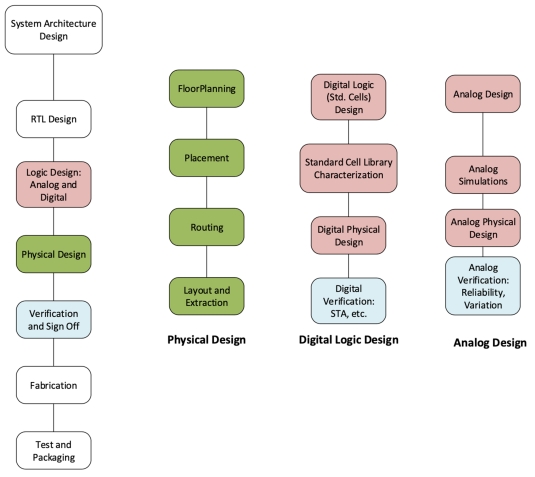


Fig. 1. **IC Design Cycle**

While there is ample scope for application of AI/ML to EDA, current efforts have been scattered across academia, research labs, and industry. To accelerate AI/ML applications to EDA, a collaborative and coordinated approach is needed. A prerequisite for this approach is establishing the best process for organizing, leveraging and sharing data, since such data are the foundation on which AI/ML applications are built. This paper proposes a data model to facilitate this interaction. The data model must satisfy the needs of the various persons who will be interacting with it, including CAD engineers (both EDA companies and CAD groups within design houses), design engineers, researchers, and academics. For example, a researcher would use the data model to develop an optimal ML algorithm, whereas a design engineer may test the performance of an ML algorithm on the data model, and a CAD engineer may want to create a product that encapsulates various ML algorithms and use the data model to benchmark and qualify. The data model must support each of these use cases.

B. Demand for Data

EDA and semiconductor stakeholders completed an industry-wide Si2 survey from April 15-May 15, 2020. Two hundred respondents shared their success, areas of interest, and roadblocks in AI/ML for EDA. The goal of the Special Interest Group is to use this information to drive industry and research direction, filling the development, standards and interoperability gaps to enable greater adoption of AI/ML in EDA. The survey respondents highlighted three areas for an AI/ML in EDA methodology flow: design and derived data, data organization, and a reference flow with associated Application Programming Interface (API). This section focuses on the interdependence of requirements for design data, derived data, and data organization for AI/ML in EDA training and inference.

Reviewing all the responses for the importance of training data, availability and a common data model is not unlike analyzing the effectiveness of online training videos and the value of search engines. From the millions of videos online, a user wants the top responses to given search criteria. Finding the video is what users want. They do not want to know the details of the search engine. In the same way, survey respondents put training data availability at a higher importance than a common data model (Fig. 2), but the quality of the data model will determine how useful the data returned by the model will be. Viewed together, ML training data availability and a common data model are valuable for AI/ML in EDA adoption.

Survey questions also addressed concerns about the lack of training data and a common data model. This is an established survey technique to check the consistency of results by asking the negative of a previous question. Interestingly, a different set of respondents expressed concerns about lack of training data, but the results for lack of a common data model were consistent (Fig. 2)

Respondents indicated the areas in which they are testing AI/ML Methods. There was a wide range of interest, but over 25% of respondents indicated simulation, place and route, compute efficiency (performance), and verification and debug were priorities (Fig. 3). These would be the areas most compelling to users for the first implementation of a common data model.

Respondents were asked to identify the types of design data they are using or would use for ML training and inference. Their answers provide a starting point for the forms of design data and derived data to be supported by a common data model. Data relating to simulation, layout, place and route, timing, power, verification, design rules, and standard cells were important to over 25% of respondents (Fig. 3).

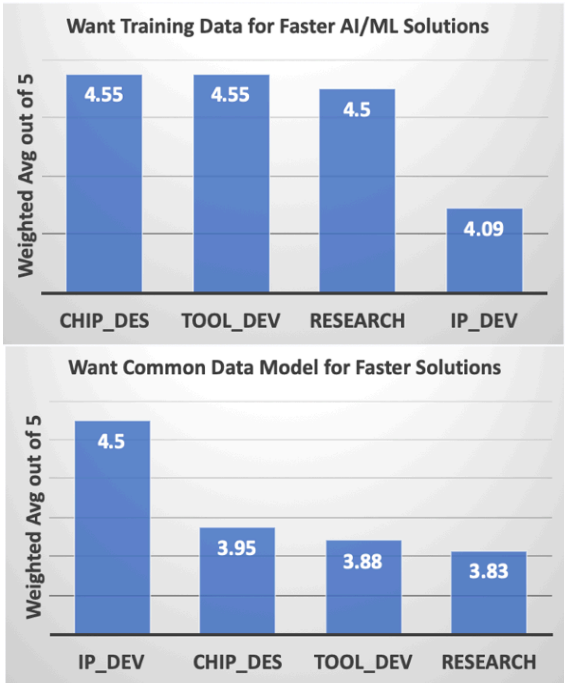


Fig. 2. Importance of training data availability and a common data model

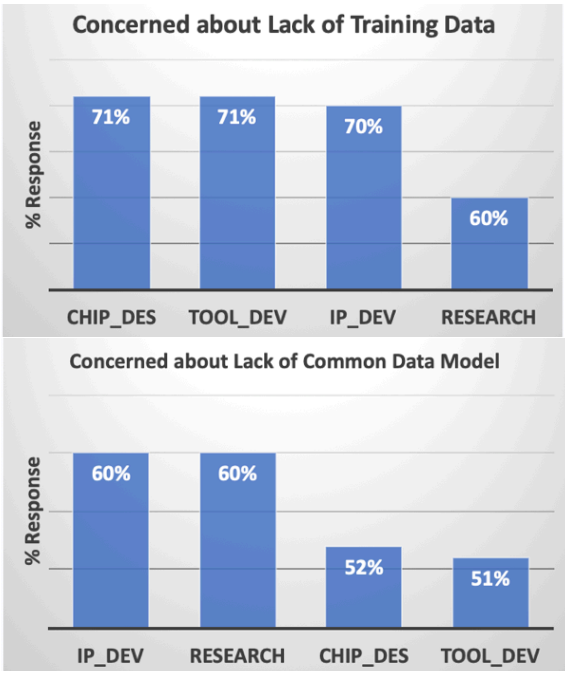


Fig. 3. Concern about lack of training data availability and a common data model by area

From these industry survey results, there is a gap in the availability and organization for AI/ML data in EDA. A common data model would address the data organization gap for chip developers, EDA tool developers, IP providers and researchers by first supporting the high interest EDA areas, design data and derived data. Without such a data model, the industry faces:

- Lower design productivity
- Lesser design quality
- Poorer design insight
- Increased design fabrication respins
- Higher costs overall
- Reduced market competitiveness
- More difficult reuse, migration and modification processes
- No capitalization on expertise
- No design resources improvement, including analog experts and EDA tools

We look at the requirements of the data model next.

C. Structure of the Data Model

A data model must be able to represent the design at various abstraction levels, from architecture to layout level. An ideal data model for EDA applications is organized around the following components.

1) Objects: These are design objects such as registers (RTL phase), gates (digital design), devices (analog design), polygons (floor planning, placement, layout), etc. Each object must have appropriate attributes which describe the object being considered. For instance, attributes for transistors could be the device type, the dimensions, the number of fins, etc., while the attributes for layout objects may be the layer, the color, the dimensions, and so on.

2) Composability and inheritance: The objects may be composed of other objects—for example, an ALU will be made of gates. Similarly, inheritance must be supported using "is-a" relationship (i.e., a transistor is a device, as is a capacitor).

3) Relationships: The different objects in a design are related to each other, either through a physical or an abstract notion of connectivity. Examples of the former include the interconnect between gates and transistors, and the wiring between polygons. An abstract relationship could represent the transactions between two architectural objects. A relationship may also describe special kinds of objects, each with their own attributes.

4) Operations: We may wish to perform an operation on objects and relationships, such as computing the timing of an arc on a gate object. Similarly, we may wish to propagate waveforms through relationships. A related concept is storing the results of an operation on the objects and relationships.

5) Design versus derived data: Whereas the "objects" described above represent design data such as registers, devices, wires, and shapes, "derived" data represents simulation results calculated with respect to design data. Examples of "derived" data include results of noise, power, logical verification analysis, and static timing analysis. One way to distinguish design from derived data is that design data represents the physical implementation which will be manufactured as an integrated circuit (this includes higher level representations such as a "netlist," which is composed of a set of circuits and wires, each of which is then represented as a set of shapes). On the other hand, derived data represents the analysis that determines whether a given implementation will meet all of its design requirements.

6) APIs: We need to provide APIs to get object attributes, define custom attributes, perform computations, get results etc. Such an API layer should ideally be compatible with existing big-data and machine learning frameworks such as TensorFlow, Pandas, etc.

7) Obfuscation: For IP-sensitive attributes on objects and results of computations, a layer must be provided to hide the true values within the data model and fetch the modified ones for the user. This layer needs to work with the API layer defined above. Applications could include obfuscation of the device-related attributes or the true delays of a timing arc.

8) Revision History: A data model must not only represent all design data but also be able to access the data of intermittent design stages. Therefore, the API needs to interact with various revision control systems used in EDA. There should be a standard header on every data set. Since AI/ML is itself an interactive process, it is beneficial to also track the model creation process and all related data to enable data analytics like benchmarking and quality metric monitoring over time.

III. Design Data

Design data for analog and digital design can be a fundamental part of an AI/ML training and inference dataset. Described here are the relationships between circuit design data generated by EDA tools and the potential new uses of this data for AI/ML training and inference for improved design and performance. These include a unified data model, data model classes, and a definition of a data model for analog circuits.

A. Unified Data Model: Digital Example

Physical design of digital ICs is a highly iterative process. Between each iteration, questions are often asked, and analysis performed to determine what happened, why it happened, and how to improve the design [6]. Therefore, we need a system which allows us to answer questions at the intersection of logic, placement, wiring, timing, power, and noise, among other variables. Designers also cannot be restricted to asking questions during the implementation flow, meaning a tool-agnostic data model to query data offline through an efficient API layer is sorely needed.

As shown in Fig. 4, existing models such as Si2 OpenAccess are well suited for representing design data. What is still needed, however, is an agreed-upon standard for efficient storage of derived data such as timing, noise, power in an offline data model, as well as a means to link derived data back to the design data objects. This superset of design data and derived data, all tightly coupled and linked, gives rise to a unified persistent data model enabling AI/ML in EDA.

As illustrated in Fig. 5, such a tightly coupled data model requires that derived data such as timing, power, and noise are treated as first class citizens (as opposed to generic properties annotated on an underlying design data representation).

Furthermore, to enable tool/flow correlation, we need the ability to extract both design and derived data (denoted "DD" in Fig. 6) at multiple points in the implementation flow. This approach requires a highly compact unified data model including both design and derived data, which can be written quickly with minimal overhead to the implementation flow.

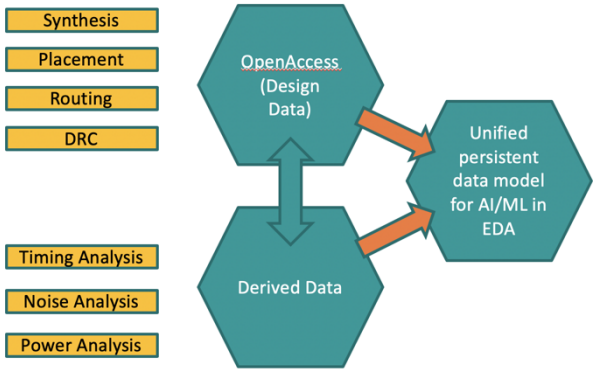


Fig. 4. High-level representation of a Unified Persistent Data Model for enabling AI/ML in EDA.[1]

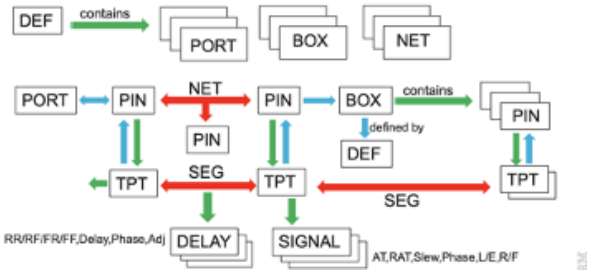


Fig. 5. Derived data such as timing as a first class citizen of a Unified Persistent Data Model.[1]

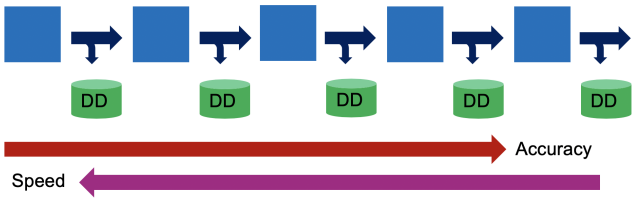


Fig. 6. Extracting design and derived data from multiple points along the implementation flow (courtesy Nathaniel Hieter, IBM).

In addition to the iterative nature of digital design closure, physical synthesis, place and route, etc. tools typically contain hundreds of options which control various behaviors. Therefore, to interrogate data effectively, it is important for a common data model to represent the various tool parameters that were used to arrive at a particular design point. For example, in the space of ML applied to EDA, one may be interested in building an inferencing engine to predict an optimal set of tool parameters based on various input features (e.g., number of logic elements, target frequency and power, available area, etc.). To train such an inferencing engine, a large amount of labeled data would likely be required, where the labels include various tool settings [7].

Modern digital design also involves closing to requirements across a varied process, voltage, temperature space. Many techniques have been developed over the years to efficiently analyze design data across process-voltage-temperature (PVT) points, including statistical timing (SSTA) and multi-mode-multi-corner (MMMC) analysis. It is therefore an important feature of a common data model to be able to represent analysis of design data across multiple PVT combinations. Such a representation will aid in the development of ML applications in EDA including engines which may perform inferencing of design characteristics across PVT space based upon a select number of discrete measurements (reducing the need for full simulation in all PVT corners, replacing this with inferencing engines capable of predicting performance at non-simulated corners) [8].

B. Data Model Classes: Analog Example

Analog design comprises two distinct phases: pre-layout and post-layout, with layout/extraction separating them as shown in Fig. 7. Each phase involves certain steps which perform analysis and transformations on the underlying analog block and generates a lot of data. Although shown sequential, in practice a lot of iteration happens between difference phases.

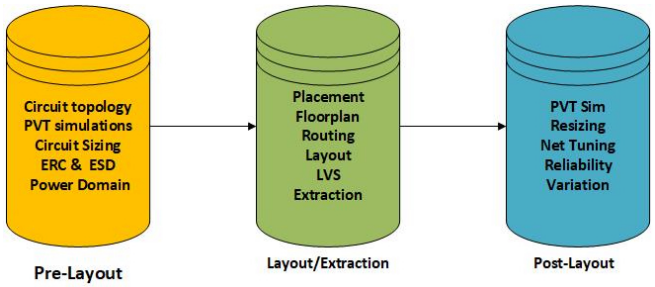


Fig. 7. Typical Analog Design Cycle: Process-Voltage-Temperature (PVT), Electrical Rule Check (ERC), Electro-Static Discharge (ESD), Layout Versus Schematic (LVS)

To capture design data from the various stages of a typical analog design cycle, we propose to construct a data model with the following components. Each class specified below should have a well-defined API to perform data queries, manipulations, and transformations.

1) Hypergraph class to represent analog circuits: In the hypergraph class, circuit connection points are represented via nodes, and devices that make up circuits are represented via

hyperedges, or ordered sequences of nodes. This method is general enough to capture any analog circuit topology. Fig. 8 shows an example.

2) Hyperedge class: The hyperedge class can be used for devices and interconnections between devices; whenever two devices share a connection point, their respective hyperedges will share a node. When hyperedges are used for storing devices, contextual attributes like associated branch currents, reliability characteristics, number of fins, geometrical location in placement/layout, and contact resistances can be added. When hyperedges are used for storing interconnection between devices, they can capture the number of pins, routing layer, pin locations, etc.

3) Node class: The hypergraph node class is used to capture simulation data such as voltage waveforms represented by time-value pairs and initial conditions used to carry out circuit simulations.

4) Device macromodel class: This class captures the electrical behavior of analog devices such as resistors, capacitors, diodes, and transistors. In analog circuit design, it is vitally important to model both the DC and transient behavior of such devices. Typical device model formulations use "current" functions as well as "charge" functions for modelling such behaviors. These functions can be modelled using constructs such as Directed Acyclic Graphs, or Sequence Of Computation-type data structures. For reference, APIs representing devices and their core functions can be found in academia and industry; for example, the ModSpec API used in the MAPP platform developed at UC Berkeley, the Spyce and Xyce device APIs developed at Sandia National Labs, and the language constructs used in Verilog A.

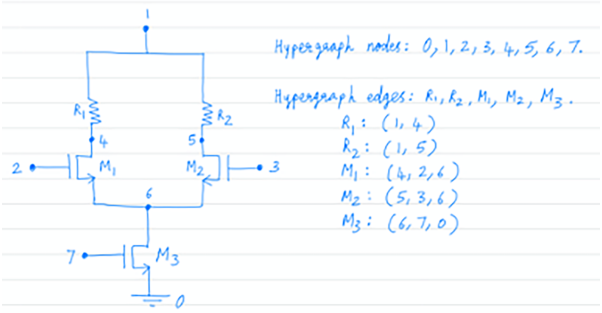


Fig. 8. An example of a hypergraph class representing analog circuits

5) Circuit macromodel class: This class allows modeling, storage, and querying of analog circuit functionality. That is, given input conditions (PVT corners, load conditions, input waveforms, analysis type, etc.), one can obtain the circuit's outputs (output waveforms or designated scalar measurements such as power drawn) by issuing appropriate queries to instances of this macromodel class.

6) Computational resource class: This class facilitates the prediction of computational resources required for process steps such as simulation, extraction, placement, and routing. APIs for this class should allow capturing circuit characteristics such as device count, type of elements, the "complexity" of the underlying device, circuit, and sub-circuit macromodels, circuit simulation conditions and parameters (for example, the minimum time-step for transient simulation or the number of harmonics for Harmonic Balance simulation), metal layers (for routing), track information (for routing), area constraints (for placement), etc., all of which will be used to predict the computational resources (runtime and memory) required for circuit analyses and simulations.

7) Continuous waveform class: Analog circuits and simulations predominantly feature continuous waveforms; these waveforms can be time-domain or frequency-domain, real-valued or complex-valued, and scalar-valued or vector-valued. We need a class to represent these waveforms, and also to intelligently query such waveforms using on-the-fly transformations such as smoothing, interpolation, and extrapolation. Such queries should support both normal lookups (what is the waveform value at t=1ns?) and inverse lookups (at what time does the waveform cross 1V?).

8) Classes for statistical data and distributions: As mentioned above, manufacturing variability is a key factor in analog circuit design. We believe a comprehensive data model should include features to represent statistical variables and quantities associated with variability analysis and randomly distributed parameters. These include correlated and uncorrelated random variables, probability density functions and mass functions, regression analysis, etc.

C. Definition of data model for Analog Circuits to serve ML/AI in EDA

Analog design experts seek to develop structured design methodologies that provide:

- Physics-based design
- Capacity to deal with complex circuits
- Connection between hand analysis and simulation
- Sufficient design insights
- Performance trade-offs exploration
- Analog design assistance

At present, three type of analog intellectual properties exist: Soft IP focused on AMS behavioral simulation, Hard IP focused on layout design and migration, and Firm IP focused on connectivity representation

Traditional EDA tools are commercially available for Soft and Hard IPs. No efficient tools for Firm IP have been commercially available and widely accepted by the analog design community. Firm IP is defined as the network of interconnections of devices such as MOS transistors, resistors, capacitors, inductors and diodes. It was introduced in the 1980s as the standard input format for Simulation Program with Integrated Circuit Emphasis (SPICE) simulators. Neither its data format nor its usage model have evolved in the decades since. Designs for Firm IP generally rely on human tacit knowledge (expertise, ideas, heuristics, etc.) and may lack a

clear understanding as opposed to formal design knowledge. We strongly believe that Firm IP is the least exploited data format by analog EDA community [9], [10].

One of the objectives of this white paper is to provide a clear definition of a data model for analog circuits in order to transform them into a seamless and standard form for efficient ML/AI for analog EDA such as reuse, migration, and many other applications. One particular step is to define a clear bridge between tacit knowledge and formal knowledge, and determine how to store it in a common data model on top of an Si2 OpenAccess database. Once a clear data model of analog circuits is provided, algorithms for ML/AI can be implemented more efficiently using the OA-based design flow.

IV. Derived Data

A design flow usually involves several EDA tools generating a great deal of data at each step, some of which is redundant or duplicative data. AI/ML-based applications in EDA may not need all of the data generated at each step of the design flow. Additionally, some design data may be proprietary to the specific EDA vendor or the design house; therefore, it is important for the data model to support mechanisms for extracting derived data from the existing design data and/or analysis data. These mechanisms can involve processes including Data Relevance, Data Cleaning, Data Transformation, Enabling Extraction of Relevant Data, and whether data is Above vs. Below the Line.

A. Data Relevance

1) Identification: Designing any ML application involves identifying the relevant data for generating the ML model. The data model should allow extraction of the relevant data from a large set of designs and analysis data. This can be achieved through a set of APIs supported by the data model. For example, an EDA tool may generate log files with tons of data which may be proprietary; however, the desired ML model may only need certain parts of the information from the log files. The data model APIs can enable efficient extraction of such information while not revealing any information about the design and particular algorithms used in the EDA tools. Each EDA tool can define the API for querying the derived data so it could be used for ML-based applications along with the derived data from other tools in the design flow. Identification should also be enabled across multiple SoC designs and IPs. For example if a given IP is designed on multiple technology nodes or even multiple revisions of a given technology node, one should be able to map/extract the relevant input features/outputs of that IP at a given design abstraction level using the data model/API. This would enable learning across multiple designs. One should be able to query multiple versions of RTL for a given IP and map them to the features implemented in the IP. The data model should support multiple tags/mappings for a given design/IP that uniquely identify that piece of data along with its features/derived data and design abstractions of various known states of the design. We need the ability to connect and identify data across multiple designs.

2) Relationships: It is often important to preserve the relationship between the derived data and its source, and the data model should allow such relationships to be defined. An example would be maximum voltage on a circuit node and the corresponding voltage waveform, where $Vmax_i = Max(Wv_i(t))$, where $Vmax_i$ is the maximum voltage on the circuit node i and $Wv_i(t)$ is the voltage waveform of the node i. The relationship here is defined by $Max()$. To enable ML across design abstractions (e.g., RTL/schematic/post-layout/extracted views of the design), it is important that relationships between such abstractions be captured. For example, for a given schematic view the ability to map the extracted view would enable ML across such design abstractions. In physical synthesis flows (APR flows), it is important to access data from various APR phases to build predictive models across design phases. Similarly, analog simulation results from schematic views could be used to accelerate post-layout extracted simulations.

3) Redundancy Removal: The derived data can be composed from more than one design or analysis. There are scenarios where derived data can have redundancies. These redundancies will not yield any new information for the ML application and hence should be removed. The data model can support mechanisms for removing redundancies in the derived data. For instance, if there is a simple relationship between the input and output, we need not store the output as derived data and store only the input and the corresponding transfer function. For example, if we know the voltage and current across a pair of nodes, the power dissipated in the device need not be stored even though the EDA tool explicitly generates the power data.

4) Balanced Sampling: With large amounts of data, sometimes with billions of data points, it may be necessary to sample relevant data from huge data sets. This scenario can be very common in modern chip design where the number of elements can be extremely large and consequently the analysis and design data generated from EDA tools will be huge; hence, it will be computationally very expensive to train the ML model with all of these data points. The data model should support APIs to allow balanced sampling of the large data set to generate a representative reduced data set with high fidelity. The balanced sampling is important so that the derived data is not skewed.

B. Data Cleaning

1) Outliers: The data generated from EDA tools will invariably have outliers, and the data model should be able to identify, document, and remove these outliers to the greatest extent possible. This will ease the burden on the ML app developer to identify and remove the outliers manually.

2) Missing Data: While extracting relevant data from a design or analysis data set, there may be missing values for certain features. The data model should be able to clearly identify which features have missing values, and notify the user if necessary.

3) Handling Duplicates: The data model should have the capability to identify and remove duplicates from derived data.

Note that duplicates can occur if two or more designs or analyses data map to the same derived data.

4) Grouping: Reducing the number of distinct features by grouping can make ML model development easier. For example, in a design, power/signal nets can be grouped together based on common characteristics, instead of each net representing a distinct feature. Composite data can be derived from the grouped feature, and the data model should have mechanisms to allow users to perform such operations.

C. Data Transformation

The design and analysis data from EDA tools should have a large variety of features such as shapes, names, values, and distributions. It is rarely feasible to use raw data directly on machine learning models; therefore, the data model should support data transformations. In some cases, the data transformations may also enable hiding of proprietary information. Many data transformations are possible, such as:

- Scaling
- Encoding categorical data
- Bias removal
- Skewed data handling
- Statistical techniques such as PCA or SVD for feature reduction
- Discretization
- Data range checking for anomaly detection
- Data normalization
- Data clustering and compaction through unsupervised learning

Derived data is not essential to build IP, but is vital to understanding the behavior of IP. Data derived from peripheral verification flows (i.e., reliability, SIPI, timing, etc.), are critical to the IP development process.

D. Enabling Extraction of Relevant Data

There are various types of relevant data in the IP development process. Each type is extracted by its corresponding EDA tools and bounded by industry standard formats. Examples include the SPF file for (parasitic) extracted netlists and IBIS.

E. Above vs. Below the Line

We must decide whether to store every waveform shape, or only a high-level summary (e.g., TNS, worst noise violation, total power), or something in between. Data from safety-related fields (i.e., ISO 26262, automotive, biomedical) should maintain stricter archiving and distribution practices.

V. IP Protection

Semiconductor design is generally known to be the art of approximation of the physical world. This is highly defined by the designer expertise, best practices and heuristics learned over several years of experience.

This process of innovation requires a thorough design model, design flow and design representation capable of capturing design intents along with design data and secure the overall knowledge in a hierarchical fashion. For instance:

- The designer may share partially to totally the design steps of an analog circuit along with its design data.
- The design team manager may decide to share or hide some of the steps and data shared by the designer
- The organization may protect its design portfolio against any unexpected leave.
- The design process should be complete and independent of the designer. It should be fully reproducible
- The IP protection should recognize designer's contributions through a patented portfolio and compensation.

Therefore, a complete design flow based on efficient design and data models is crucial for the IP Protection and Management of Innovation (disruptive or Breakthrough) within a corporate organization. We define three levels of sensitivity for obfuscation of sensitive IP, each with different handling guidelines.

1) Shareable data: The least sensitive data, such as concepts, high-level model architectures, widgets or connectors that would benefit fellow EDA and circuit designers. This level does not contain any company proprietary, or foundry-specific information.

2) Partially sharable data: This data is derived from proprietary design or flows, can be modified or enhanced, but cannot be reverse engineered. Examples include hyper parameters and timing libraries. Decryption keys for this data can be provided conditionally.

3) Confidential data: Data in this level include details of proprietary designs or flows, such as architecture diagrams or schematic netlists. This data is encrypted, and can be shared if mandated by the flow. All confidential data should only be executable, and not readable.

VI. Inferencing Models

Machine Learning models are generated in the first place with the intent of using them as *predictions* in a downstream application or process or a solution. This is formally known as model *inferencing,* and interchangeably used as model *evaluation* or model *query* or model *polling*.

During the inferencing step, the user will supply a fixed set of constant parameters as inputs to the model and get back the predicted/processed quantity of the interested as output. For example, if an ML model has been constructed *a priori* to predict run-times of different design blocks based on inputs of block area, wirelength, number of flops, ram count etc., the user in a new block instance will put in the fixed values and get back the new predicted run time along with probability. To support specific query types an API is defined and constructed that will work on an ML model since most models are complex in nature.

Inferencing has different modes of use in practice:

1) Offline or stand alone. In this mode, a model with its defined inference API is queried offline, and subsequently the *predicted* values are used separately in a traditional or existing workflow. This allows for sanity checking prior to the use of the predicted values in the downstream step.

2) Online or Integrated. In this mode, the API is embedded as a replacement for traditional or deterministic metric of that said parameter. This allows for seamless automation of use of ML models. This mode is also used in *Streamed* applications where *on the fly* decisions are made.

Challenges in *Inferencing* exist if the application using the predictions is implemented on the Edge or part of an Edge-AI implementation as in IoT applications or if each inferencing query is time consuming or if the number of users for a given inference API are large in number. As models get refreshed due to staleness, some APIs also get redefined. Further, a single model might be used for different purposes at different stages of design or workflow. There are mitigation strategies in place based on the problem in hand and the model hosting choices or end-user needs.

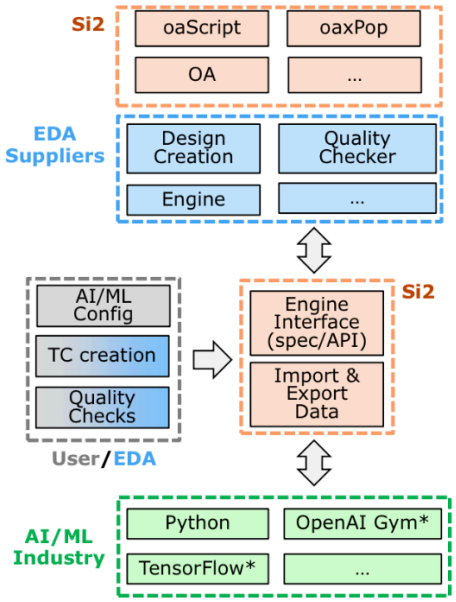
In order to handle AI specific techniques such as inferencing, AI/ML in EDA solutions need the expertise of the best data scientists. However, data scientists are not trained in EDA – they are trained to work with business, medical, video, and web data. The design data and derived data models should be developed assuming the data scientists are looking at that type of data for the first time. A data scientist is looking for consistent labeling, a familiar format such as comma separated values or jpg, and compatibility with the Open Neural Network Exchange (ONNX) format. This allows a data scientist to train ML models using tools including Python ML libraries, TensorFlow, or university research tools. The data scientist doing the work will not need to be a circuit design and analysis expert to be successful.

VII. Discussion

The requirements for a common data model for AI/ML in EDA are illustrated here by scenarios or use cases. This section highlights Si2 OpenAccess and Common Data Model Requirements, Key Analysis Domains, applications for Shape Based Data, and a Structured Classification Methodology.

A. Si2 OpenAccess and Common Data Model Requirements

Si2 OpenAccess could be a building block for a complete AI/ML in EDA system (Fig. 9). The common data model requirements detailed in this paper would create an infrastructure which defines basic AI/ML features and allows EDA proprietary extensions. EDA suppliers would focus on writing to a standard AI/ML interface, and providing prepackaged AI/ML configurations for end users. Industry standard flows and engines would be leveraged, and the end user would own their application-specific data, features, and tests [11].



* Other names and brands may be claimed as the property of others.

Fig. 9. Suggestions for Role, Scope, and Interactions

B. Key Analysis Domains

1) Automotive: There has been a great advancement of technology in automotive electronics over the past decade, and it will continue evolving for the foreseeable future. The demand is high for a common data model enabling sharing. In addition, stricter rules (i.e., ISO 26262 Functional Safety) apply for automotive-grade IPs; hence, non-automotive data models may not entirely apply.

2) IoT: Similar to automotive, IoT technology continues to advance and evolve, and with it, the need for a common data model grows.

3) Consumer Product: Always a major sector in the electronic industry, new product fields keep emerging over time, such as smart phones, connected devices, etc.

4) Medical Devices: Stricter rules apply to this sector than most, as data may contain sensitive personal information. The different data formats for biomedical signals, such as EEG and ECoG, present additional challenges in adapting to a data model.

Each analysis domain should adopt different extraction methodologies and subject to different scenarios (regions, local laws, policies)

C. Shape Based Data

There are several opportunities for using machine learning-enabled EDA flows that leverage correlations between design physical layout information and vast amounts of silicon data to drive significant optimizations in IC physical implementation and semiconductor manufacturing. Examples of such applications include: optimizing performance of DRC/DFM and other physical verification checks using ML, ML-based design hotspot detection and DFM fixing for yield enhancement, employing ML to drive improvements in OPC modeling and mask making turnaround times, ML-enabled scan test analysis

and yield debug, and design aware ML-optimized semiconductor manufacturing recipe setup.

A standardized way to efficiently store and represent design physical data with linkages to various types of derived data is critical to enabling AI/ML for these applications. Along with compact representations of geometric data for all design layers, a mechanism of storing or deriving connectivity, annotating shapes with voltage and power domains, parasitic and timing information is needed as well. The ability to obfuscate and abstract-out some types of data will be desirable to protect IP, while still enabling downstream ML apps to make design intent-aware optimizations, such as providing design houses the ability to provide information to semiconductor foundries on shapes that are part of critical, timing-sensitive nets, without providing detailed timing information. The data model should ideally also provide the ability to store certain types of derived data and annotate shapes with outputs of analysis tools like lithography simulations and Chemical-Mechanical-Polishing topography simulations. Hierarchical representations, along with abilities to group and cluster based on selections of stored features, will enable efficiencies in ML training and inference flows. The data model should also preserve the ability for APIs to perform design location-aware grouping and sampling.

D. Structured Classification Methodology

Descriptions of current data classifications in different design areas suggest a methodology for bridging the gap between classification and ML data preparation and labeling by defining common standards. When there is interest in an AI/ML application, subject matter experts and stakeholders would meet to define the input design and derived data which needs structured classification, for example to and from multiple EDA tools and/or multiple semiconductor technologies. They would also define how the data would be referenced, for example through a common API, and the data transformations needed for AI/ML. The standard for this area can be published for industry use, preferably with a prototype demonstration flow. This standard can then be replicated and modified for new AI/ML applications. As described in Section II-A, Fig. 4, this standard for AI/ML data could be built on top of an existing design database, for example Si2 OpenAccess.

VIII. CONCLUSIONS

In order to accelerate AI/ML applications to EDA, a collaborative and coordinated approach is needed. A prerequisite for this approach is establishing the best process for organizing, leveraging and sharing data, since such data are the foundation on which AI/ML applications are built. Furthermore, from Si2 industry survey results, there is a gap in the availability and organization for AI/ML data in EDA. A common data model would address the data organization gap for chip developers, EDA tool developers, IP providers and researchers by first supporting the high interest EDA areas, design data and derived data.

An ideal data model for EDA applications is organized around the following components: Objects, composability and inheritance, relationships, operations, design versus derived data, APIs, and a means for obfuscation for sensitive IP

Further specialization is required in the space of analog design where the following signal vectors are highly relevant for design capture:

1) Analog simulation time-series data: data obtained directly from SPICE simulator or after post-processing as performance metrics.

2) Electrical design parameters: the parameters used during a manual and typical analog design.

3) Sizes and biases: the inputs to SPICE-like simulators.

4) Small-signal parameters: parameters arising from linear analysis performed during linearization by evaluating compact models such as BSIM and PSP.

5) Performance models: models used to evaluate linear and nonlinear performances.

In the space of digital design, one needs a data model to answer questions at the intersection of logic, placement, wiring, timing, power, and noise, among other variables. In particular, what is still needed, is an agreed-upon standard for efficient storage of derived data such as timing, noise, power in an offline data model, as well as a means to link derived data back to the design data objects. This superset of design data and derived data, all tightly coupled and linked, gives rise to a unified persistent data model enabling AI/ML in EDA.

A design flow usually involves several EDA tools generating a great deal of data at each step, some of which is redundant or duplicate data. AI/ML-based applications in EDA may not need all of the data generated at each step of the design flow. Additionally, some design data may be proprietary to the specific EDA vendor or the design house; therefore, it is important for the data model to support mechanisms for extracting derived data from the existing design data and/or analysis data. These mechanisms can involve processes including Data Relevance, Data Cleaning, Data Transformation, Enabling Extraction of Relevant Data, and whether data is Above versus Below the Line.

A complete design flow based on efficient design and data models is crucial for the IP Protection and Management of Innovation (disruptive or Breakthrough) within a corporate organization. We define three levels of sensitivity for obfuscation of sensitive IP, each with different handling guidelines: shareable data, partially sharable data, and confidential data.

The common data model requirements detailed in this work define an infrastructure for common AI/ML features with EDA proprietary extensions. EDA suppliers would focus on writing to a standard AI/ML interface and provide prepackaged AI/ML configurations for end users. Industry standard flows and engines would be leveraged, and the end user would own their analysis results. Si2 OpenAccess could be a building block for a complete AI/ML in EDA system. These subject matter experts agree that a standard, common method for classification and structure of machine learning training and inference data for interoperability is critical to enable and accelerate the use of artificial intelligence and machine learning in semiconductor electronic design automation.

REFERENCES

[1] B. Shook *et al*., "MLParest: Machine Learning based Parasitic Estimation for Custom Circuit Design," presented at the Design Automation Conf., Online, July 20-24, 2020.

[2] A. Mirhoseini *et al.*, "Chip Placement with Deep Reinforcement Learning," 2020, https://arxiv.org/pdf/2004.10746.pdf

[3] A. Mirhoseini *et al.*, "Device Placement Optimization with Reinforcement Learning," 2017, https://arxiv.org/pdf/1706.04972.pdf

[4] H. Wang *et al.*, "GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning," 2020, https://arxiv.org/abs/2005.00406

[5] X. Li, N. Qi, Y. He, and B. McMillan, (2019) "Practical Resource Usage Prediction Method for Large Memory Jobs in HPC Clusters," in Lecture Notes in Computer Science, vol. 11416. D. Abramson, B. de Supinski, Ed., Singapore, Mar. 2019, pp. 1-18, doi: https://doi.org/10.1007/978-3-030-18645-6

[6] L. Stok *et al.*, "Empowering the Designer Through Advanced Analytics and Machine Learning," presented at the 56th Design Automation Conf., Las Vegas, NV, USA, June 2-6, 2019.

[7] L. Stok. (2015). EDA 3.0 Time to refactor Logic Synthesis [PowerPoint]. Available: https://bit.ly/Leon_Stok_EDA_3

[8] C. Visweswariah *et al*., "First-Order Incremental Block-Based Statistical Timing Analysis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2170-2180, Oct. 2006, doi: 10.1109/TCAD.2005.862751

[9] A. Malak *et al.*, "Fast multidimensional optimization of analog circuits initiated by monodimensional global Peano explorations," *Integration,* vol. 48, pp. 198-212, Jan. 2015, doi: https://doi.org/10.1016/j.vlsi.2014.04.002

[10] R. Iskander, M. Louërat, and A. Kaiser, "Hierarchical sizing and biasing of analog firm intellectual properties," *Integration,* vol. 46, no. 2, pp. 172-188, Mar. 2013, doi: https://doi.org/10.1016/j.vlsi.2012.01.001

[11] J. D. Masters, "Genetic algorithm EDA experiment and suggestions for Si2 AI/ML WG," unpublished.