On the Use of Artificial Neural Networks for the Automated High-Level Design of ΣΔ Modulators

Pablo Díaz-Lobo, Gustavo Liñán-Cembrano and José M. de la Rosa, *Fellow, IEEE*

***Abstract*—This paper presents a high-level synthesis methodology for Sigma-Delta Modulators (ΣΔMs) that combines behavioral modeling and simulation for performance evaluation, and Artificial Neural Networks (ANNs) to generate high-level designs variables for the required specifications. To this end, comprehensive datasets made up of design variables and performance metrics, generated from accurate behavioral simulations of different kinds of ΣΔMs, are used to allow the ANN to learn the complex relationships between design-variables and specifications. Several representative case studies are considered, including single-loop and cascade architectures with single-bit and multi-bit quantization, as well as both Switched-Capacitor (SC) and Continuous-Time (CT) circuit techniques. The proposed solution works in two steps. First, for a given set of specifications, a trained classifier proposes one of the available ΣΔM architectures in the dataset. Second, for the proposed architecture, a Regression-type Neural Network (RNN) infers the design variables required to produce the requested specifications. A comparison with other optimization methods – such as genetic algorithms and gradient descent – is discussed, demonstrating that the presented approach yields to more efficient design solutions in terms of performance metrics and CPU time.**

***Index Terms*—Design Automation, Optimization, Neural Networks, Analog-to-Digital Converters, Sigma-Delta Modulation.**

I. Introduction

SIGMA-Delta Modulators (ΣΔMs) are one of the best techniques to implement Analog-to-Digital Converters (ADCs) in a wide range of applications – from instrumentation, biomedical devices and automotive sensors to communications. In spite of their potential advantages in terms of robustness and efficiency, the design of high-performance ΣΔMs involves a number of tasks carried out in a top-down/bottom-up design flow from systems to chip, which requires a certain degree of know-how and expertise. In most cases, the main design bottleneck is at the system level, where finding the best ΣΔM architecture for a given set of specifications, and mapping those specifications onto circuit-level electrical design parameters become crucial to get a good design choice [1], [2].

Over the years, a number of design methodologies and CAD tools have been presented to automate the design of ΣΔMs. These tools are mostly focused on optimizing system-level tasks – such as architecture selection, loop-filter design, behavioral modeling, simulation and sizing, as well as electrical design and validation [3]–[9]. The most general approach is based on the so-called optimization-based synthesis method, where an optimizer is combined with a performance evaluator (usually a simulator) to find the optimum design. A number of optimization algorithms have been proposed to this end, including genetic, simulated annealing, or multi-objective evolutionary Pareto fronts, to cite a few [3], [4], [7], [10].

Recent works demonstrate that Artificial Intelligence (AI) algorithms can be applied to automate, or to assist, analog circuit design [11]–[19]. Their use in ΣΔMs has been employed to improve the performance metrics of ΣΔMs and other ADCs by means of linearization or calibration techniques based on Artificial Neural Networks (ANNs) [20], [21]. Some authors have proposed using ANNs in an optimization-based synthesis methodology [11]–[13], [15], [20], [22]. In some of them, the ANN has been trained to replace the simulator, while other approaches consider ANNs as an optimization engine. In the latter case, the ANN is trained to size a given system for a set of specifications. Thus, the ANN should be trained with sized solutions known from prior optimized designs. Once the ANN is trained, it is able to automate the sizing process and generate optimum design solutions for new sets of specifications which were not considered in the training dataset. This method has been successfully applied to design essential analog circuits such as operational amplifiers [11].

In our initial publication on this topic [23], we investigated the use of manually designed neural networks in deriving high-level design variables for some ΣΔMs architectures. Here we present the evolution of this initial concept into a self-contained comprehensive framework for automatic architecture selection and high-level design. In this framework, neural networks with optimized architectures, combined with a library of classifiers topologies, are the core of a methodology that, starting from very high-level and general specifications, can swiftly provide users with a suitable topology and its corresponding high-level design variables in negligible time (less than 50 milliseconds in our simulations). We demonstrate the versatility of our approach, applicable to either single-bit or multi-bit quantization, single-loop or cascade configuration, discrete or continuous time operation, with a case study that embeds an extensive dataset encompassing various ΣΔM architectures, including 2nd-order SC-ΣΔM, 3rd- and 4th-order cascade 2-1 and 2-1-1 SC-ΣΔM, as well as a 2nd-order Gm-C ΣΔM. The networks have been automatically optimized and trained to learn the complex mapping from specifications onto high-level design variables, enabling them to predict the most suitable set of design variables for a

Manuscript submitted August 16, 2023; reviewed October 7, 2023.

This work was supported in part by Grants PID2019-103876RB-I00, PID2022-138078OB-I00, funded by MCIN/AEI/10.13039/501100011033, by the European Union ESF Investing in your future, by ERDF A way of making Europe, and by "Junta de Andalucía" under Grant P20-00599.

Pablo Díaz-Lobo, Gustavo Liñán-Cembrano and José M. de la Rosa are with the Institute of Microelectronics of Seville, IMSE-CNM (CSIC/University of Seville), Parque Tecnológico de la Cartuja, C/ Américo Vespuccio 28, 41092 Sevilla, SPAIN, e-mail: [pabdiaz,jrosa]@imse-cnm.csic.es, gustavo.linan@csic.es

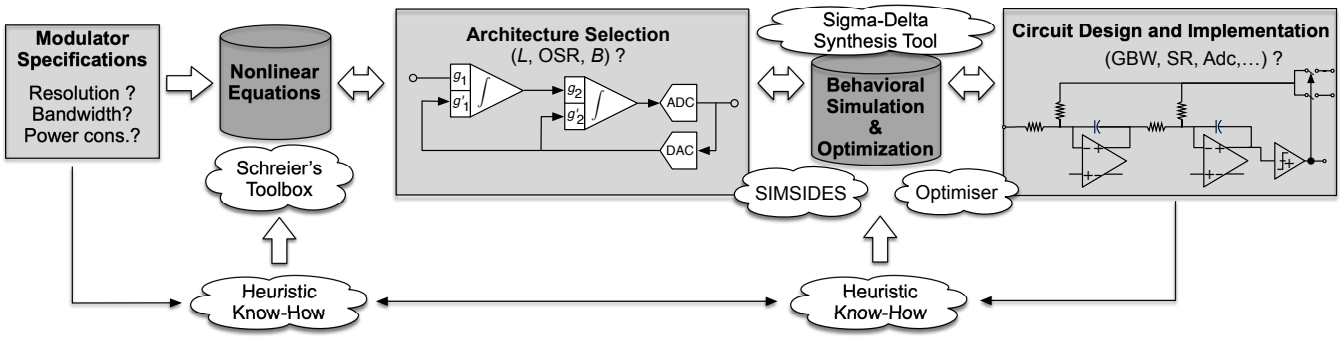


Fig. 1. Usual design flow diagram applied to the design of ΣΔMs.

given set of specifications. The results achieved with our proposed method are compared with those obtained using other optimization algorithms, demonstrating very significant improvements in terms of the CPU time.

The paper is organized as follows. Section II revisits prior art on design methodologies and EDA tools for ΣΔMs, with emphasis on the optimization-based system-level sizing procedure. Section III describes the proposed ANN-based high-level synthesis methodology. Section IV presents the results for some case studies encompassing four different modulator architectures. Section V discusses the comparison with other existing approaches. Finally, conclusions are drawn in Section VI.

II. Background and Prior Art on Optimization-based Synthesis of ΣΔMs

The design methodology of ΣΔMs follows the well-known hierarchical *top-down/bottom-up* approach. In this *divide-and-conquer* strategy, a ΣΔM is partitioned into several *abstraction* levels (system-level, building block-level, circuit-level and device/physical-level), so that at each abstraction level, a design (or *sizing*) process takes place, to transmit (or to *map*) the system specifications in a hierarchical way – from ΣΔM architecture to circuit [24].

This process is conceptually depicted in Fig. 1 for the different tasks involved in the system-level design of ΣΔMs: from specifications to building-block/circuit-level implementation. Starting from the ΣΔM *specifications*; i.e. Effective Number of Bits (ENOB) and signal bandwidth (BW), the first design problem consists of finding out the best modulator architecture that meets such specifications with the minimum power consumption. At the first step, ideal design equations – without including the effect of circuit errors – of the Noise Transfer Function (NTF) and ENOB are used to get an estimation of main ΣΔM system-level parameters, i.e. OverSampling Ratio, OSR, loop-filter order, L, and number of bits of the embedded quantizer, B. Once these parameters are known, the ΣΔM is analyzed by using more accurate nonlinear model equations. Schreier's MATLAB® ΣΔ-toolbox is widely used to do this task [25]. This toolboox is a collection of MATLAB scripts that include nonlinear models of ΣΔMs, which can be applied to *automatically* synthesize a ΣΔM Loop Filter (LF), providing the desired noise shaping while keeping system stability [1], [9].

Once the ΣΔM architecture has been selected, the next design task is to analyze the influence of circuit nonidealities to obtain the specifications of main ΣΔM building blocks, i.e. the amplifiers, comparators, switches, etc. To this purpose, an accurate – but computationally efficient – performance evaluator, usually a simulator, is used. The modulator needs to be modeled at a high abstraction level, albeit taking into account the effect of nonideal physical phenomena at circuit and device level.

An efficient technique to do this is the so-called behavioral modeling approach, which allows to simulate ΣΔMs in a fast but precise way [4]. Fig. 1 highlights some exemplary behavioral simulators which are normally used by ΣΔM designers. One of them is the so-called SIMSIDES [8]– a time-domain simulator developed in the MATLAB/SIMULINK environment and available at `www.imse-cnm.csic.es/simsides`. This tool is able to simulate any arbitrary ΣΔM, by considering the effect of main circuit nonidealities. Behavioral models included in SIMSIDES have been verified by electrical simulations and experimental measurements from state-of-the-art chips in order to guarantee the accuracy of the simulations. Behavioral models of building blocks are implemented in C-coded MATLAB S-functions to keep high accuracy and high computational efficiency – typically a 2^{16} clock-cycle simulation takes a few seconds CPU time [4].

Another widely used system-level simulation approach – mostly focused on CT-ΣΔMs – is based on the so-called *lifting* method [7]. This approach allows to speed up the simulation by using Discrete-Time (DT) solvers and hardware (GPU) acceleration, drastically reducing the simulation time – typically hundreds or thousands of simulations are done within seconds – while keeping high accuracy. Based on this technique, a web-based tool named Sigma-Delta Synthesis Tool [26] is available at `www.sigma-delta.de`. This tool also provides a design automation environment for CT-ΣΔMs, including Signal Transfer Function (STF) engineering and automated LF scaling. Behavioral simulators are usually guided by an optimization engine to explore the multi-dimensional design space and automate the high-level sizing process. The final objective is to find the optimum solution, i.e. the best set of design variables (building-block performance parameters), that satisfy the ΣΔM ADC specifications with the minimum power

dissipation. This paper proposes replacing the optimizer–behavioral simulator system with neural networks that will learn, from extensive datasets, how to map a point P within the specifications space onto a point Q in the high-level design-variables spaces. This work assumes that the first steps of the synthesis procedure shown in Fig. 1, i.e. the internal architecture of a given modulator type, the scaling of loop-filter coefficients, Out-of-Band Gain (OBG), etc. are given. However, ANNs might also be applied to this purpose. The procedure – beyond the scope of this paper – would essentially be the same as that followed in this work but applying the sizing process at a higher abstraction level. In such a case, the design variables would be the loop-filter coefficients, and the $\Sigma\Delta$M subcircuits (integrators, resonators, comparators, etc.) should be considered as ideal blocks.

III. Proposed Methodology

A. Problem Definition

As stated in previous section, the high-level design of a $\Sigma\Delta$M involves solving two types of problems. First, for a given vector of specifications, $\bar{\Gamma}$, a suitable architecture has to be selected from a family of alternatives $\{A_j\}$, where j represents available topologies. In our case, specifications typically involve ENOB, BW, Signal-to-Noise Ratio (SNR), Signal-to-Noise and Distortion Ratio (SNDR), Total Harmonic Distortion (THD), a Figure of Merit (FOM)[1], etc., but they might also include constrains imposed by higher hierarchy considerations if the modulator is to be embedded in a more complex system (e.g. forcing discrete-time vs. continuous time operation or vice-versa, defining a maximum area occupation, setting a maximum power consumption, etc.), and even include constrains imposed by application-related scenarios (i.e. military, industrial, space, automotive, mass market,...). Based on the selected topology, a vector of architecture-dependent design variables $\bar{\epsilon}(A_j)$ needs to be found in such a way that specifications $\bar{\Gamma}$ are not only met but also optimized according to some predefined metrics.

B. Proposed Solution

For the first problem, finding a suitable architecture from the information in our dataset, we propose to consider it as obtaining an inference from a trained classifier, C, which maps system performance metrics ($\bar{\Gamma}_i$) into a categorical variable A_j (the selected architecture). The process of selecting and training the classifier is described in detail in Section III-D. The second problem must yield the design variables which guarantee that the received set specifications are met for the chosen $\Sigma\Delta$M architecture. Here we propose utilizing Regression-type Neural Networks (RNNs), trained on comprehensive datasets, to infer these design variables. Hence, our methology, detailed in the flow charts in Fig. 2, can be basically described by these two operations:

$$A_j = C(\bar{\Gamma}_i);\ \bar{\epsilon_k} = RNN(\bar{\Gamma}_i, A_j) \tag{1}$$

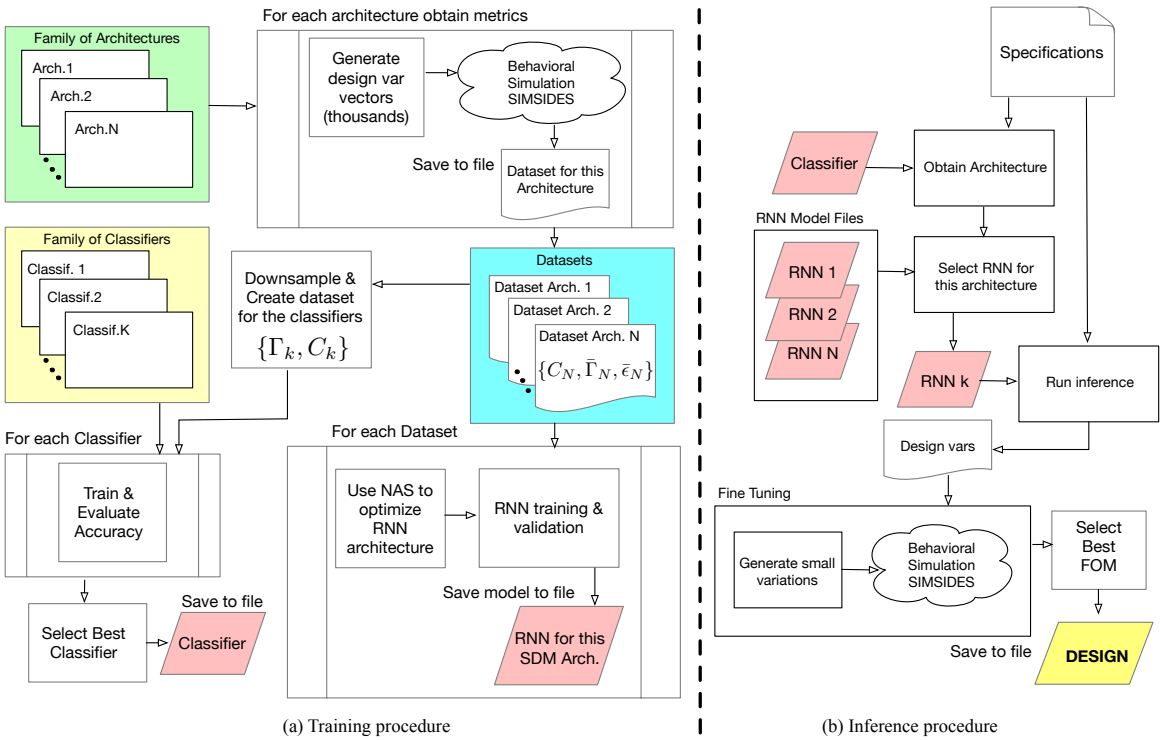
[1]The Schreier's Figure of Merit – denoted as FOMS [1] and widely adopted by $\Sigma\Delta$M community – will be used in this work.

Where $C(\cdot)$ is a call to the classifier to obtain an architecture, and $RNN(\cdot)$ is a call to the neural network inference process to generate the design variables. Section III-E provides detailed information on how we propose to optimally design and train the regression neural networks. Utilizing pre-trained classifiers and networks offers the significant advantage of reducing the inference time, i.e., the time required to select a modulator architecture and obtain its high-level design variables from a new set of specifications, to just a few milliseconds. Furthermore, memory limitations and CPU time-related concerns are alleviated as they pertain to the dataset generation and training processes, which need to be executed only once or infrequently when incorporating new datasets/architectures. However, the primary drawback of this approach is that the quality of the obtained solution heavily relies on the accuracy (in a broad sense) of both the classifier and the networks, which, in turn, are highly dependent on the quality of the available dataset [27].

C. Designing the Dataset

Our proposal is to format every input d_i in the dataset as a triplet of the form $\{C_i, \bar{\Gamma}_i, \bar{\epsilon_i}\}$ where C_i is a categorical variable which defines the architecture of the modulator, $\bar{\Gamma}_i$ is a vector of $\Sigma\Delta$M performance metrics, (i.e., a point in the specifications space) and $\bar{\epsilon_i}$ is a vector which contains the design variables (i.e., a point in the design-variables space) that produced such metrics in a behavioral simulation of this modulator architecture using SIMSIDES [2], [4]. Regardless of the specific modulator architecture, $\bar{\Gamma}_i$ must always be made up of the same performance metrics for all the architectures being incorporated into our framework. This is simply justified by the fact that the classifier needs to operate on $\bar{\Gamma}$ across the whole dataset in order to find the best-suited solution for a particular set of specifications $\bar{\Gamma}_i$. However, different modulators will require different high-level design variables to be found. Thus, if a unified $\bar{\epsilon_i}$ format is to be used for all the architectures under consideration, we will need to define the $\bar{\epsilon_i}$ vector with as many elements as there are different design variables across all the modulators in our dataset, forcing us to pad with "N/A" (Not Applicable), and handle it, all inputs that do not correspond to the specific modulator in each particular dataset entry. Though this solution offers the advantage of using a single dataset file, it lacks scalability as it necessitates recreating this file every time a new architecture is added to our framework and retraining the classifier and the RNN from scratch. Alternatively, one can choose to create a specific dataset in the form of $\{C_i, \bar{\Gamma}_i, \bar{\epsilon_i}\}$ for each modulator, where C_i remains the same for all inputs in each particular dataset. Individual networks can then be tailored for each case, optimizing them specifically for their corresponding architectures. This approach enables us to optimize a network for each modulator, training it solely on specific $\{\bar{\Gamma}_i, \bar{\epsilon_i}\}$ data, rather than seeking an optimum and more complex solution that handles the entire dataset encompassing all architectures. Thus, (1) suffers an apparently simple but conceptually crucial transformation becoming:

$$A_j = C(\bar{\Gamma}_i);\ \bar{\epsilon_k} = RNN_{A_j}(\bar{\Gamma}_i) \tag{2}$$



(a) Training procedure (b) Inference procedure

Fig. 2. Flow diagrams for the proposed methodology. (a) Training and dataset preparation. (b) Inferring a design from specifications.

where, now, the regression operation $RNN(\bar{\Gamma}_i, A_j)$ in (1) has been transformed in $RNN_{A_j}(\bar{\Gamma}_i)$, i.e, the role of the classifier is selecting a specific network from a library of already trained ones, and none of the networks receive the categorical architecture variable as input. As far as the training of the classifier is concerned, it will only require reading the $C_i, \bar{\Gamma}_i$ information for each modulator in the dataset and simply concatenate the information before training. In our opinion, having specific datasets for each architecture is a better solution, considering both scalability and neural network performance. Thus, adding a new modulator architecture to the framework only requires creating its particular dataset in the behavioral simulator, training the classifier, and optimizing and training its associated network, which is preferable to creating a completely new dataset file by augmenting the number of columns and adding "N/A" in all the previously existing columns, and then training again a single network whose complexity grows as new architectures are added.

D. Considerations on the Classifier Definition

Selecting an appropriate architecture for a classifier is a crucial step in machine learning and data analysis. The choice of architecture directly impacts the model's performance and its ability to generalize well on unseen data. However, determining the optimal classification technique is not a one-size-fits-all process. It heavily depends on various factors, including the characteristics of the dataset, statistical properties of the data, and the specific problem at hand [28]. The dataset plays a central role in architecture selection, as its size, complexity, and distribution of classes influence the model's behavior. Imbalanced datasets, where certain classes are significantly more abundant than others, pose a challenge as they can lead to biased predictions towards the majority classes [29]. In our case, we propose to always feed the classifier with a subsampled version of the data available for each of the architectures in our framework in such a way that all classes will have same presence. Moreover, the statistical properties of the data, such as feature correlation, co-linearity, dimensionality, and noise, also influence the architecture selection. Since we are not *a-priori* imposing any limitation on which data is to be present in the metrics vector $\{\bar{\Gamma}\}$ (until a case study is presented in Section IV), we have implemented a procedure which explores different classifier types and returns the classifier featuring a better confussion matrix for the particular dataset under consideration. Our solution currently covers: Quadratic and Linear Discriminant Analysis [30], Support Vector Machines SVM [31] with linear, polynomial, and radial-basis function kernels, Gaussian and Multinomial Naive-Bayes [28], Decission Trees [32], Random Forest [33], Gradient Boosting Classifiers [34], and neural network-based classifiers [35].

E. Network Architecture Search and Model Optimization

The backbone of our solution consists of utilizing an optimized multivariate regression network [36] for each modulator architecture to map system metrics (specifications) onto high-level design variables, making the network to play the role

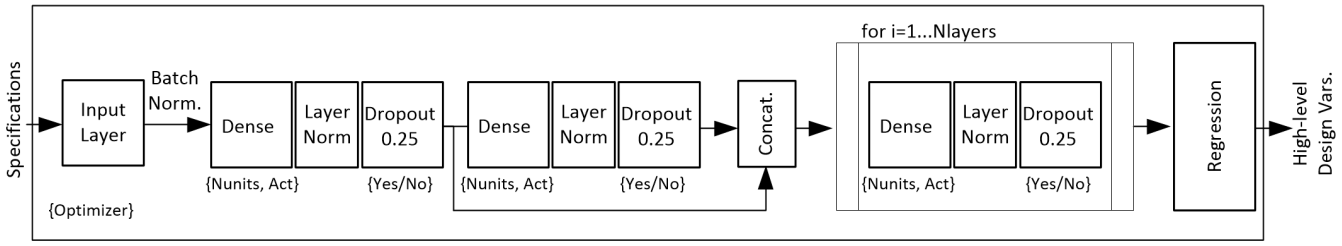


Fig. 3. Network Template Architecture which is optimized using NAS

of the optimizer in the conventional approach depicted in Fig. 1. However, the problem of manually defining a network architecture which minimizes the regression error[2] usually implies a tedious and time-consuming process where network parameters such as number of layers, neurons per layer, etc., are varied in an iterative way until a solution is accepted. To overcome this, we have integrated Neural Architecture Search (NAS[3]) [37] techniques in our design framework to automatically explore the hyper-parameters space and identify a network architecture for each modulator using the Keras tuner API [38], [39]. Our solution first defines a parameterized network template, shown in Fig. 3, and a search space for the following hyper-parameters:

- Number on neurons (units) per layer: $\in [32, 64]$ step 4.
- Number of additional layers after the concatenation block in Fig. 3: $\in [1, 6]$ step 1.
- Activation function: $\in [relu, tanh]$, where $relu$ stands for Rectified-Linear Unit and $tanh$ stands for hyperbolic tangent.
- Dropout layers [40]: Whether to insert dropout layers between dense layers or to bypass them.
- Optimizer: $\in \{Adam, SGD, RMSprop, Adadelta\}$.

and then runs the grid-search optimization until the search space has been exhausted or an *early-stop* condition[4] occurs. Once the hyper-parameters for each network are available, the fitting process takes place using a 80% - 20% division for the training and validation subsets (TS, VS). For any existing $\{\bar{\Gamma}_k, \bar{\epsilon}_k\}$ pair in the training dataset (TS) of a particular $\Sigma\Delta$M architecture, the RNN produces a regression output $\bar{\epsilon}_k^{'}$ such that, overall, during the training process, a loss function (MSE in our case) $E_{TS} = f(\bar{\epsilon}_k, \bar{\epsilon}_k^{'}), k \in TS$ is minimized. Then, the network validation process checks for over-fitting[5] by evaluating this same metric E over the unseen data in the validation set (VS); $E_{VS} = f(\bar{\epsilon}_j, \bar{\epsilon}_j^{'}), j \in VS$, letting us guarantee that whenever $|E_{TS} - E_{VS}| < \delta$, with delta a predefined threshold, the network has been able to generalize the problem with an accuracy E_{VS}.

[2]Assuming a dataset with $j = 1...M$ inputs, and a $i = 1...N$-elements target y_i and output vectors $\hat{y}_i$, then $MSE = \frac{1}{N \cdot M} \sum_{i=1}^{N} \sum_{j=1}^{M} (y_{i,j} - \hat{y}_{i,j})^2$.

[3]NAS is a technique that automatically explores and finds the best-performing neural network architecture for a specific task.

[4]Meeting specifications or not improving the solution in a predefined number of iterations known as the *patience* parameter.

[5]It occurs when a neural network becomes overly specialized in capturing noise and random fluctuations in the training set so that it performs exceptionally well on its training data but fails to generalize to new, or unseen data.

F. Cross-Validation and Result Improvement

Unfortunately, using the MSE to quantify the accuracy of the networks only guarantees that the errors between the inferred design variables vector $\bar{\epsilon}_k^{'}$ and the target design variables $\bar{\epsilon}_k$ are, on average, bounded by the MSE. However, due to the highly non-linear nature of $\Sigma\Delta$Ms, it does not provide a clear insight about how closely the performance metrics $\bar{\Gamma}_k^{'}$ obtained for the inference design variables $\bar{\epsilon}_k^{'}$ match the requested ones $\bar{\Gamma}_k$. To address this concern, our methodology incorporates a final step which involves using the behavioral simulator [4] not only for cross-validation but also for result improvement. In order to do so, we first create a small number (n_{iters}) of slightly varied versions of the solution provided by he network $\bar{\epsilon}_k^{'}$ where each of its components ($DV_n, n = 1...N_{vars}$)[6] is randomly modified by adding a small amount taken from an uniform distribution within $\pm r(\%) \times 100$ range:

$$DV_{n,a} = DV_n\,(1 + \alpha_{n,a})\,, \text{ where } \begin{array}{l} a = 1 \ldots n_{\text{iters}} \\ n = 1 \ldots N_{vars} \\ \alpha_{n,a} \sim U(-r, r) \end{array} \quad (3)$$

Essentially, this cloud of points represents a random sampling of a small, well-defined region within the design-variables space around the point selected by the network where the average distance of these additional points to the original solution is constrained to be less than the MSE. Afterwards, we employ the behavioral simulator to evaluate the resulting performances not only for the point predicted by the network $\bar{\epsilon}_k^{'}$ but also for the cloud of random points around it.$\{\bar{\epsilon}_{k,a}^{'}\}_{a=0...n_{\text{iters}}}$:

$$\{\bar{\Gamma}_{k,a}^{'}\} = BS(A_j, \{\bar{\epsilon}_{k,a}^{'}\}) \quad (4)$$

where A_j is the selected modulator architecture, $BS(\cdot)$ is a call to the behavioral simulator, and $\{\bar{\epsilon}_{k,0}^{'}\} \equiv \bar{\epsilon}_k^{'}$.

Finally, we select the high-level design point which produces the best Schreier's FOM [1]. Conceptually, our approach can be described as a two-step process. First, the neural network provides a coarse solution for the design variables which serves as a starting point. In the second step, a small random search is conducted around this initial solution to identify similar solutions featuring better FOM. Clearly, this second step is open to further development by leveraging advanced optimization solutions, such as gradient-driven methods or simulated annealing, among others, representing an excellent opportunity for future improvements of our design framework.

[6]DV stands for Design Variable and N_{vars} is the number of design variables.

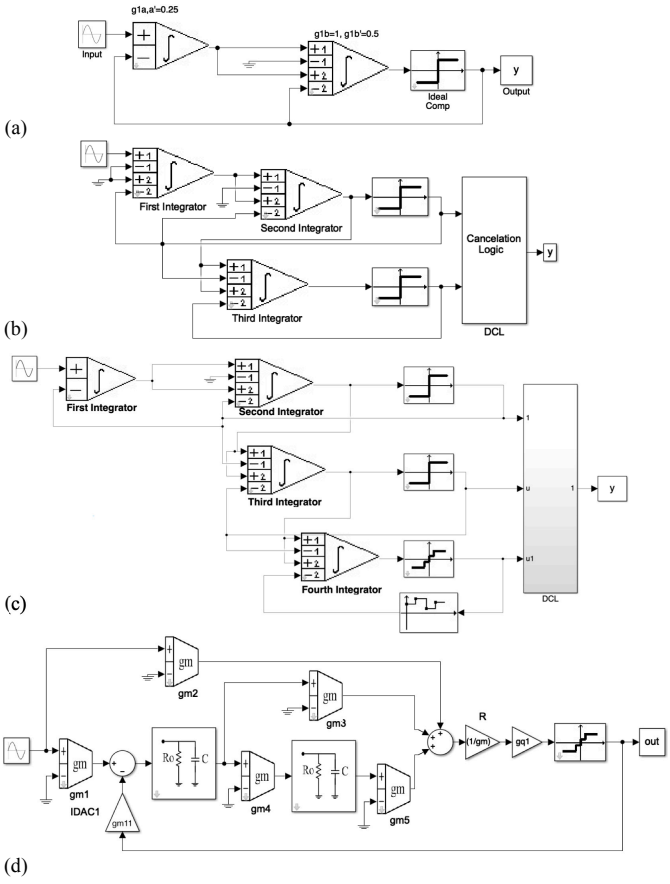


Fig. 4. SIMSIDES models of the ΣΔMs used as case studies: (a) 2nd-order SC-ΣΔM (I). (b) 3rd-order cascade 2-1 SC-ΣΔM (II). (c) 4th-order cascade 2-1-1 SC ΣΔM with 3-bit quantization (III). (d) 2nd-order Gm-C ΣΔM with 3-level quantization (IV). (All models include main circuit nonidealities such as limited input/output swings, thermal noise, finite DC gain, GBW, maximum output current, nonlinear transconductance, etc. [2]).

IV. Case Study

The versatile design methodology proposed in this paper can be applied to any arbitrary ΣΔM. Without loss of generality, let us consider four case-study architectures which include single-loop and cascade loop-filter topologies, single-bit and multi-bit quantization as well as switched-capacitor and continuous-time circuit techniques. Fig. 4 shows the SIMSIDES diagrams of the ΣΔMs under study: (I) a 2nd-order SC-ΣΔM (Fig. 4(a)), (II) a 3rd-order cascade 2-1 SC-ΣΔM (Fig. 4(b)), (III) a 4th-order cascade 2-1-1 SC-ΣΔM with 3-bit quantization (Fig. 4(c)) and (IV) a 2nd-order 3-level CT-ΣΔM based on Gm-C integrators. Regardless of the architecture, the specifications vector $\bar{\Gamma}$ comprised the required modulator resolution, in terms of the SNR, the signal bandwith BW (expressed by the OSR), and the power consumption (Pow) $\bar{\Gamma}_k \equiv \{\text{SNR}_k, \text{OSR}_k, \text{Pow}_k\}$. Conversely, the design variables vector $\bar{\epsilon}_k$ were obviously different for each architecture. In order to generate this case study' dataset, we ran behavioral simulations of each architecture under consideration using the design variables and ranges listed in Table I for five values of the OSR, namely: OSR$= 32, 64, 128, 256, 512$. For two of the cases, the 2nd-ord Gm-C-ΣΔM and the 3th-ord 2-1 SC-ΣΔM, the variables space was covered using nested loops, whereas for the other two cases design variables were generated randomly within meaningful predefined ranges. Moreover, for the 2-1-1 cascade architecture (case III), we considered the design variables of the first amplifier, and defined the other three amplifiers' parameters as a scaled version of those of the first one. The dataset generation involved evaluating more than 200,000 random designs. The combined use of behavioral simulation (SIMSIDES), GPU acceleration and the MATLAB Parallel Computing Toolbox™ allowed us to obtain results at a rate of about 10,000 simulations per hour of CPU time[7]. Needless to mention, since we were generating random designs, not all combinations produced simulation results that were valid for classifiers and networks training as they featured poor SNR. Thus, prior to the training process, the dataset was filtered to remove designs where SNR<50dB. The final number of elements in each dataset is listed in the *Valid Points* column in Table I, amounting to nearly 120,000 $\{C_k, \bar{\Gamma}_k, \bar{\epsilon}_k\}$ (*Class, Specifications, Design Vars.*) entries.

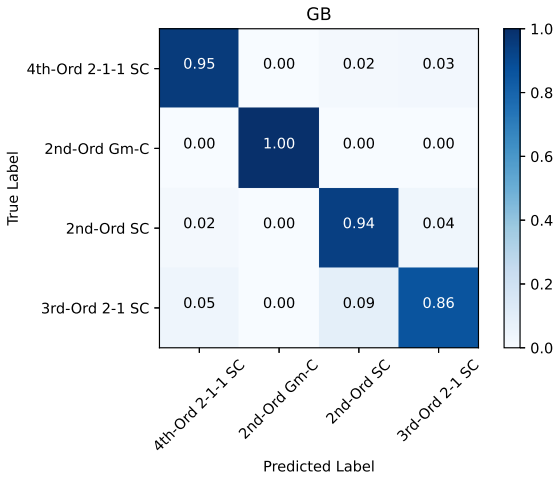


Fig. 5. Confusion Matrix for the Gradient Boosting (GB) Classifier.

A. Training Results

The training involved executing two independent processes (see Fig. 2(a)).

First we trained the classifiers. There, in order to avoid class-imbalance related problems, we down-sampled the dataset to obtain an equal representation of all classes (ΣΔM architectures). We set a 80%-20% division for Training Set (TS) and Validation Set (VS), obtaining the results in Table II for the four best performing options. The Gradient Boosting classifier, whose confusion matrix is depicted in Fig.5, provided the best accuracy over the Validation Set (i.e. over unseen previous data) and we selected it as the best option for our case study.

Second, each modulator's neural network topology and hyperparameters were optimized using NAS, and trained afterwards. The dataset was also split in 80%-20% subsets for training and validation.

[7]All simulations have been carried out in a PC with a 5.0 GHz, i9-12900F CPU, 64-GB RAM and NVIDIA® GeForce RTX™3060TI GPU.

TABLE I
DESIGN PARAMETERS AND RANGES USED TO GENERATE ΣΔM DATASETS

ΣΔM Arch.	Parameter(s)	Description	Range (Units)	Scale	Points	Scheme	Valid P.
2nd-Ord SC	a_o	Finite DC Gain	$(10, 1000)$	Log.	5×10^4	Random	23,432
	g_m	Amp. Transconductance	$(10\mu, 1\text{m})$, (A/V)	Log.	5×10^4		
	i_o	Max. Amplifier Output Current	$(100\mu, 10\text{m})$, (A)	Log.	5×10^4		
	V_n	Input-Referred Noise	$(10\text{p}, 100\text{n})$, $(\text{V}/\sqrt{\text{Hz}})$	Log.	5×10^4		
2nd-Ord Gm-C	$a_{o\{1,2\}}$	$\text{OTA}_{1,2}$ Finite DC Gain	$(10, 1000)$	Log.	40, 40	Nested	16,334
	$GBW_{\{1,2\}}$	$\text{OTA}_{1,2}$ GBW	$(1, 1000)$, MHz	Log.	25, 25	Loops	
	$IIP_{3,\{1,2\}}$	$\text{Integrators}_{1,2}$ 3rd-order intermod. prod.	(-10, 50), (dBm)	Lin.	10		
	IIP_{FF}	FeedForward path integrators IIP_3.	(-10, 50), (dBm)	Lin.	10		
3rd-Ord 2-1 SC	$a_{o\{1,2\}}$	$\text{Stage}_{1,2}$ Amplifiers Finite DC Gain	$(100, 10000)$	Log.	50, 50	Nested	58,570
	$g_{m\{1,2\}}$	$\text{Stage}_{1,2}$ Amplifiers Transconductance	$(10\mu, 10\text{m})$, (A/V)	Log.	25, 25	Loops	
	$i_{o\{1,2\}}$	$\text{Stage}_{1,2}$ Max. Amplifier Output Current	$(100\mu, 10\text{m})$, (A)	Log.	10, 10		
4th-Ord 2-1-1 SC	a_{o1}	First Stage Amplifiers Finite DC Gain	$(1, 1000)$	Log.	5×10^4	Random	19,769
	g_{m1}	First Stage Amplifier Transconductance	$(10\mu, 1\text{m})$, (A/V)	Log.	5×10^4		
	i_{o1}	First Stage Max. Amplifier Output Current	$(100\mu, 10\text{m})$, (A)	Log.	5×10^4		
	$k_{2,3,4}$	Scaling factors for 2^{nd}, 3^{rd}, and 4^{th} Stage Amps $\{a_o, g_m, i_o\}$ with respect to Amp_1	$(0.2, 1)$	Lin.	5×10^4		

We applied the grid-search NAS algorithm for each modulator and fine-tuned the resulting network template for either 2000 epochs or until an early-stopping condition was met[8].Table III summarizes the training results. Notably, the MSE values for the training and validation sets were very similar in all cases, demonstrating that none of the networks experienced overfitting. The NAS CPU time varied between 1 hour and 53 minutes and 6 hours and 4 minutes due to the differences in number of output variables, optimal number of layers and dataset dimensions. Once the optimal network architecture was determined by the NAS algorithm, the fine-tuning of each network took between 60 and 397 seconds to complete —almost negligible when compared to the NAS CPU time. Finally, the validation took less than 0.125 seconds in the worst case (2nd- ord. SC). Note that, during this time, this network generated 4687 designs, i.e. it only required 26μs to produce the high-level design variables for every set of specifications

B. Validating Results

To fully validate our methodology, we randomly selected 1000 $\{\bar{\Gamma}_k, \bar{\epsilon}_k\}$ pairs from the validation set (250 per architecture) and computed the relative deviation vector[9] $\bar{\rho}_k$ between the obtained specification vector, $\bar{\Gamma}_k^{'}$, and its requested value $\bar{\Gamma}_k$.

$$\bar{\rho}_k = \frac{\bar{\Gamma}_k^{'} - \bar{\Gamma}_k}{\bar{\Gamma}_k} \tag{5}$$

[8]The early-stopping condition was defined as no significant improvement in the loss function over the last 50 epochs.

[9]Here, vector division is defined as component-wise division.

TABLE II
CLASSIFIER TRAINING RESULTS

Classifier	Acc_{TS}(%)	Acc_{VS}(%)	CPU Time(s)
Grad. Boost.	94.4	93.7	6.47
Rand. Forest	99.9	92.4	0.62
Dec. Tree	100	91.4	0.11
SVM RBF	82.8	82.9	16.09

To obtain the resulting specifications $\bar{\Gamma}_k^{'}$, we employed the method described in Section III-F, equation (3), with $n_{iters} = 10$ and $\alpha = 5\%$.

Table IV shows the resulting FOM for the four architectures in our case study. Observe that the mean value of the deviation, i.e., the difference between the obtained FOM and the requested one, is very close to zero but positive in three of the cases and just slightly negative in the case of the cascade 2-1-1 SC. This result shows that the networks produced centered designs, in other words, designs meeting specifications, demonstrating that they learned to implement the specifications$\longrightarrow$high-level design variables mapping. Besides, the probability of producing designs with better than required specifications were of at least 68.5% in the worst case (2nd-ord Gm-C) and peaked to 78.5% for the cascade 2-1 SC. Finally, notice how the use of the small random search method, significantly improves the resulting FOM.

V. DISCUSSION

We have compared our solution against alternative more "conventional" optimizers incorporated in the SIMSIDES environment [41] available in MATLAB®, namely Genetic, Gradient Descent, and Positive Basis Np1. Before entering into the comparison results it is crucial to highlight the fact that our solution does not have any optimization capability. Every network has just learned how to map specifications onto design variables that guarantee (up to a level of accuracy) that these specifications are met, whether these specifications are optimal in any way or not, is completely beyond the network. This is an important concept; the optimization that occurs during the neural network training refers to the minimization of the error in generating this mapping not to how optimal the specifications are because specifications are inputs for the network.

In order to obtain the specifications to pass to the networks, we fixed OSR$=$ 128, and commanded the optimizers to maximize SNR for each of the modulator architectures in our case study. Power consumption was estimated by the

TABLE III
RNN TRAINING RESULTS

Case Study	N_{Layers}	$N_{Units}^{(a)}$	Dropout	Optimizer [38]	# Train	# Validation	MSE_{TS}	MSE_{VS}	$T_{NAS}^{(b)}$	$T_{Train}^{(c)}$	$T_{Val}^{(d)}$
2^{nd}-ord. SC	5	32	No	adam	18,746	4,687	0.034	0.033	2h.32min	63.81s	0.123s
2^{nd}-ord. Gm-C	5	32	No	adam	13,068	3,267	0.075	0.073	1h53min	105s	0.094s
3^{rd}-ord. 2-1 SC	6	32	No	RMSProp	47,862	11,965	0.031	0.028	6h4min	397s	0.025s
4^{th}-ord. 2-1-1 SC	1	32	Yes	RMSProp	15,816	3,954	0.016	0.017	2h40min	60.07s	0.088s

[a] Number of neurons per layer. [b] NAS GPU Time(s). [c] RNN GPU Training Time(s). [d] RNN GPU Validation Time(s).

TABLE IV
VALIDATION RESULTS

Result	$^{(a)}\rho_{FOM_{0,0}}$	$^{(b)}\rho_{FOM_{10,5\%}}$	Boost(%)
Architecture: 2nd-ord SC			
Mean Value	0.0059	0.0184	211.9
$^{(c)}$P($\rho > 0$)	0.6331	0.7470	18.0
$^{(d)}$Percentile(5)	-0.0976	-0.0826	15.4
Percentile(50)	0.0100	0.0196	96.0
Percentile(95)	0.0814	0.0960	17.9
Architecture: 2nd-ord Gm-C			
Mean Value	0.0215	0.0267	20.9
P($\rho > 0$)	0.6580	0.6850	4.1
Percentile(5)	-0.0572	-0.0550	3.8
Percentile(50)	0.0219	0.0269	28.1
Percentile(95)	0.1201	0.1253	4.3
Architecture: 2-1 Cascade SC			
Mean Value	0.0638	0.0741	16.1
P($\rho > 0$)	0.6979	0.7850	12.5
Percentile(5)	-0.1847	-0.1656	10.3
Percentile(50)	0.0196	0.0272	38.8
Percentile(95)	0.3881	0.4018	3.5
Architecture: 2-1-1 Cascade SC			
Mean Value	-0.0773	-0.0622	19.5
P($\rho > 0$)	0.6536	0.6997	7.1
Percentile(5)	-0.5917	-0.5825	1.6
Percentile(50)	0.0088	0.0102	15.9
Percentile(95)	0.0558	0.0575	3.0

[a] $\rho_{FOM_{0,0}}$ is the relative deviation (see eq.(5)) for the FOM when using only the design variables inferred by the RNN.
[b] $\rho_{FOM_{10,5\%}}$ is the relative deviation (see eq.(5)) for the FOM when improving the RNN result using the small random variations method (10 samples, $\pm 5\%$ of random variation).
[c] P($\rho_S > 0$) is the probability of a given relative deviation ρ_S to be larger than 0.
[d] Percentile(x)=T: When sorted, $(1-x)\%$ of the points are above T.

TABLE V
COMPARISON WITH OTHER OPTIMIZATION ALGORITHMS

Algorithm	CPU Time (min)	SNR (dB)	*P* (mW)	FOM(dB)
Architecture: 2nd-ord SC				
This work	0.05	86.01	10.16	149.0
Gradient	9	85.8	10.80	148.5
Genetic	18	88.1	18.41	148.5
Np1	5	86.6	20.00	146.6
Architecture: 2nd-ord GmC				
This work	0.05	75.19	0.07	182.6
Gradient	3	88.6	0.8	185.5
Genetic	129	90.2	0.8	187.0
Np1	10	88.0	0.4	187.9
Architecture: 3rd-ord Cascade 2-1 SC				
This work	0.05	115.8	2.7	181.5
Gradient	11	115.7	8.0	176.7
Genetic	145	116.2	4.1	180.1
Np1	12	115.9	8.0	176.9
Architecture: 4th-ord Cascade 2-1-1 SC				
This work	0.05	143.44	0.6	241.5
Gradient	4	144.32	1.8	237.9
Genetic	90	144.32	13.6	229.0
Np1	10	143.9	1.5	238.55

behavioral simulator SIMSIDES from the biasing currents obtained for the amplifiers during the SNR optimization. Then, for each architecture, we selected the best performing solution provided by the optimizers in terms of the FOM, and passed its results (SNR, OSR, Power) to the corresponding neural network to infer the design variables.

Table V presents the results of this process. First, we can highlight a noteworthy aspect: our solution obtained a CPU time improvement of at least $\times 60$ across the various modulators when compared to the available optimizers in SIMSIDES. Clearly, this is due to the fact that the optimizers do not incorporate any *a priori* knowledge about the problem whereas the networks, during the training phase, did learn how to precisely map specifications into design variables. Moreover, our networks are not too complex, involving just a few hundreds of neurons and a relatively low number of layers, consequently inference time is reduced to just a few milliseconds. Indeed, most of the time required to obtain the design variables was spent in the final search among the randomly generated points which involved using the behavioral simulator 10 times –as detailed in Section IV-B. As a by-product, due to the fine-tuning phase, our solution produced, in all cases, designs with a smaller power consumption thus yielding to better FOMs in three out of the four case studies. Probably, the most important contribution provided by our approach is the very short time required to obtain valid designs from new sets of specifications. Once the training phase is over, our framework is able to choose a modulator architecture and obtain its design variables from specifications in less than 50 milliseconds. However, we cannot neglect the fact that it requires a significant training time. In our opinion, this must be considered an initial cost that gets amortized proportionally based on the number of times the network is used.

Another important concern arises from the abundance of specification-design variable pairs within the datasets: why not utilize this information directly, treating it like a *Look-Up-Table* (LUT) where valid designs can be readily found? In other words, when new specifications are required, why not identify the nearest corresponding point in the dataset and simply select its associated design variables as the solution? The answer to this question is far from being simple as it heavily depends on how accurate are the networks producing design vars from specs as compared to the resolution of the data in the LUT, i.e. how densely the space of variables was explored when producing the dataset. Let us employ the two-dimensional example in Fig.6 to illustrate this. There, assume

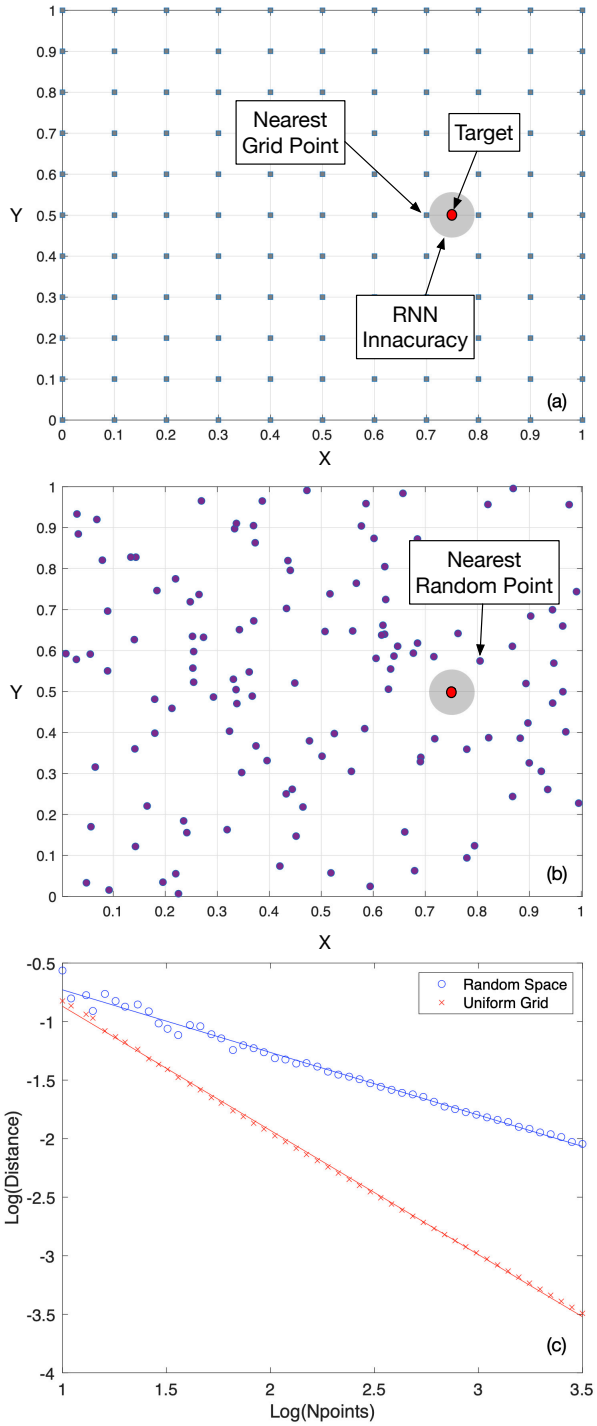


Fig. 6. Uniform vs. Random Coverage of the Design Space. (a) Uniform Coverage (b) Random Coverage (c) Average minimum distance between generated points vs. Number of generated points (log-log scales)

that the problem is to map specifications onto the 2D design variables space $[x, y] \in [0, 1]$. In order to create the dataset we have to obtain pairs of design-variables and specifications using a behavioral simulator and to do so, we would have two options; using a grid-based distribution of the design variables (which implies using nested loops in practice) Fig.6(a), or pick points randomly within this space Fig.6(b). In the first case, the resolution of the created LUT Δ_{LUT}^{Grid}, i.e. the average minimum distance between dataset entries, scales down with the number of points in the grid $\Delta_{LUT}^{Grid} \propto N_{points}^{-1}$ –Fig.6(c) solid line– whereas in the case of the randomly picked points, it scales down with its square-root[10], $\Delta_{LUT}^{Rand} \propto N_{points}^{-1/2}$ – Fig.6(c) dashed line. This has important implications when we have to compare the performance of the neural-network based method vs. just using the dataset as a LUT since it can be demonstrated that the average squared distance $\bar{d^2}$ between any point in a N-Dimensional space to the nearest point of a grid of resolution Δ_{LUT} is given by:

$$\bar{d^2} = \frac{N_{dims}}{12} \cdot \Delta_{LUT}^2 \tag{6}$$

To facilitate a comparison of the accuracies of the LUT and network approaches in our case study, we utilized the LUT as a predictor for design variables, just as we did when implementing our neural network based solution. We employed the training dataset as the exploration space and the entries from the validation set as the targets for which we sought predictions. Therefore, for every vector of specifications in the validation dataset, we selected the most similar one (nearest distance) in the training dataset and computed the L2-norm-squared[11] between their associated design variables. Table VI shows the obtained MSE for the LUT approach together, for comparison purposes, with the already presented values for our solution (Table IV). It can be clearly seen that the neural network approach outperformed the LUT method in all instances in our case study, with improvements varying between 66.7% for the Cascade 2-1 SC $\Sigma\Delta$M case and 34.6% for the Cascade 2-1-1 SC $\Sigma\Delta$M.

In summary, the extent to which the neural network solution surpasses a search among available designs (the LUT solution) in a comprehensive dataset hinges on the density of exploration within the specifications-design variables space during dataset generation, the complexity of the problem, and the number of design variables being sought. Generally, we can anticipate the networks to yield at least the same MSE as the LUT in the worst-case scenario. However, for highly dense datasets where data points are closely packed, the ability of the neural nets to generalize among very close neighboring points becomes less crucial, potentially resulting in only marginal performance gains compared to merely selecting the nearest point in the dataset. The true power of neural network solution comes into play when dealing with sparse or irregularly spaced datasets. In such cases, the network can learn patterns and dependencies across distant data points, and its ability to interpolate and generate new points can be much more valuable.

VI. Conclusion

The use of neural networks for the automated high-level design of $\Sigma\Delta$ modulators has been discussed. It has been demonstrated that trained classifiers can be used to identify a suitable $\Sigma\Delta$M topology for a given set of specifications,

[10]The problem is equivalent to the calculation of the 'mean free path' in solid-state physics or statistical theory.

[11]Since the average of this norm across the dataset is, by definition, the MSE.

TABLE VI
LUT VS. RNN COMPARISON

Architecture	Scheme	MSE_{LUT}	MSE_{RNN}
2nd-Ord. SC	Random	0.077	0.033
2nd-Ord. Gm-C	Nested Loops	0.173	0.073
3rd-Ord. 2-1 SC	Nested Loops	0.084	0.028
4th-Ord. 2-1-1 SC	Random	0.026	0.017

whereas regression-type neural networks can be trained to obtain the design variables that produce these specifications. This methodology can be applied to any arbitrary $\Sigma\Delta$M architecture, considering, for instance, either single-bit or multi-bit quantization, single-loop or cascade-loop filter implementation, and continuous time or discrete-time operation. The results have been compared with other optimization engines available in SIMSIDES, a widely used behavioral simulator, featuring a competitive performance in terms of the obtained figure of merits, whereas producing significant CPU time improvements. Finally we have also discussed how our approach produces better solutions than the simple use of the designs database as a Look-Up-Table.

REFERENCES

[1] S. Pavan, R. Schreier, and G. C. Temes, *Understanding Delta-Sigma Data Converters.* Wiley-IEEE Press, 2nd ed., 2017.

[2] J. M. de la Rosa, *Sigma-Delta Converters: Practical Design Guide.* Wiley-IEEE Press, 2nd ed., 2018.

[3] K. Francken *et al.*, "A high-level simulation and synthesis environment for delta-sigma modulators," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, pp. 1049–1061, August 2003.

[4] J. Ruiz-Amaya *et al.*, "High-Level Synthesis of Switched-Capacitor, Switched-Current and Continuous-Time $\Sigma\Delta$ Modulators Using SIMULINK-based Time-Domain Behavioral Models," *IEEE Trans. on Circuits and Systems – I: Regular Papers*, pp. 1795–1810, Sep. 2005.

[5] R. Schreier and G. C. Temes, *Understanding Delta-Sigma Data Converters.* IEEE Press, 2005.

[6] S. Pavan, "Systematic Design Centering of Continuous Time Oversampling Converters," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, pp. 158–162, March 2010.

[7] T. Bruckner *et al.*, "A GPU-Accelerated Web-based Synthesis Tool for CT Sigma-Delta Modulators," *IEEE Transactions on Circuits and Systems - I: Regular Papers*, vol. 61, pp. 1429–1441, May 2014.

[8] J. M. de la Rosa, "Design Automation of $\Sigma\Delta$ Converters: A Review of Modeling, Synthesis and Optimization Techniques," *Proc. of the IEEE Intl. Conf. on Electron Devices and Solid-State Circuits (EDSSC)*, October 2017.

[9] J. Wagner, M. Ortmanss, and J. M. de la Rosa, "Man or Machine – Design Automation of Delta-Sigma Modulators," *Proc. of the IEEE Intl. Symp. on Circuits and Systems (ISCAS)*, pp. 4229–4232, May 2018.

[10] M. Velasco, R. Castro-Lopez, and J. M. de la Rosa, "High-Level Optimization of $\Sigma\Delta$ Modulators Using Multi-Objetive Evolutionary Algorithms," *Proc. of the IEEE Intl. Symp. on Circuits and Systems (ISCAS)*, pp. 1494–1497, May 2016.

[11] N. Lourenco *et al.*, "On the Exploration of Promising Analog IC Designs via Artificial Neural Networks," *Proc. of the 2018 Intl. Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2018.

[12] Y. Li *et al.*, "An Artificial Neural Network Assisted Optimization System for Analog Design Space Exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 2640–2643, October 2020.

[13] E. Afacan *et al.*, "Review: Machine learning techniques in analog/RF integrated circuit design, synthesis, layout and test," *Elsevier Integration, the VLSI Journal*, vol. 77, pp. 113–130, November 2021.

[14] M. Fayazi *et al.*, "Applications of Artificial Intelligence on the Modeling and Optimization for Analog and Mixed-Signal Circuits: A Review," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, pp. 2418–2431, June 2021.

[15] A. Budak *et al.*, "An Efficient Analog Circuit Sizing Method Based on Machine Learning Assisted Global Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. IEEE-xplore Early Access Article, 2021.

[16] P. Jaraut *et al.*, "Augmented Convolutional Neural Network for Behavioral Modeling and Digital Predistortion of Concurrent Multiband Power Amplifiers," *IEEE Trans. on Microwave Theory and Techniques*, vol. 69, pp. 4142–4156, September 2021.

[17] P. Vaz, A. Gusmão, N. Horta, N. Lourenço, and R. Martins, "Speeding-up complex rf ic sizing optimizations with a process, voltage and temperature corner performance estimator based on anns," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1570–1574, 2022.

[18] J. Domingues, A. Gusmão, N. Horta, N. Lourenço, and R. Martins, "Accelerating voltage-controlled oscillator sizing optimizations with ann-based convergence classifiers and frequency guess predictors," in *2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pp. 1–4, 2022.

[19] K.-E. Yang, C.-Y. Tsai, H.-H. Shen, C.-F. Chiang, F.-M. Tsai, C.-A. Wang, Y. Ting, C.-S. Yeh, and C.-T. Lai, "Trust-region method with deep reinforcement learning in analog design space exploration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1225–1230, 2021.

[20] S. Bansal *et al.*, "Neural-Network Based Self-Initializing Algorithm for Multi-Parameter Optimization of High-Speed ADCs," *IEEE Transactions on Circuits and Systems - II: Express Briefs*, vol. 68, pp. 1384–1388, January 2021.

[21] J. M. de la Rosa, "AI-Assisted Sigma-Delta Converters – Application to Cognitive Radio," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, pp. 2557–2563, June 2022.

[22] G. Wolfe and R. Vemuri, "Extraction and use of neural network models in automated synthesis of operational amplifiers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, pp. 198–212, February 2003.

[23] P. Díaz-Lobo and J. M. de la Rosa, "High-Level Design of Sigma-Delta Modulators using Artificial Neural Networks," *Proc. of the IEEE Intl. Symp. on Circuits and Systems (ISCAS)*, May 2023.

[24] G. Gielen and J. Franca, "CAD Tools for Data Converter Design: An Overview," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, pp. 77–89, February 1996.

[25] R. Schreier, *The Delta-Sigma Toolbox*. [Online]. Available: http://www.mathworks.com/matlabcentral, 2017.

[26] M. Ortmanns *et al.*, *Uni Ulm Sigma-Delta Synthesis Tool*. [Online]. Available: http://www.sigma-delta.de.

[27] C. Renggli, L. Rimanic, N. M. Gurel, B. Karlas, W. Wu, and C. Zhang, "A data quality-driven view of mlops," *IEEE Data Engineering Bulletin*, vol. 44, pp. 11–23, March 2021.

[28] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[29] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[30] B. Ghojogh and M. Crowley, "Linear and quadratic discriminant analysis: Tutorial." https://arxiv.org/abs/1906.02590, 2019.

[31] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[32] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[33] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[34] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[35] J. M. Zurada, *Introduction to Artificial Neural Systems*. West Publishing Company, 1992.

[36] K. P. Murphy, *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

[37] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," Google Research, 2017.

[38] Keras, *Keras API Reference*. [Online]. Available: https://keras.io/api/., 2021.

[39] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, *et al.*, "Kerastuner." https://github.com/keras-team/keras-tuner, 2019.

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfit-

ting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[41] B. Cortes-Delgadillo *et al.*, "Embedding MATLAB Optimisers in SIMSIDES for the High-Level Design of $\Sigma\Delta$ Modulators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, pp. 547–551, May 2018.

Pablo Díaz-Lobo received his B.Sc. degree in Physics and Mathematics at the University of Seville, Spain, in 2022. In January 2023, he joined the Institute of Microelectronics of Seville (IMSE) as a junior research scholar, with a scholarship funded by the Spanish National Research Council (CSIC).

His main research interests include artificial neural networks and their application to the automated design of analog and mixed-signal integrated circuits.

Gustavo Liñán-Cembrano is a Tenured Scientist of the Spanish National Research Council at the Instituto de Microelectrónica de Sevilla (IMSE-CNM CSIC-Univ. Sevilla). He received hist degree in Physics in 1996, and PhD degree in Microelectronics. His research career focused at the beginning in the design of high complexity mixed-signal chips for vision, embedding sensing and processing at the pixel level. For about 10 years he was involved in designing CMOS imagers exhibiting very high dynamic range for industrial applications, including automotive. In recent years he has focused his research in vision processing application using artificial intelligence, among other techniques, for ecology applications, where he has published in the top journals, including Science and a software package which is freely distributed with more than 2000 users all over the world. He has authored more than 100 journal and conference papers. He has received the Best Paper Award of the International Journal of Circuit Theory and Applications two times and has been the co-chair of the SPIE's Microtechnologies for the New Millenium Conference also two times.

José M. de la Rosa (Fellow, IEEE) received the M.S. degree in Physics in 1993 and the Ph.D. degree in Microelectronics in 2000, both from the University of Seville, Spain. Since 1993 he has been working at the Institute of Microelectronics of Seville (IMSE), which is in turn part of the Spanish Microelectronics Center (CNM) of the Spanish National Research Council (CSIC). He is also a Full Professor at the Dpt. of Electronics and Electromagnetism of the University of Seville. His main research interests are in the field of analog and mixed-signal integrated circuits, especially high-performance data converters. In these topics, Dr. de la Rosa has participated in a number of Spanish and European research and industrial projects, and has co-authored over 260 international publications, including journal and conference papers, book chapters and the books Systematic Design of CMOS Switched-Current Bandpass Sigma-Delta Modulators for Digital Communication Chips (Kluwer, 2002), CMOS Cascade Sigma-Delta Modulators for Sensors and Telecom: Error Analysis and Practical Design (Springer, 2006), Nanometer CMOS Sigma-Delta Modulators for Software Defined Radio (Springer, 2011) and CMOS Sigma-Delta Converters: Practical Design Guide (Wiley-IEEE Press, 2013, 2nd Edition, 2018). He is in the World's Top 2% Scientists List from Stanford University (editions 2019, 2020 and 2022).

Dr. de la Rosa is an IEEE Fellow and member of the IEEE Circuits and Systems Society (CASS) and the IEEE Solid-State Circuits Society (SSCS). He is a Member-at-Large of the IEEE-CASS Board of Governors (BoG) for the 2023-2025 term. He served as a Distinguished Lecturer of IEEE-CASS (term 2017-2018), and as Chair of the Spain Chapter of IEEE-CASS during the term 2016-2017. He was at the front of the Editorial Board of IEEE Transactions on Circuits and Systems II: Express Briefs, where he served as Deputy Editor-in-Chief since 2016 to 2019, and as Editor-in-Chief in the term 2020-2021. He is a member of the TechRxiv Editorial Advisory Board since 2022. He also served as Associate Editor for IEEE Transactions on Circuits and Systems I: Regular Papers, where he received the 2012-2013 Best Associate Editor Award and was Guest Editor for the Special Issue on the Custom Integrated Circuits Conference (CICC) in 2013 and 2014. He served as Guest Editor of the Special Issue of the IEEE J. on Emerging and Selected Topics in Circuits and Systems on Next-Generation Delta-Sigma Converters. He is a member of the Analog Signal Processing Technical Committee of IEEE-CASS and of the Steering Committee of IEEE MWSCAS. He has also been involved in the organizing and technical committees of diverse international conferences, among others IEEE ISCAS, IEEE MWSCAS, IEEE ICECS, IEEE LASCAS, IFIP/IEEE VLSI-SoC, DATE and ESSCIRC. He served as TPC chair of IEEE MWSCAS 2012, IEEE ICECS 2012, IEEE LASCAS 2015 and IEEE ISICAS (2018, 2019). He has been a member of the Executive Committee of the IEEE Spain Section (terms 2014-2015 and 2016-2017), where he served as Membership Development Officer during the term 2016-2017. He has been recently appointed as Editor-in-Chief of IEEE TCAS-I for the term 2024-2025.