

Berkeley Open MOS dataBase (BOMB): A Dataset for Silicon Technology Representation Learning

*Rohan Lageweg
Vladimir Stojanovic, Ed.
Kourosh Hakhamaneshi, Ed.*



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-192

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-192.html>

August 13, 2021

Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**Berkeley Open MOS dataBase (BOMB):
A Dataset for Silicon Technology Representation Learning**

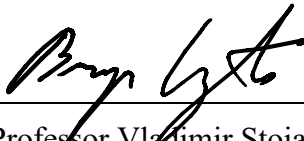
by Rohan Lageweg

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

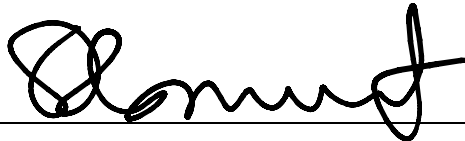


Professor Vladimir Stojanovic
Research Advisor

08/11/2021

(Date)

* * * * *



Professor Borivoje Nikolic
Second Reader

8/12/2021

(Date)

Berkeley Open MOS dataBase (BOMB): A Dataset for Silicon Technology Representation Learning

Rohan Lageweg

August 2, 2021

Abstract

In this work, we present Berkeley Open MOS dataBase (BOMB), the first ever open source database of un-annotated transistor characterization data that can be used for technology representation learning. BOMB was created by running large scale transistor characterization scripts on various device flavors and technology nodes. Each data point in BOMB is essentially a multi-dimensional array capturing I-V and Y-V characterization of transistors. The meta-information revealing the technology is removed to maintain confidentiality such that the dataset to be used by a greater community of researchers. Additionally, we present an API with which the data can be accessed and visualized, as well as a framework with which additional data can be collected and added to the dataset. Finally, we provide statistics about the distribution of datapoints and visualizations which demonstrate the inherent structure of the dataset. Plans for future work involving downstream machine learning tasks are also discussed.

Acknowledgements

I would like to begin by thanking Nick Werblun, who guided me from my time as EE 16A Lab ASE, through being part of EE 16A Course Staff and finally into the world of undergraduate research. I would also like to thank Professor Vladimir Stojanovic for his sponsorship and excellent guidance for this project and my research.

I would like to thank Arjun Mishra for his work on the machine learning aspect of this project, and Kourosh Hakhamaneshi for his continuous mentorship and guidance throughout the duration of this project.

Finally, I would like to thank my family and in particular my parents for supporting me throughout my education.

Contents

1	Introduction	4
2	BOMB: Structure and Framework	6
2.1	SimData Class: An API to work with BOMB	7
2.2	Abstracting away SimData Class: Using main.py to view dataset	8
3	How was BOMB created?	9
3.1	Example: Generate dataset using BOMB repository framework	9
4	BOMB Repository	13
5	Dataset Properties	14
6	Conclusion and Future Direction	17

1 Introduction

The semiconductor industry owes its rapid progress in the last century to the scaling of silicon process technology which has constantly allowed designers to fit more processing power into the same area footprint and make today’s digital processing capabilities a reality. However, this comes at a great cost of re-design efforts for making integrated circuits (ICs) into production as the technology scales. Recent attention has been made to shortening the design cycle by utilizing advances made in machine learning on automating some manual pieces of this endeavor such as high-level synthesis [1], logic synthesis [2,3], placement and routing [4–6], analog mixed signal design and layout exploration [7–10], manufacturing [11,12], .etc ¹.

One thing that still remains an issue with these deep learning solutions is their capability to transfer their knowledge across technologies i.e. if a NN is trained on a particular circuit task in a given technology, it will only work in that process node and to reuse it in a new unseen technology, the training has to be done from scratch with a new data collected consistent with the target technology node. This problem is of special importance in analog mixed signal (AMS) design automation where the impact of technology changes on performance is much more than their digital counterparts. Also digital design has enjoyed a plethora of automation tools that facilitate the technology migration.

Typically, the data in this domain comes from simulations that are expensive to run and therefore large scale data collection is prohibitively expensive, making data-hungry machine learning methods impractical. Such limitation poses a serious challenge when considering transferring to new technologies. Ideally, the machine learning system trained on several technologies should have built enough prior knowledge about the circuit design procedure, transferring the knowledge in a few-shot / zero-shot manner. The motivation for this work is to see *if we can learn a representation of silicon technologies that can be used during training of circuit design models and enable transferring design knowledge to new unseen technologies?* To this end, we will present a dataset, a data collection procedure and suggest an exemplar ML algorithm for learning such representation.

The transistor is the basic building block of all modern electronic circuits, usually configured as switch in digital circuits or an as amplifier in analog circuits. Of particular interest to analog circuit designers are so-called “Y-parameters”, which capture the behavior of a transistor when linearized around a chosen operating point. These parameters describe a mathematical model of the computations done by transistors abstracting away the physical phenomena that fundamentally explain the movement of charge carriers in silicon ². An understanding of how these small signal parameters behave under various process, temperature and voltage conditions is informative about the technology and performance of the circuit.

In this work, we present Berkeley Open Mos dataBase (BOMB), the first ever open source database of un-annotated transistor characterization data that can be used for technology representation learning. BOMB was created by running large scale transistor characterization scripts on various device flavors and technology nodes. Each data point in BOMB is essentially a multi-dimensional array capturing I-V (current vs. voltage) and Y-V characterization of transistors. Since the confidentiality of the technology information is of special importance we have removed the meta-information revealing the technology identity so that it can be used by a greater community of researchers.

We hope that with the development of this dataset we can fuel the research direction of employing deep learning approaches for chip design. We summarize the contribution of our work as follows:

1. We present BOMB, the first large scale open source database for silicon technology with 96,600 transistor characterizations.
2. We provide statistical studies of the distribution of the data points within BOMB and explain what kind of technology specific information can be inferred from them.

¹For a comprehensive survey on using machine learning in electronic design automation see [13].

²To read more about Y-parameters see [14]

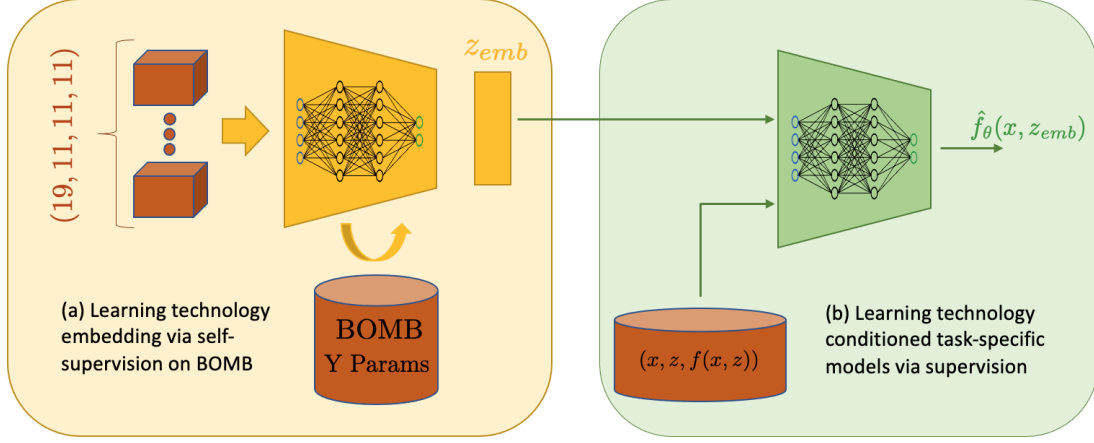


Figure 1: This figure shows two phases: 1) pretraining the representation learning using BOMB dataset 2) using the learned representation for technology conditioned downstream circuit task prediction

3. We suggest how a pre-existing unsupervised learning procedure (VAE) can be used to learn representations that enable learning technology conditioned circuit models in a sample efficient manner.

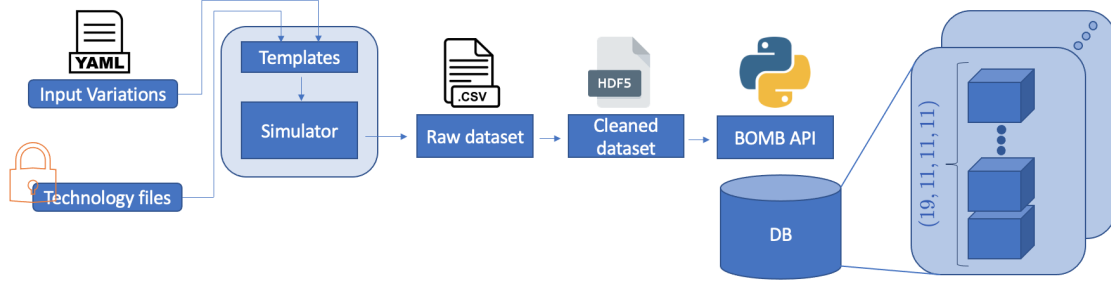


Figure 2: High level figure describing the structure of BOMB and how it was generated.

2 BOMB: Structure and Framework

Each datapoint in the BOMB dataset consists of multi-dimensional arrays capturing Ibias and Y-parameters across the three transistor terminal voltages V_{bs} , V_{gs} and V_{ds} . The datapoints are collected over Monte Carlo, process, temperature, and device variations. The result is a dataset that encapsulates all the information about a given CMOS technology. Our vision is that this dataset will be used to learn a representation of the silicon technology that can be used during training of circuit design models and enable transferring design knowledge between technologies. The dataset will first be used in a pre-training phase, where the representation is learned. This representation can then be used to downstream circuit design tasks.

This data has been collected for two state-of-the-art CMOS technologies so far. The sweep ranges for each input variable are:

- Monte Carlo: 100 randomly generated Monte Carlo variations. These are generally used to model imperfections and mismatches in the manufacturing process.
- Temperature: -20C, 27C, 120C
- Process: Depends on the technology. A range of typical, fast and slow process corners.
- Device: Depends on the technology. A range of device types provided by the foundry for the given technology.
- Terminal voltages: Voltages are swept from 0 to the supply voltage in 11 uniformly spaced steps.

The end result is a Python dictionary of multidimensional Numpy arrays for each of Ibias and the Y-parameters, stored in HDF5 file format. The dictionary is organized as follows:

```

1 bomb.data = {
2   "ibias": np.ndarray([montecarlo, temperature, process, device, Vbs, Vgs, Vds]),
3   "y11": np.ndarray([montecarlo, temperature, process, device, Vbs, Vgs, Vds]),
4   ...
5   "y33": np.ndarray([montecarlo, temperature, process, device, Vbs, Vgs, Vds])
6 }
```

The HDF5 files can be loaded into a SimData object using the API provided in the BOMB repository. A full description of the SimData class is provided in the following section. Below is an example of the code used to load the data.

```

1 >> bomb = SimData.load("Technology_A_data.hdf5")
2 >> data = bomb.data["ibias"]
3 >> print(data.shape)
4 >> (100, 5, 3, 9, 11, 11, 11)

```

2.1 SimData Class: An API to work with BOMB

The SimData class has the following instance variables:

SimData.data: Dictionary containing multi-dimensional arrays for each of Ibias and Y-parameters formatted as shown above. Note that each array consists of seven dimensions, with each dimension corresponding to Monte Carlo, process corner (tt, ss, ff etc), temperature (usually -20C, 27C, 120C), device type (depends on the technology node) and the bias voltages V_{bs} , V_{gs} and V_{ds} respectively.

SimData.sweep_params: List of sweep variables that correspond to each dimension in the SimData.data. Hard coded as: ['montecarlo', 'process', 'temp', 'vbs', 'vgs', 'vds']

SimData.resaped_data: Populated with a single Numpy array (dim=[-1, 19, 11, 11, 11]) of reshaped data when the corresponding getter method is called. We treat each [19, 11, 11, 11] array as a single datapoint, carrying information about Ibias and the real and imaginary parts of all nine Y-parameters and corresponding to a single combination of Monte Carlo, process, temperature and device. The number of datapoints then is equal to the product of the number of Monte Carlo, process, temperature and device variations.

SimData.ss_params: Populated with small signal parameters calculated from y-parameters when the corresponding getter method is called.

The SimData class also defines the following methods:

SimData.__getitem__(item_key): Defined such that SimData['montecarlo'] returns list of Monte Carlo values in dataset, SimData['process'] returns list of process values in dataset, etc. and SimData['ibias'] returns multi-dimensional array containing Ibias data, SimData['y11'] returns multi-dimensional array containing y11 data etc.

SimData.load(hdf5_file): Load dataset into SimData object from HDF5 file

SimData.save(hdf5_file): Save dataset into HDF5 file from SimData object

SimData.concat(objs, axis): Concatenate list of SimData objects across given axis (montecarlo, process, temp, etc)

SimData.get_resaped_data(): Getter method to populate SimData.resaped_data with reshaped data (dim=[-1, 19, 11, 11, 11])

SimData.get_ss_params(): Getter method to populate SimData.ss_params with small signal parameters

SimData.PCA(filter): Plot PCA for reshaped dataset of stacked Ibias and Y-parameters, and color clusters according to filter (montecarlo, process, temp, or device)

SimData.TSNE(filter): Plot TSNE for reshaped dataset of stacked Ibias and Y-parameters, and color clusters according to filter (montecarlo, process, temp, or device)

SimData.sanity_plot(): Plot real(y21), -real(y31), both of which equal gm and also plot ibias for 0th and 1st MC indices. Use as a sanity check to ensure data looks reasonable

SimDataWrapper is an additional class that adds functionality when handling multiple HDF5 files representing multiple datasets from different technologies.

The only instance variable is **SimData_objs**, a list of SimData objects. It defines the following methods. Note that x, y are defined as the index of SimData object in **self.SimData_objs** list and the index of datapoint in reshaped data.

SimDataWrapper.get_data(x, y): Get data corresponding to x, y

SimDataWrapper.get_metadata(x, y): Get metadata (montecarlo, process, temp and device) corresponding to x, y

`SimDataWrapper.search(mc, process, temp, device)`: Search for the specified metadata
`SimDataWrapper.advanced_search(search_slice)`: Search for all datapoints with the specific search slice, where each search slice can be a specific Monte Carlo, process, temperature, or device

2.2 Abstracting away SimData Class: Using main.py to view dataset

While the `SimData` class provides a useful framework with which to view and manipulate the dataset, it is part of the source code of the repository and may take some groundwork before it can be used effectively. With this in mind, the BOMB repository also provides an "out-of-the-box" way of viewing and visualizing the dataset, through command line arguments. The user can simply run

```
1 python main.py data --load_hdf5 <path_to_hdf5_file>
```

This will load the HDF5 file and open an interactive prompt from where various commands can be entered to examine and visualize the data.

```
20:39:37 $ python main.py data --load_hdf5 data/anonymized_dataset/tech1_nmos.hdf5
A python debugger window is now open! From here, you can examine and visualize the data you have loaded!
data.sweep_params to see sweep parameters
data['montecarlo'] / data['process'] etc to see values of sweep params corresponding to indices of data array
data.data to see all data. WARNING: THIS IS LARGE
data.data.keys() or data['y11'].shape to get a feel for the data
data.sanity_plot() for sanity plot of real(y21) (=gm), -real(y31) (=gm) and ibias
data.PCA(filter='process') to see PCA representation of data
c to exit debugger
--Return--
> /tools/B/mos_char_prj/gf12_workspace/bomb/main.py(111)call_data()->None
-> breakpoint()
(Pdb) data['process']
['process_0', 'process_1', 'process_2', 'process_3', 'process_4']
(Pdb) data['device']
['device_0', 'device_1', 'device_2', 'device_3', 'device_4', 'device_5', 'device_6']
(Pdb) data['ibias'].shape
(100, 5, 3, 7, 11, 11, 11)
(Pdb) data.sanity_plot()
```

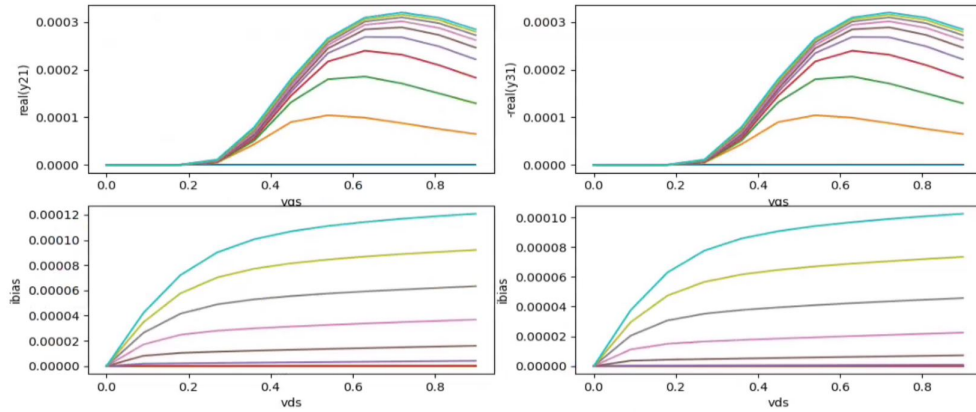


Figure 3: This figure shows an example of using `main.py` to interactively view the dataset

3 How was BOMB created?

Figure 2 describes, from a high level how the BOMB dataset was created. We will demonstrate this process in more detail with the help of an example, generating a dataset from scratch based on the `cds_ff_mpt` technology (a Cadence generic PDK for FinFET and multi-patterned technology).

3.1 Example: Generate dataset using BOMB repository framework

In order to run MOS characterization to create a new dataset (or add onto an existing dataset), this repository needs to be added to a working bag setup for a given technology, since it relies on a certain folder structure format. The idea is that the user creates a customized yaml from the given template, then generates the ocn scripts that can be run in the Cadence CIW. Here is a step by step guide, using the creation of a dataset on the `cds_ff_mpt` technology as an example:

1. Clone a BAG (Berkeley Analog Generator) workspace for the desired technology, cd into the BAG workspace, follow its own instructions, and then clone the BOMB repo.
2. Open up Cadence library manager. Under `bag_testbenches/bag3_testbenches`, make a copy of `mos_tb_sp` named `mos_tb_sp_<device>`. Replace the DUT with the appropriate device from the PDK library.

ex. We replace the DUT with the 'n1svt' device (NMOS 1V standard voltage threshold) and name the schematic `mos_tb_sp_n1svt`.

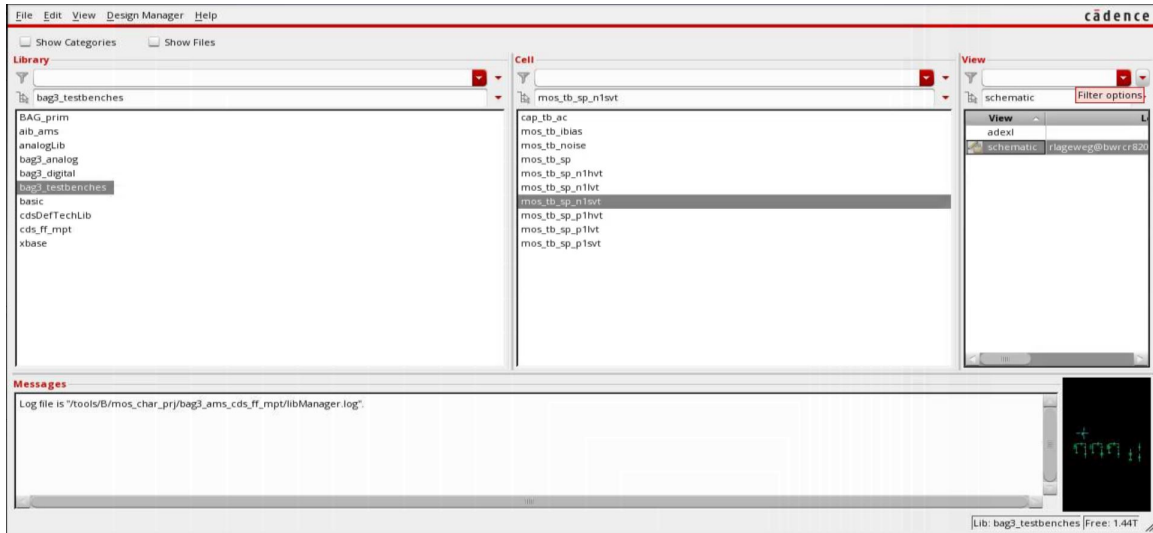


Figure 4: (Step 2) Open up Cadence library manager.

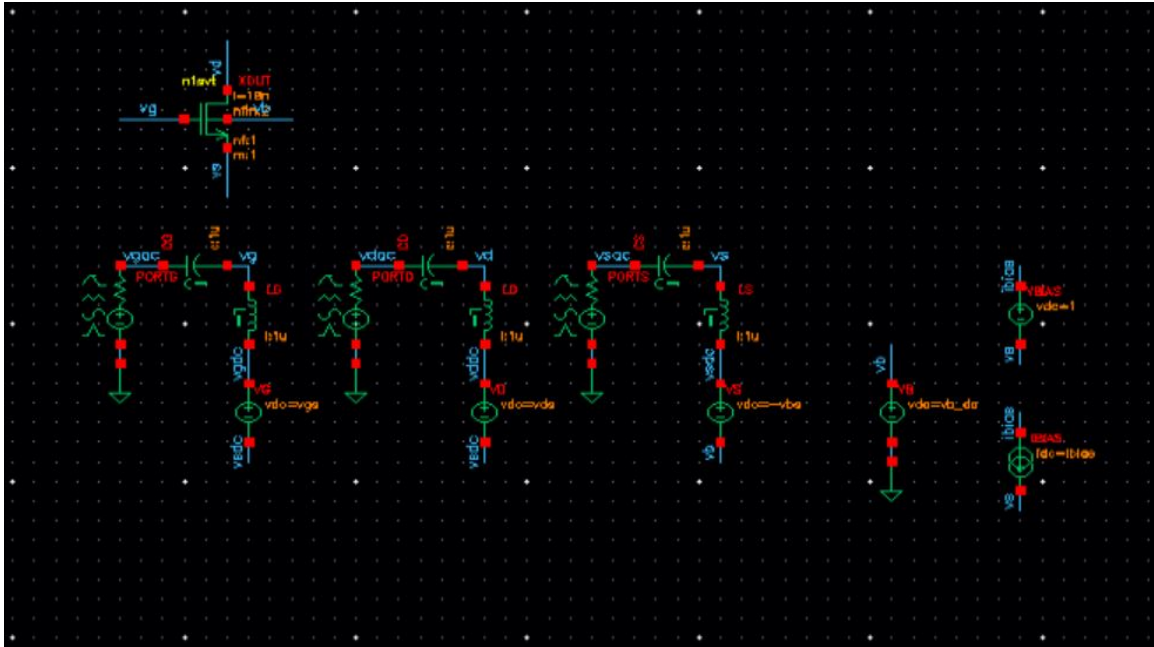


Figure 5: (Step 2) Replace the DUT with the appropriate device from the PDK library.

3. Make a copy of template.yaml and enter technology specific sweep parameters. The device name should match the device name of the schematic (`mos_tb_sp_<device>`). It's good practice to keep the total number of simulations under 200k to avoid stack overflow messages from Virtuoso. The number can be adjusted by creating sub yamls; for example sweep 5 corners in one yaml and 4 corners in another yaml. We will merge the data collected later.

```

1  ---
2  # Sweep inputs
3  monte Carlo: 10
4  temp: ['-20', '27', '120']      # should be a list
5  process: ['mc']                  # should be a list
6  device: 'n1svt'
7  vdd: 1
8  is_nmos: True
9  tech_model: '/tools/cadence/GSPDK/cds_ff_mpt_v_0.5/models/spectre/cds_ff_mpt.scs'
10 bag_version: 'bag3'
11
12 # Path inputs; starting root is root of bomb repo
13
14 # Directory path where csv's from Cadence sweep will be stored. If not specified, then defaults to 'data/<yaml_name>'
15 data_outdir: ''
16
17 # Directory path where generated ocean script is stored. If not specified, then defaults to 'generated_files/'.
18 # Recommended to leave this as default
19 generated_files_outdir: ''
20
21 # Directory path where scratch files from Cadence sweep are generated. Delete these files after the sweep completes
22 scratch_outdir: '/tools/scratch/mos_char_prj'
23
24 # Monte Carlo sweep settings
25 maxjobs: 8
26 mcSeed: 1
27 mcStart: ''

```

Figure 6: (Step 3) Make a copy of template.yaml and enter technology specific sweep parameters.

4. cd into bomb, source .bashrc and run `python main.py ocean --yaml <path_to_yaml>`.

```

(base) {rlageweg@bwrcr820-2:/tools/B/mos_char_prj/bag3_ams_cds_ff_mpt}
15:37:36 $ cd bomb/
(base) {rlageweg@bwrcr820-2:/tools/B/mos_char_prj/bag3_ams_cds_ff_mpt/bomb}
15:37:38 $ source .bashrc
(base) {rlageweg@bwrcr820-2:/tools/B/mos_char_prj/bag3_ams_cds_ff_mpt/bomb}
15:37:42 $ python main.py ocean --yaml nlsvt.yaml
Created /tools/B/mos_char_prj/bag3_ams_cds_ff_mpt/bomb/data/nlsvt
device: nlsvt
number of corners: 3
number of mc: 10
number of simulations: 3630
{'montecarlo': 10, 'temp': ['-20', '27', '120'], 'process': ['mc'], 'device': 'nlsvt', 'vdd': 1, 'is_nmos': True, 'tech_model': '/tools/cadence/GDPK/cds_ff_m
pt v 0.5/models/spectre/cds_ff_mpt.scs', 'bag_version': 'bag3', 'data_outdir': '/tools/B/mos_char_prj/bag3_ams_cds_ff_mpt/bomb/data/nlsvt', 'generated_files_
outdir': '/tools/B/mos_char_prj/bag3_ams_cds_ff_mpt/bomb/generated_files', 'scratch_outdir': '/tools/scratch/mos_char_prj', 'maxjobs': 8, 'mcSeed': 1, 'mcSta
rt': ''}
Enter <load("/tools/B/mos_char_prj/bag3_ams_cds_ff_mpt/bomb/generated_files/generated_nlsvt.ocn")> in Cadence CIW to run sweep

```

Figure 7: (Step 4) Run `python main.py ocean --yaml <path_to_yaml>`.

- Follow the printed output from `main.py` to run the `generated_monte_carlo_*.ocn` in Cadence CIW. This will generate a csv for each of Ibias and the Y-parameters in the output directory specified in the yaml file.

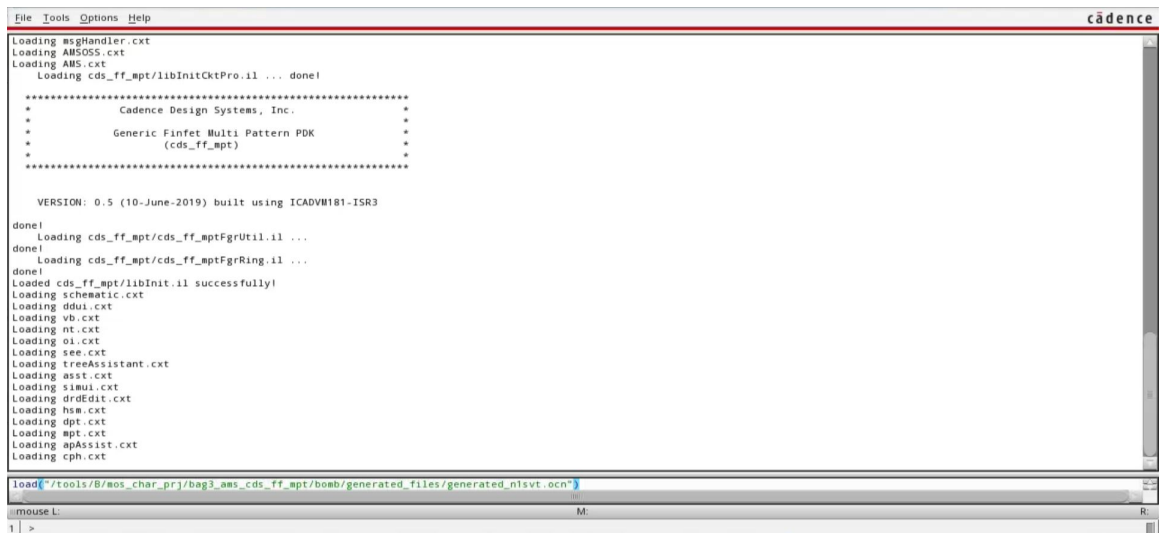


Figure 8: (Step 5) Run the `generated_monte_carlo_*.ocn` in Cadence CIW.

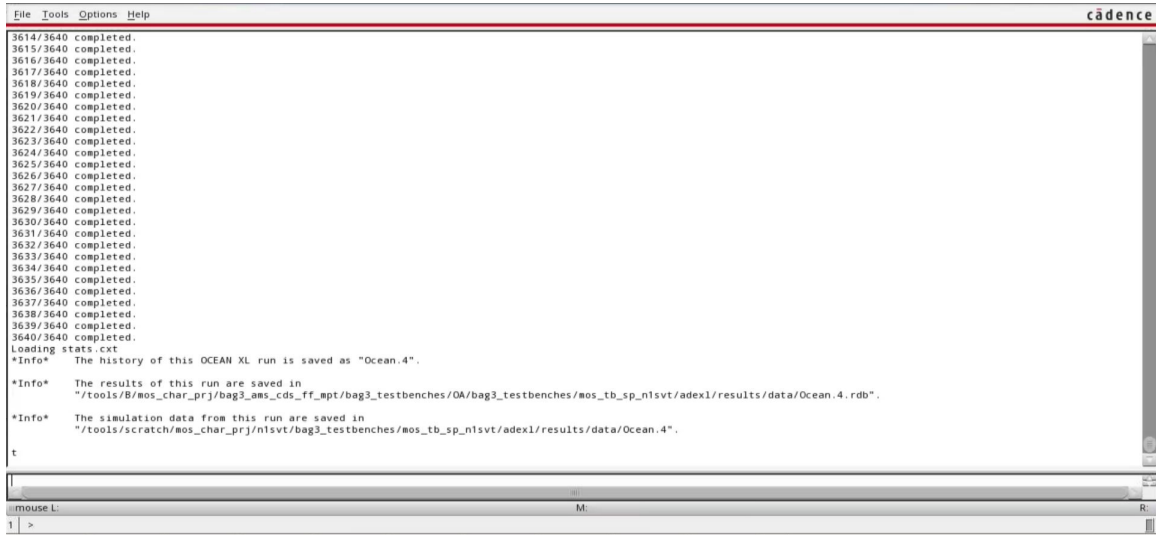


Figure 9: (Step 5) Cadence CIW after completion of the sweeps.

```

(base) {rlageweg@bwrcr820-2:/tools/B/mos_char_prj/bag3_ams_cds_ff_mpt/bomb}
15:37:49 $ ls data
./ ../ .gitignore nlhvt/ nlvt/ nlsvt/ plhvt/ plvt/ plsvt/
(base) {rlageweg@bwrcr820-2:/tools/B/mos_char_prj/bag3_ams_cds_ff_mpt/bomb}
16:00:32 $ ls -lrth data/nlsvt/
total 30M
drwxrwsr-- 8 rlageweg bwrcgroup 4.0K Jun 24 15:37 ../
-rw-r--r-- 1 rlageweg bwrcgroup 482 Jun 24 15:45 sweep_info.txt
-rw-r--r-- 1 rlageweg bwrcgroup 1.8M Jun 24 15:45 ibias.vcsv
-rw-r--r-- 1 rlageweg bwrcgroup 3.0M Jun 24 15:45 y11.vcsv
-rw-r--r-- 1 rlageweg bwrcgroup 3.1M Jun 24 15:46 y12.vcsv
-rw-r--r-- 1 rlageweg bwrcgroup 3.1M Jun 24 15:46 y13.vcsv
-rw-r--r-- 1 rlageweg bwrcgroup 3.1M Jun 24 15:46 y21.vcsv
-rw-r--r-- 1 rlageweg bwrcgroup 3.0M Jun 24 15:46 y22.vcsv
-rw-r--r-- 1 rlageweg bwrcgroup 3.1M Jun 24 15:46 y23.vcsv
-rw-r--r-- 1 rlageweg bwrcgroup 3.1M Jun 24 15:46 y31.vcsv
-rw-r--r-- 1 rlageweg bwrcgroup 3.1M Jun 24 15:46 y32.vcsv
drwxr-sr-x 2 rlageweg bwrcgroup 4.0K Jun 24 15:46 ./
-rw-r--r-- 1 rlageweg bwrcgroup 3.0M Jun 24 15:46 y33.vcsv

```

Figure 10: (Step 5) Data from the completed Cadence sweeps.

Once the Cadence sweeps have been run, there exists another script to convert the outputted csv's into HDF5 file format. Simply run

```
python main.py csv --sim_datadir <path_to_output_csvs_directory>
--save_hdf5 <path_to_desired_HDF5_file_save_location>
```

This will extract all the relevant information from the csv's and store it in a single SimData object. The data will also be stored in HDF5 file format which can then be loaded into a SimData object quickly. The API for loading from HDF5 files and working with SimData objects has been discussed above.

It is also possible to merge two HDF5 files together along a given axis (Monte Carlo, process, temperature or device) with the following command:

```
python main.py concat --hdf5_list <path_to_HDF5_A> <path_to_HDF5_B>
--concat_axis <montecarlo/process/temp/device>
--save_hdf5 <path_to_desired_HDF5_file_save_location>.
```

4 BOMB Repository

By now, the functionality of the BOMB repository in accessing the dataset and creating new datasets has been introduced. Now we show in detail how the code in the BOMB repository is organized.

- **src:**
 - **scripts:**
 - * **ocean.py:** uses jinja templates to generate Ocean script which is run in Cadence CIW to collect data
 - * **csv.py:** parse csv outputs from Cadence sweeps and return SimData object
 - * **data.py:** defines SimData class which enables parsing and visualization of dataset, as well as saving to loading from hdf5 files
 - **templates:** contains templates necessary for **ocean.py** to run
- **data:** data from Cadence sweeps should be saved here, as well as hdf5 files storing the data
- **main.py:**
 - defines 4 functions and uses argparse to determine which function to call:
 - * **ocean:** calls **src/scripts/ocean.py** to generate ocean script for Cadence sweep based on input yaml
 - example usage: `python main.py ocean --yaml <path_to_yaml>`
 - * **csv:** calls **src/scripts/csv.py** to parse data generated by Cadence sweep into SimData object. Can also optionally provide an argument to save SimDarta object into hdf5 file
 - example usage:
`python main.py csv --sim_datadir <path_to_sim_data_directory>
--save_hdf5 <path_to_hdf5_file>`
 - * **data:** calls **src/scripts/data.py** to load SimData object from hdf5 file
 - example usage: `python main.py data --load_hdf5 <path_to_hdf5_file>`
 - * **concat:** calls **src/scripts/data.py** to concat multiple hdf5 files together along a given axis. Can also optionally provide an argument to save concatenated object into hdf5 file
 - example usage:
`python main.py concat --hdf_list <path_to_hdf5_file_1> <path_to_hdf5_file_2>
--concat_axis device --save_hdf5 <path_to_hdf5_file>`
 - `python main.py -h` to see argument options
- **template.yaml:**
 - template yaml file that defines sweep parameters (general and technology specific)
 - make your own copy and enter technology specific sweep parameters

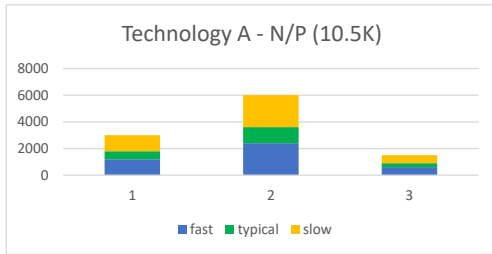
5 Dataset Properties

The intent of designing BOMB dataset is to capture a diverse set of transistor characteristics across a variety of technologies, process corners, temperatures, and device types. To this end, the initial version of BOMB includes 96K device characterizations collected via simulation on various dimensions of variability.

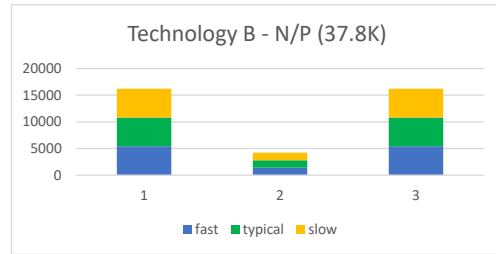
The dataset consists of N and P type transistors from two modern CMOS processes, for a total of four different sources of variations: Device type, temperature, process corner, and Monte Carlo. Both technologies are categorized into three different device types. Note that similarly labeled devices from the two technologies are not necessarily related, since they correspond to different manufacturing procedures. Besides varying the device type and temperature during simulation, we also include coarse and fine variations introduced during manufacturing that are modeled by process corner and Monte Carlo simulations, respectively. Process corners can be loosely categorized into typical, slow, or fast. Some circuits may run slower or faster than nominal specifications, depending on the process corner at which they are fabricated. The Monte Carlo variation is typically modeled via statistical distributions that can be sampled during simulation, allowing designers to account for these variations during their designs. Table 1 summarizes the sources of variations and Figure 11 illustrates the number different datapoints per each split.

Table 1: Sources of variations in the BOMB dataset

Data Subset		Size of Dimension				Num. Datapoints
Technology	CMOS	Monte Carlo	Process	Temperature	Device	
A	NMOS	100	5	3	3	10500
A	PMOS	100	5	3	3	10500
B	NMOS	100	9	3	3	37800
B	PMOS	100	9	3	3	37800



(a)



(b)

Figure 11: The histogram of the dataset according to different splits of device type and process corner.

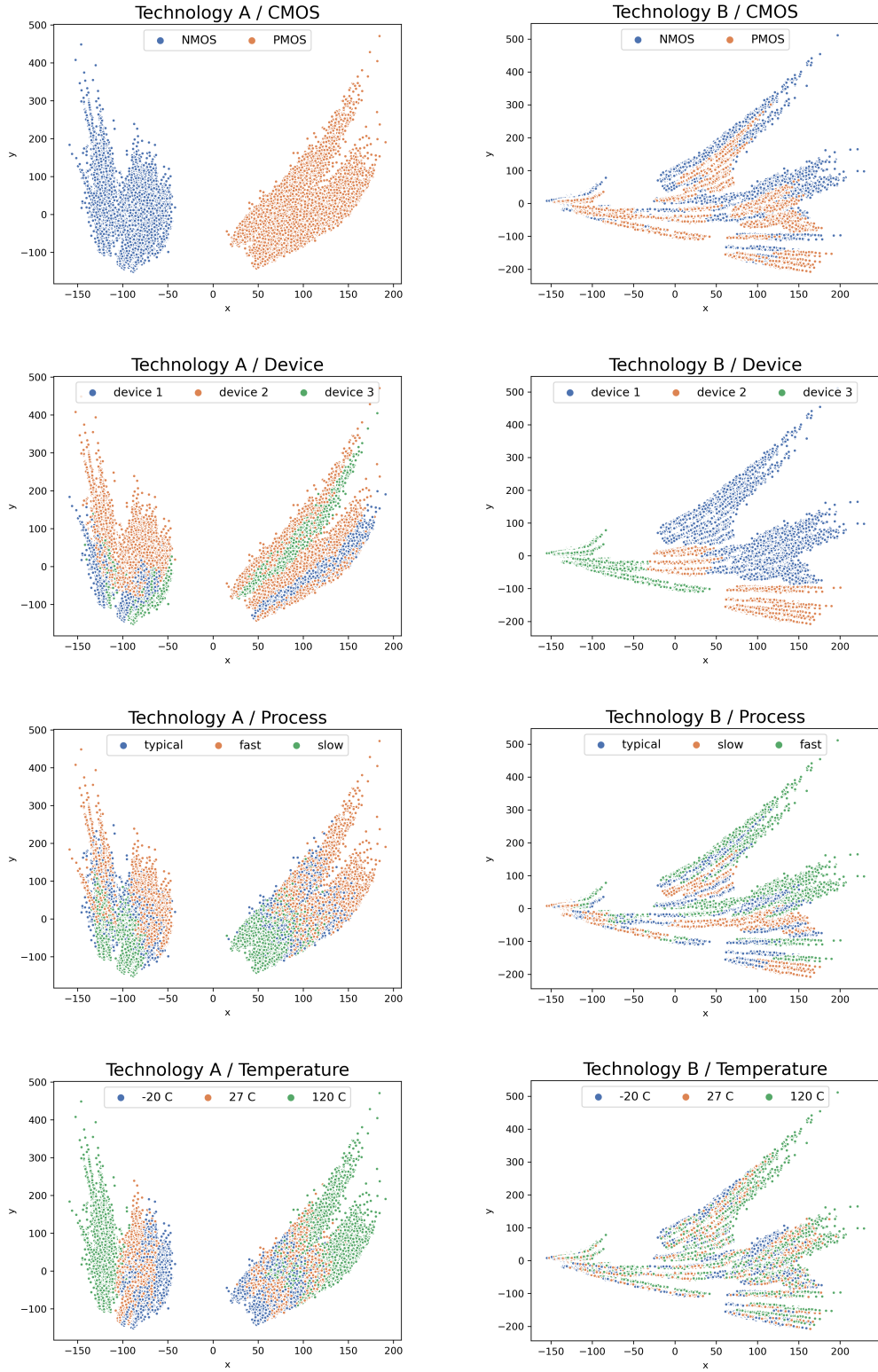


Figure 12: The distribution of the dataset projected to two dimensions using PCA. We can see clear structure in the data that can be compressed to a lower dimensional representation

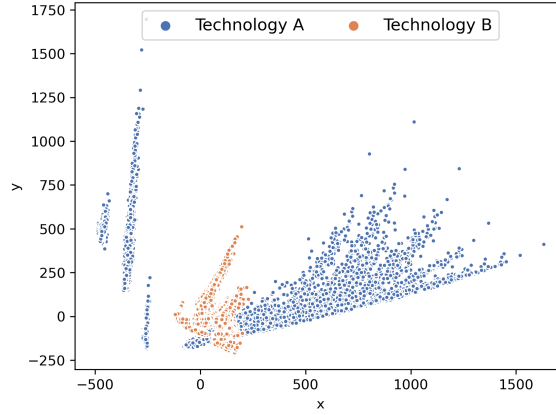


Figure 13: Projection of technology A and B on to the same space.

These inherent technology dependent categorizations lend themselves naturally to interesting training/test splits for experimental machine learning tasks. For example it would be interesting to see if a predictive model pretrained on diverse set of fast and slow corners can be efficiently fine-tuned for predictions on typical corners, in a few-shot manner. The inherent structure of the dataset can be visualized by applying PCA or t-SNE on the dataset. Figure 12 is an example of one such visualization, performing PCA on the Technology A and B NMOS subsets.

We observe that for both technologies A and B, the clearest clustering is by device, indicating that the largest principal component is very informative about the device type. The information about temperature and process on the other hand, is slightly blended for technology A, which motivates learning a better non-linear representation for transistor characterization data that can capture this high level information.

Finally, we can compute the principal components of technology B and project technology A onto the basis vectors given by the PCA of technology B. The result is shown in Figure 13. The clear separation of the two technologies indicates that the generalization across different technology nodes will be non-trivial.

6 Conclusion and Future Direction

In this work we have presented Berkeley Open MOS dataBase (BOMB), the first ever open source database of transistor characterization data that can be used for technology representation learning. We have described the structure of the dataset and how it can be accessed using the SimData class provided in the BOMB repository. With the help of an example, we have shown how the framework in the repository can be used to create a new dataset on a different silicon technology, or to create more data to add to the existing dataset. Finally, we provided statistical studies of the distribution of datapoints within BOMB and visualizations using PCA to demonstrate an inherent structure to the dataset that can be taken advantage of for later downstream machine learning tasks.

In future work, we would be interested in understanding the usefulness of BOMB for unsupervised representation learning of CMOS technologies and how it can be utilized to facilitate generalization to downstream circuit predictive model across technologies. To this end we plan to consider investigation of the following questions: 1) Can we use variational auto-encoders (VAEs) to learn a continuous embedding that preserves important properties of transistors? 2) Can we use the pre-trained embedding encoder to learn a circuit predictive model that can generalize to unseen device types?

References

- [1] Hosein Mohammadi Makrani, Hossein Sayadi, Tinoosh Mohsenin, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. Xppe: cross-platform performance estimation of hardware accelerators using machine learning. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 727–732, 2019.
- [2] Winston Haaswijk, Edo Collins, Benoit Seguin, Mathias Soeken, Frédéric Kaplan, Sabine Süsstrunk, and Giovanni De Micheli. Deep learning for logic optimization algorithms. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2018.
- [3] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. Drills: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 581–586. IEEE, 2020.
- [4] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.
- [5] Yi-Chen Lu, Jeehyun Lee, Anthony Agnesina, Kambiz Samadi, and Sung Kyu Lim. Gancts: A generative adversarial framework for clock tree prediction and optimization. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.
- [6] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. Routenet: Routability prediction for mixed-size designs using convolutional neural network. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.
- [7] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.
- [8] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. Autocckt: deep reinforcement learning of analog circuit designs. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 490–495. IEEE, 2020.
- [9] Kourosh Hakhamaneshi, Nick Werblun, Pieter Abbeel, and Vladimir Stojanović. Bagnet: Berkeley analog generator with layout optimizer boosted with deep neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.
- [10] Yaguang Li, Yishuang Lin, Meghna Madhusudan, Arvind Sharma, Wenbin Xu, Sachin S Sapatnekar, Ramesh Harjani, and Jiang Hu. A customized graph neural network model for guiding analog ic placement. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.
- [11] Yiyang Jiang, Fan Yang, Bei Yu, Dian Zhou, and Xuan Zeng. Efficient layout hotspot detection via binarized residual neural network ensemble. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1476–1488, 2020.
- [12] Wei Ye, Mohamed Baker Alawieh, Yibo Lin, and David Z Pan. Lithogan: End-to-end lithography modeling with generative adversarial networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

- [13] Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, et al. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 26(5):1–46, 2021.
- [14] Yannis Tsividis. *Operation and Modeling of the Mos Transistor (The Oxford Series in Electrical and Computer Engineering)*. Oxford University Press, Inc., USA, 2004.