SoftwareX 30 (2025) 102166

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

MLFoMpy: A post-processing tool for semiconductor TCAD data with machine-learning capabilities

Enrique Comesaña [a,*], Julian G. Fernández [b], Natalia Seoane [b], Antonio García-Loureiro [b]

[a] Escola Politécnica Superior de Enxeñaría, Universidade de Santiago de Compostela, Campus Terra, 27002 Lugo, Spain
[b] Centro Singular de Investigación en Tecnoloxías Intelixentes, Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain

ARTICLE INFO

Keywords:
Semiconductor device modeling
Nanoelectronics
Data processing
Machine learning
Python
Pytorch

ABSTRACT

We present MLFoMpy, a Python package for post-processing data from semiconductor device simulations. The software automatically extracts key figures of merit from current–voltage curves of field effect transistor and calculates statistical analyses for these curves. MLFoMpy also includes machine learning tools to predict figures of merit and current–voltage curves for devices with intrinsic variability. Additionally, it offers data visualization tools to plot current–voltage curves and statistical graphs.

Code metadata

Current code version	0.0.4
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00022
NanoHub tool	https://nanohub.org/tools/mlfompyusc/
Legal Code License	GNU General Public License v3.0
Code versioning system used	git
Software code languages, tools, and services used	Python, Matplotlib, seaborn, Torch, scikit-learn
Compilation requirements, operating environments & dependencies	Linux OS, Tested for Python 3.7 on Ubuntu 22.04
If available Link to developer documentation/manual	https://mlfompy.readthedocs.io/
Support email for questions	modev@usc.gal

1. Motivation and significance

In the realm of nanoelectronics, an unresolved concern revolves around the approaching limit of transistor scaling. This impending limitation threatens the continuous progression of the digital revolution witnessed in the past five decades [1]. Thus, it is pressing to explore new alternatives for implementation in forthcoming transistor technology nodes. At present, new field-effect transistors (FET) architectures are being proposed like gate-all-around devices such as nanosheet FETs or nanowire FETs [2], due to their exceptional electrostatic control [3].

A FET is a multi-gate semiconductor device with three terminals, source (connected to ground), gate (connected to a V_G bias) and drain (connected to a V_D bias). The drain current I_D is the flow of charge carriers inside the device. FETs can be affected by different sources of variability like line-edge-roughness (LER) or metal-grain-granularity (MGG), among others, that modify their electrical characteristic, making it necessary to perform extensive statistical studies [4].

As advanced FET architectures emerge, a critical gap remains in understanding the impact of variability sources on their electrical performance. Comprehensive and automated statistical analyses tailored to nanosheet and nanowire FETs are still needed. This demands extensive exploration and significant computational resources. Thus, developing tools to systematize and optimize these analyses is crucial for ensuring the viability of future technology nodes [5].

1.1. Figures of merit

Fig. 1 shows examples of the I_D vs. V_G characteristics for a n-type FinFET at two drain biases. Several relevant figures of merit (FoM)

* Corresponding author.
E-mail address: e.comesana@usc.es (Enrique Comesaña).

https://doi.org/10.1016/j.softx.2025.102166
Received 9 January 2025; Received in revised form 28 March 2025; Accepted 14 April 2025
Available online 2 May 2025

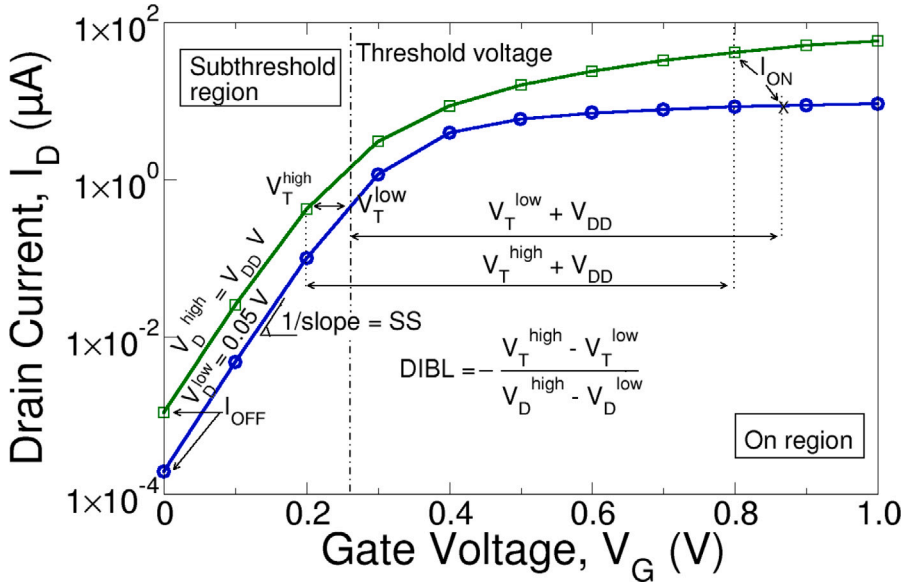


Fig. 1. Drain current (I_D) - Gate voltage (V_G) curves for a generic FET device indicating relevant figures of merit that characterize the device at a low drain bias (V_D) and at a high drain bias (V_{DD}).

are indicated, namely the off-current (I_{off}), sub-threshold slope (SS), threshold voltage (V_{th}), on-current (I_{on}) and drain-induced-barrier lowering ($DIBL$). These FoMs characterize the behavior of transistors, allowing for a fair comparison between different device dimensions, architectures and designs. However, in the literature, there is not an standard that specifies the appropriate extraction criteria to establish the different FoM values [6], which complicates comparisons between works. This becomes particularly relevant for variability studies because, as previously shown [7], the selected extraction criteria influence the standard deviation of the statistical distributions. Therefore, the automatic extraction of the FoMs is essential for a systematic handling of the generated data. In addition, the computational cost of variability studies is becoming prohibitive since the correct modelling of nanometric FETs requires the use of 3D simulators that include quantum effects. Consequently, the development of alternative techniques that allow to reduce simulation times and/or to expand the scope of the investigation is crucial. Recently, machine-learning (ML) close code solutions have been developed in the field of nanoelectronics, aimed to predict the impact of different sources of variability in state-of-the-art transistors [8,9].

1.2. Mlfompy

MLFoMpy is a Python-based package that post-processes and manipulates semiconductor device current–voltage characteristics. MLFoMpy has four main functionalities: (i) automatic extraction of the most relevant FoMs, (ii) calculation of statistical parameters for ensembles of variability-affected devices, (iii) machine-learning based prediction of the different FoMs and full I_D–V_G curves and iv) data visualization tools to plot the analysis results.

Some functionalities of this software have been previously presented [10] and they have been used to extract the threshold voltage values in variability studies of state-of-the-art FETs [11]. Although MLFoMpy was initially developed to work alongside VENDES, an in-house-built 3D variability-enabled semiconductor device simulator [12], it currently also accepts inputs from both experimental data and commercial simulators, such as Silvaco TCAD [13] or Synopsys TCAD [14].

2. Software description

In this section, we describe the software architecture and the software functionalities implemented in the different modules of the tool.

Table 1
Data sources supported by MLFoMpy.

Data source	Description
Text files	Text files containing comma or tab-separated columnar data, managing different file formats produced by simulation softwares.
Local repository	The repositories should be organized following the structure of the examples published in the Zenodo repository submitted by A. Garcia-Loureiro *et al.*[15].
Remote repository	Nextcloud or Owncloud data repositories are supported. Other remotely stored data can be handled with minor software adaptations. The files and directories in the remote repositories should follow the same structure as in the local repositories.

2.1. Mlfompy: Logical architecture

The logical architecture of MLFoMpy is illustrated in Fig. 2. The core of the software is the `MLFoMpy_dataset (fds)` class, which is the element for the data organization. It provides a basic set of methods and properties to obtain information of the data volumetry and integrity. The `fds` objects are the main input/output parameter for an extended set of functions that provide the functionality of the software. The implemented functions are divided into four main groups or modules: data integration, FoM extraction, machine learning (ML) models, and data visualization. This module organization will be explained later in the development architecture section.

The first set of functions, included in the `parser` module, provides the functionality to integrate data from different sources (summarized in Table 1) into a `fds` object. The library includes preprocessing subroutines to check the validity of the imported data because the inputs can be noisy (as in the case of experimental data or Monte Carlo simulations at low biases) or include non finalized simulations. MLFoMpy performs sanity checks, filtering the data by searching for empty files and NaN values, checking the stability of the final current values, or establishing if the acquired input data is enough to extract the different FoMs with adequate accuracy. The input data is also interpolated to increase the precision in the estimation of FoMs.

The `extraction methods` module provides the functionality to extract the different FoM from the data stored in the `fds` object. The details of mathematical methods used to extract the FoM are explained

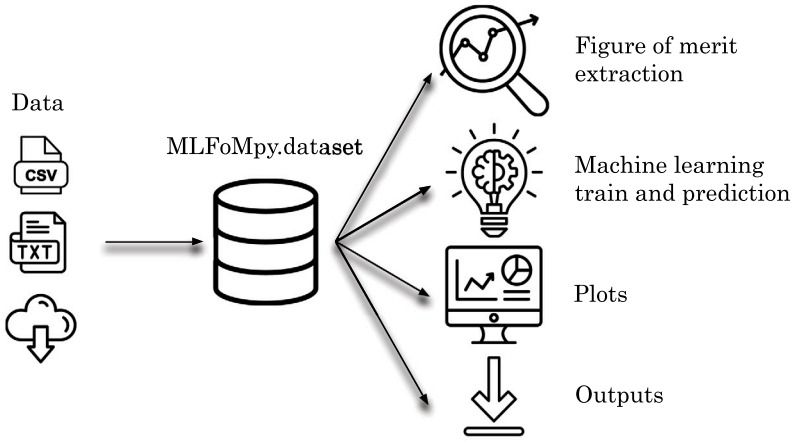


Fig. 2. Main classes and functions of MLFoMPy and their relationships.

in Section 2.4.1. The `machine learning models` module and `machine learning training and testing` module provide a set of tested ML models and methods to train and test them, see Section 2.4.3. These models can be used to predict the electrical response and the FoMs from the physical design parameters of the different devices.

The `plots` module provides a set of functions to plot the data stored in the `fds` object: FoM extraction representations, histograms with variability analyses and ML prediction depiction. The `outputs` module provides the functionality to export and save the data stored in the `fds` object in *.json* format to be used in other software or to be shared with other users. Lastly, the `auxiliary` module provides a set of support functions used across all the other modules: anomalous and validity data detection, interpolation and data completion functions and logging functions.

2.2. Mlfompy process architecture

The process architecture of MLFoMpy is illustrated in Fig. 3. Following the standard workflow, first the user will integrate the source data in a `fds` instance using the parser functions. Then, the FoMs are obtained using the extraction functionality. From the obtained FoMs, plots or output files with the statistical analyses can be produced. The output files, combined with the device variability characteristics, can be used to train and test ML models to predict I–V characteristic and FoMs. Predicted data from ML models can also be imported into new fds objects for further analysis using the MLFoMpy functionality, in the same way as was done with raw source data from simulations. MLFoMPy is implemented as a Python library, therefore, the described workflow, can be created in a Jupyter Notebook or in a Python script. The ML models implemented in MLFoMpy are designed to run in the computer CPU, no special hardware is required like GPUs or TPUs.

2.3. Mlfompy development architecture

The software code and modules are organized in a directory structure, where every module is a Python file with the same name as the module in the directory `src/mlfompy` shown in Fig. 4. The directory and file organization follows the recommendations of the Python Enhancement Proposal 402 (PEP 402) [16] and the Python Packaging User Guide [17]. The software code is written using the recommendations of the Python Enhancement Proposal 8 (PEP 8) [18].

The code is maintained, using the Git version control system, in a public repository on a self-hosted GitLab platform [19]. MLFoMpy users can submit issues, ask for help or propose new features using a form included in the Read the Docs [20] software documentation.

MLFoMPy is mainly developed using a test-driven development methodology [21]. The software has been tested only for the Linux

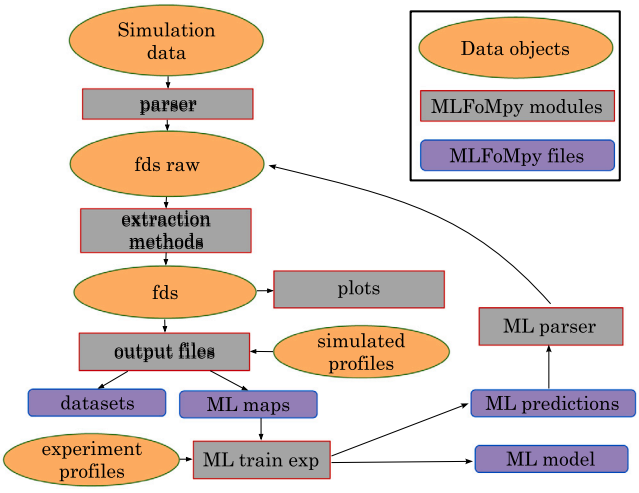


Fig. 3. The process of MLFoMpy shows the standard workflow of the tool and its relationship with the `fds` object. The flowchart also illustrates the software architecture and the interactions between the different modules of the tool.

operating system, using the Python pytest framework [22]. The tests are organized in a `test` directory. The tests are run automatically using the GitLab CI/CD (Continuous Integration/Continuous Delivery) tool.

The software is distributed using the Python Package Index (PyPI) [23] and it is published under the GNU General Public License, version 3 [24]. It can be installed using the Python Package Manager (pip). The main basic dependencies of the software are the Python programming language, version 3.6 or higher, the numpy library [25], the scipy library [26], the matplotlib library [27] and the seaborn library [28]. The ML models are implemented using the scikit-learn library [29], the torch library [30] and the pytorch-lightning library [31]. When installing the software, users have the option to include or exclude the ML model capabilities adding the `[ML]` option in the installation command.

The code documentation is generated using the reStructuredText format. To generate the documentation, the docstrings of the different elements of the software are written following the recommendations of the Python Enhancement Proposal 257 (PEP 257) [32]. In particular, the docstrings of the modules, classes, functions and methods are written using the numpydoc format. The readable documentation can be generated using the Sphinx documentation generator. The documentation is published in the Read the Docs platform [20].

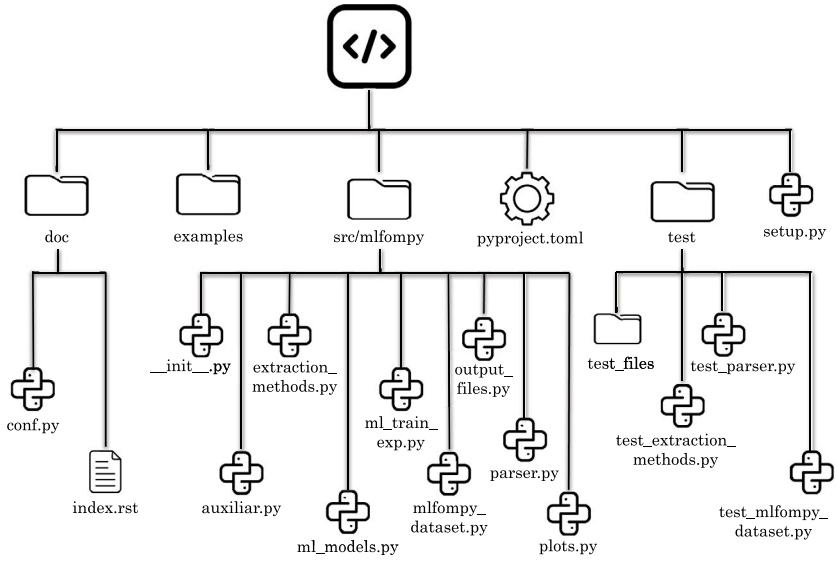


Fig. 4. The diagram depicts the physical architecture of MLFoMpy and highlights the modularity of the tool.

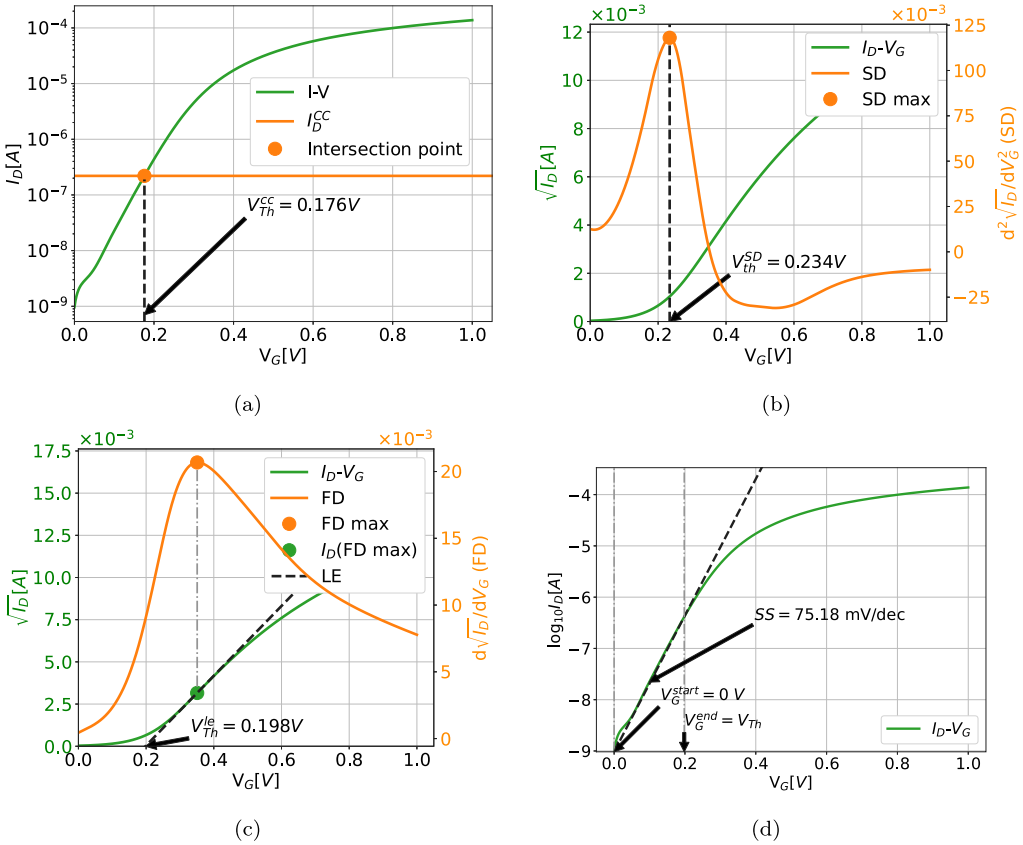


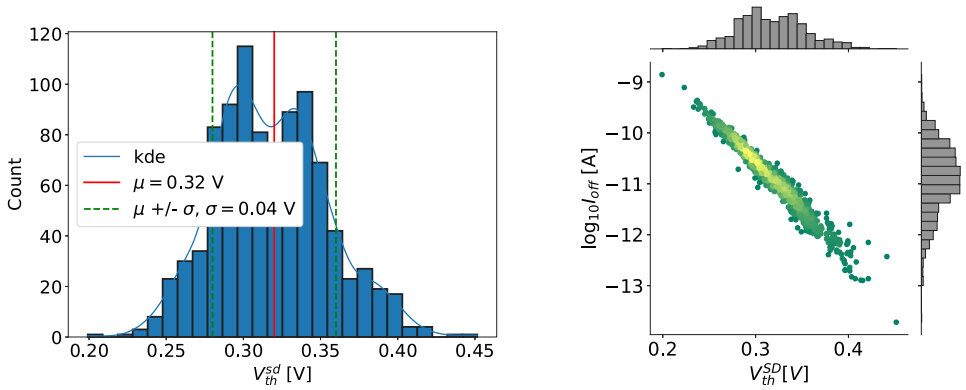
Fig. 5. I_D vs. V_G curves showing the (Fig. 5(a)) constant current (CC) and (Fig. 5(b)) second derivative (SD) threshold voltage (V_{th}) extraction methods and (Fig. 5(c)) the linear extrapolation (LE) threshold voltage extraction method and (Fig. 5(d)) the sub-threshold slope (SS) estimation.

2.4. Software functionalities

This section describes the main functionalities of the MLFoMpy library, including the automatic extraction of FoMs, their statistical analyses, the ML prediction functionality and the data visualization tools.

2.4.1. Automatic extraction of foms in i_D–v_G curves

The software extracts several FoMs that characterize the device behavior from I_D–V_G characteristics. For the extraction of the V_{th}, several methods have been implemented. The constant current (CC), see example in Fig. 5(a), chooses the V_{th} that corresponds with a user-defined value of the drain current, $I_D{}^{CC}$. The second derivative (SD), see example in Fig. 5(b), defines V_{th} as the gate bias for which the



(a) (b)

Fig. 6. Examples of statistical analyses generated by MLFoMPy: (Fig. 6(a)) the distribution of threshold voltage (extracted by the SD method) due to a particular source of variability (including the standard deviation (σ) and the mean value (μ)) and (Fig. 6(b)) I_{off} vs. V_{th} (extracted by the SD method) including the two figures of merit distributions.

derivative $d^2\sqrt{I_D}/dV^2{}_G$ is maximum (expression valid for high drain bias conditions, the equivalent expression implemented for low drain bias can be found in [6]). The linear extrapolation (LE), see example in Fig. 5(c), estimates the maximum of the derivative $d\sqrt{I_D}/dV_G$ (labeled as FD max in the figure), and obtains V_{th} as the V_G axis intercept of the linear extrapolation of the tangent to the $\sqrt{I_D}$–V_G curve at its FD max point (valid for high drain bias conditions, the equivalent expression that has been implemented for low drain bias is available in [6]). The SS, Fig. 5(d), is defined as the inverse of the slope in the linear region (i.e. the subthreshold region) of the I_D–V_G characteristics defined between two user-defined gate bias values, $V_G{}^{start}$ (set to 0.0 by default) and $V_G{}^{end}$ (set to V_{th} by default). The I_{off} is calculated as the drain current at a user-defined gate bias (0.0 V by default), and the I_{on} is dependent on V_{th} and the supply voltage value, V_D, being the drain current for which V_G=V_{th}+V_D.

2.4.2. Statistical analysis for ensembles of variability-affected semiconductor devices

Once the main FoMs that affect the behavior of the devices have been extracted, MLFoMpy automatically performs simple statistical analyses, calculating for instance, the mean value of the distribution and its standard deviation, the fit of the data to the Gaussian distribution or to the kernel density estimation (light blue line in Fig. 6(a)), the skewness and Kurtosis of the distribution, or the Pearson correlation coefficient between different variables. MLFoMPy also includes automatic detection of outlier values, very relevant to avoid errors when processing large datasets.

2.4.3. Machine-learning based prediction of foms and full i_D–v_G curves

For each variability-affected device, the software stores in *.json* format, its I_D–V_G curve, the different extracted FoMs and the variability profile that characterizes the deviation of the device behavior from the ideality. This information will be used as input parameters for the neural networks. This data will be scaled (different methods implemented: StandardScaler, MinMaxScaler, MaxAbsScaler, RobustScaler, QuantileTransformer, and PowerTransformer), and split into three datasets (train, test and validation). A PCA (Principal Component Analysis) is also implemented to reduce the number of relevant features if needed. For the ML training different regressors are implemented, namely Multi-Layer Perceptron (MLP), Linear, Decision Tree, Random Forest and Support Vector Machine. Currently, three ML models are included and optimized in MLFoMpy, two MLPs that allow to predict the FoMs for LER and MGG, and another MLP that allows to estimate the I_D–V_G curves for MGG-affected devices.

For a more detailed discussion on the implementation of the ML models and methods mentioned above, the reader is referred to the software documentation [20]. Comparative studies of the different ML models implemented in MLFoMpy can be found in [33,34]. The first study [33] compares and evaluates various ML approaches for predicting the impact of variability sources on the electrical performance of nanosheet FETs. The second study [34] provides a detailed description of dataset construction, MLP models, hyperparameter tuning, and the training/validation processes. Both studies also include performance analyses, as well as CPU time and memory usage comparison.

2.4.4. Plotting tools

MLFoMpy includes different plotting functionalities, comprising the visualization of the different extraction methods (see examples in Fig. 5), scatter plots between two variables (see example in Fig. 6(b) showing the relationship between the I_{off} and the V_{th} extracted using the SD method), histograms showing the distribution of a particular figure of merit (see example in Fig. 6(a) for the V_{th} extracted using the SD method), or the representation of I_D–V_G characteristics both in linear and logarithmic scales (see examples in Fig. 7 comparing simulation data and ML predictions). In addition, to easily assess the quality of the ML prediction, the software includes scatter plots comparing the simulated values versus the ML predicted ones, showing their correlation via the coefficient of determination R^2.

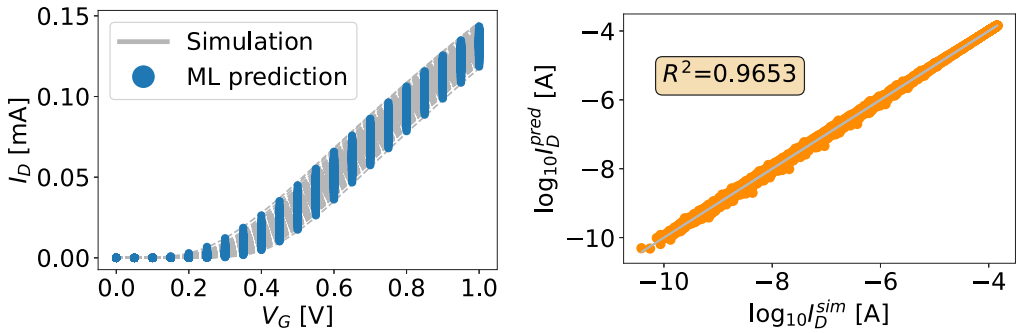
3. Illustrative examples

In this section we present some representative examples of the main functionalities of MLFoMpy. Code snippets are provided to show how to use the software to extract the different FoMs, to perform statistical analysis of the data, or to predict a full I_D–V_G curve using machine-learning functionalities. In the snippets some non-representative code lines are not included. Check the tool documentation for complete versions of the codes.

3.1. Extraction of a FoM from simulated data and statistical analysis

The Listing 1 shows how to extract the V_{th} and the I_{off} from a dataset of simulated I–V curves. The dataset is imported from a set of JCJB simulations. The extraction is performed using the SD method for V_{th} and the drain current at V_{gs} = 0 V for I_{off}, as explained in Section 2.4.1.

The script also shows how to print the basic statistics of the extracted FoMs. The extracted values are stored in a *.json* file for further use. Next, the histograms of the extracted FoMs are generated and saved to disk files as images (Fig. 6(a) for V_{th}). Additionally, the script generates correlation plots between the extracted FoMs (V_{th} and I_{off}, Fig. 6(b)).



(a) (b)

Fig. 7. (7(a)) I_D vs. V_G characteristics plotted with MLFoMPy comparing simulated values and ML-based predictions. (Fig. 7(b)) Drain current ML predictions against simulated values with the coefficient of determination R^2.

```
# IV curves from JCJB (drift-diffusion output)
fds = MLFoMpyDataset()
prs.iv_from_JCJB(fds,path=Path('jcjb_example/'))
# Threshold voltage extraction using the SD method
extraction.threshold_voltage(fds,'SD')
# Off current extraction
extraction.off_current(fds,vg_ext=0.0)
# Printing FoM DD statistics on terminal
aux.print_fom_stats(fds)
# Storing to output file (Figure_of_merit.json)
output.fom_to_json('examples/jcjb_example/',fds)
# Storing to output file (ler_ml_maps.json) that will
# be generated on device path by default
output.ML_fom_to_json_ler('NW',fds,width=7,label='10nm')
# Histogram plots.
plots.hist(fds,'vth','SD')
# Correlation plots between FoMs with their histograms.
plots.fom_correlation(fds,fom1='vth',fom2='ioff')
```

Listing 1 Extraction of the V_{th} and I_{off} from a dataset of simulated I–V curves

3.2. Prediction of the I-v curve using machine learning models

The Listing 2 shows how to predict the I–V curve of a set of FETs using MLP models, as explained in Section 2.4.3.

The input dataset is generated from a synthetic MGG profile generator (see an example of how to generate these profiles in the software repository [19]). The profiles used to train the ML models are simulated to obtain the JCJB data, and the resulting simulated I–V curves are used as input for this example.

The script shows how to load the simulation data and how to split the dataset into training, validation, and test sets. A MLP model is trained and then evaluated using the R^2 score. I–V curves for new configurations are predicted and the results are evaluated again using the R^2 score. Finally, the script generates plots of the predicted vs simulated I–V curve. An example of these plots can be seen in Fig. 7(a).

```
# Import the dataset and dataset preprocessing
# _tr, _va, _te stand for training, validation and test
dt = json.load(open('ml_iv_fom_MGG_GS3.json'))
X, Y = ml_train_exp.input_output_iv(dt)
X_tr,X_va,X_te,Y_tr,Y_va,Y_te = ml_train_exp.split_data(
    X, Y, test_size = 0.2)
# Train and test the MLP model
model = ml_train_exp.mgg_train_test_iv(
    X_tr,X_va,X_te,Y_tr,Y_va,Y_te)
# Predict the IV curves for new configurations
y_predicted = model(X_te)
# Plotting the prediction versus simulated IV curves
plots.prediction_versus_simulation_plot(
    simulation=Y_test_data, prediction=y_hat,
    r2=r2_score(Y_te, y_predicted),
    xlabel='Sim',ylabel='Pred')
plots.iv_curves_simulation_prediction(
    i_simulated=Y_te,i_predicted=y_pr,scale='lin')
```

Listing 2 Prediction of the I–V curve using machine learning for MGG variability

4. Impact

The primary contribution of this work is the development of an integrated platform that allows for a speedy post-processing of data arising from semiconductor device simulations. The user can automatically obtain the relevant FoMs, choosing from different extraction criteria, facilitating future comparisons among scientific works. Using the same tool, the user can plot results and obtain standard statistical analysis of them, and generate input datasets that can be employed for further studies. All these capabilities simplify the tasks that need to be carried out by researchers when analyzing variability-affected semiconductor devices, in which the amount of data that has to be processed is very large. In addition, the incorporation of ML tools in MLFoMpy also allows for new research questions to be addressed that would be too computationally expensive using solely semiconductor device simulation. Some of these questions include the prediction of the effect of different sources of variability [34], or the optimization of beyond the state-of-the-art device structures [35]. Note that, the aforementioned works [34,35] used the resources provided by MLFoMpy.

MLFoMpy is a fast, open-source tool for semiconductor variability analysis, focusing on statistical FoM extraction and ML-based predictions. Unlike Silvaco Victory Analytics [13] and Synopsys Sentaurus [14], which provide full TCAD simulations, MLFoMpy analyzes existing TCAD or experimental data. While commercial alternatives are costly and complex, MLFoMpy is free, user-friendly, and serves as a

complementary tool, accelerating statistical analysis and offering quick insights without requiring full TCAD simulations.

To the authors knowledge, no other tool offers the same capabilities as MLFoMpy. Most statistical analyses in the literature are done manually with ad-hoc scripts, which can be time-consuming and error-prone. We believe MLFoMpy will significantly impact the semiconductor device modeling community by enabling faster, more efficient analyses and addressing new research questions that would otherwise be computationally expensive.

5. Conclusions

We have developed MLFoMpy, a Python-based package aimed at the post-processing of data arising from semiconductor device simulations. The software allows to automatically extract the most relevant figures of merit from current–voltage characteristic curves. It also calculates several statistical parameters for sets of current–voltage curves. Finally, MLFoMpy incorporates machine-learning based prediction tools to estimate figures of merit and full current–voltage curves for semiconductor devices affected by a particular source of intrinsic variability.

Future developments of MLFoMpy will focus on expanding its capabilities by enhancing its machine learning models with deep learning techniques for more advanced pattern recognition and improved device behavior forecasting. These new models will not only extract electrical characteristics and figures of merit but also will allow to analyze intrinsic properties such as electrostatic potential, electric field, carrier densities, and carrier generation/recombination processes. This will provide a more comprehensive understanding of device operation and enable more accurate predictions of performance under various operating conditions.

6. Conflict of interest

No conflict of interest exists: We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no financial support for this work that could have influenced its outcome.

CRediT authorship contribution statement

Enrique Comesaña: Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Julian G. Fernández:** Writing – review & editing, Visualization, Validation, Software, Methodology, Investigation. **Natalia Seoane:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Antonio García-Loureiro:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work presented in this paper is funded by the Xunta de Galicia and FEDER funds (ED431F 2020/008, ED431C 2022/16) and by the Agencia Estatal de Investigación (PID2022-141623NB-I00, PID2022-142709OB-C21/PID2022-142709OA-C22).

References

[1] Iwai H. Impact of micro-/nano-electronics, miniaturization limit, and technology development for the next 10 years and after. ECS Trans 2021;102(4):81–8. http://dx.doi.org/10.1149/10204.0081ecst.

[2] IEEE. International roadmap for devices and systems. 2022, URL https://irds.ieee.org.

[3] Nagy D, Espiñeira G, Indalecio G, García-Loureiro AJ, Kalna K, Seoane N. Benchmarking of FinFET, nanosheet, and nanowire FET architectures for future technology nodes. IEEE Access 2020;8:53196–202. http://dx.doi.org/10.1109/ACCESS.2020.2980925.

[4] Seoane N, Fernandez JG, Kalna K, Comesaña E, García-Loureiro A. Simulations of statistical variability in n-Type FinFET, nanowire, and nanosheet FETs. IEEE Electron Device Lett 2021;42(10):1416–9. http://dx.doi.org/10.1109/LED.2021.3109586.

[5] Amuru D, Zahra A, Vudumula HV, Cherupally PK, Gurram SR, Ahmad A, Abbas Z. AI/ML algorithms and applications in VLSI design and technology. Integration 2023;93:102048. http://dx.doi.org/10.1016/j.vlsi.2023.06.002.

[6] Ortiz-Conde A, García-Sánchez FJ, Muci J, Terán Barrios A, Liou JJ, Ho CS. Revisiting MOSFET threshold voltage extraction methods. Microelectron Reliab 2013;53(1):90–104. http://dx.doi.org/10.1016/j.microrel.2012.09.015.

[7] Espiñeira G, García-Loureiro AJ, Seoane N. Does the threshold voltage extraction method affect device variability? IEEE J Electron Devices Soc 2021;9:469–75. http://dx.doi.org/10.1109/JEDS.2020.3046122.

[8] Akbar C, Li Y, Sung W-L. Transfer learning approach to analyzing the work function fluctuation of gate-all-around silicon nanofin field-effect transistors. Comput Electr Eng 2022;103:108392. http://dx.doi.org/10.1016/j.compeleceng.2022.108392.

[9] Butola R, Li Y, Kola SR. A machine learning approach to modeling intrinsic parameter fluctuation of gate-all-around si nanosheet MOSFETs. IEEE Access 2022;10:71356–69. http://dx.doi.org/10.1109/ACCESS.2022.3188690.

[10] Espiñeira G, Seoane N, Nagy D, Indalecio G, García-Loureiro AJ. Fompy: A figure of merit extraction tool for semiconductor device simulations. In: 2018 joint international EUROSOI workshop and international conference on ultimate integration on silicon. 2018, http://dx.doi.org/10.1109/ULIS.2018.8354752.

[11] Fernandez JG, Seoane N, Comesaña E, García-Loureiro A. Pelgrom-based predictive model to estimate metal grain granularity and line edge roughness in advanced multigate mosfets. IEEE J the Electron Devices Soc 2022;10:953–9. http://dx.doi.org/10.1109/JEDS.2022.3214928.

[12] Seoane N, Nagy D, Indalecio G, Espiñeira G, Kalna K, García-Loureiro AJ. A Multi-Method Simulation Toolbox to Study Performance and Variability of Nanowire FETs. Materials 2019;12(15):2391–406. http://dx.doi.org/10.3390/ma12152391.

[13] Silvaco TCAD. 2023. URL https://silvaco.com/tcad.

[14] Synopsys TCAD. 2023. URL https://www.synopsys.com/manufacturing/tcad.html.

[15] Garcia-Loureiro A, Seoane N, Fernandez JG, Comesaña E, Pichel JC. Figures of merit that characterize silicon gate- all-around nanowire FETs affected by line edge roughness variability. 2023, http://dx.doi.org/10.5281/zenodo.7674909.

[16] Smith EV. PEP 420 - implicit namespace packages. 2012, URL https://peps.python.org/pep-0420/.

[17] Python Packaging Authority. Python packaging user guide. 2021, URL https://packaging.python.org/.

[18] van Rossum G, Warsaw B, Coghlan N. PEP 8 - style guide for python code. 2001, URL https://peps.python.org/pep-0008/.

[19] Fernández JG, Comesaña E, Seoane N, García-Loureiro AJ. MLFoMpy - GitHub repository. 2023, URL https://gitlab.citius.gal/modev/mlfompy.

[20] Fernández JG, Comesaña E, Seoane N, García-Loureiro AJ. MLFoMpy's documentation. 2003, URL https://mlfompy.readthedocs.io/.

[21] Beck K. Test-driven development: by example. Addison-Wesley signature series, Addison-Wesley; 2003.

[22] Krekel H, Oliveira B, Pfannschmidt R, Bruynooghe F, Laugher B, Bruhin F. pytest 8.2. 2004, URL https://github.com/pytest-dev/pytest.

[23] Fernández JG, Comesaña E, Seoane N, García-Loureiro AJ. MLFoMpy - PyPI project page. 2023, URL https://pypi.org/project/mlfompy/.

[24] Free Software Foundation. Gnu general public license v3.0. 2007, URL https://www.gnu.org/licenses/gpl-3.0.html.

[25] Harris CR, Jarrod Millman K, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[26] Virtanen P, Gommers R, Oliphant TE, et al. SciPy 1.0: Fundamental algorithms for scientific computing in python. Nature Methods 2020;17(3):261–72. http://dx.doi.org/10.1038/s41592-019-0686-2.

[27] Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(3):90–5. http://dx.doi.org/10.1109/MCSE.2007.55.

[28] Waskom ML. seaborn: statistical data visualization. J Open Source Softw 2012;6(60):3021. http://dx.doi.org/10.21105/joss.03021.

[29] Pedregosa F, Varoquaux G, et al. Scikit-learn: Machine learning in python. J Mach Learn Res 2011;12(85):2825–30, URL https://dl.acm.org/doi/10.5555/1953048.2078195.

[30] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, et al. PyTorch: An imperative style, high-performance deep learning library. In: 33rd international conference on neural information processing systems, vol. 721, 2019, p. 8026–37, URL https://dl.acm.org/doi/10.5555/3454287.3455008.

[31] Falcon W, The PyTorch Lightning team. Pytorch lightning. 2019, URL https://www.pytorchlightning.ai/.

[32] Goodger D. PEP 257 - docstring conventions. 2001, URL https://peps.python.org/pep-0257/.

[33] García-Loureiro A, Seoane N, Fernández JG, Comesaña E, Pichel JC. A machine learning approach to model the impact of line edge roughness on gate-all-around nanowire FETs while reducing the carbon footprint. PLoS ONE 2023;18(7). http://dx.doi.org/10.1371/JOURNAL.PONE.0288964.

[34] Fernandez JG, Comesaña E, Seoane N, Pichel JC, García-Loureiro AJ, Bescond M. An accurate neural network model to study threshold voltage variability due to metal grain granularity in nanosheet FETs. In: 9th joint international EuroSOI workshop and international conference on ultimate integration on silicon. 2023.

[35] Fernandez JG, Etesse G, Comesaña E, Seoane N, García-Loureiro AJ, Bescond M. Optimization of thermionic cooling semiconductor heterostructures with deep learning techniques. In: International conference on simulation of semiconductor processes and devices. 2023.