Fast and Efficient Constraint Evaluation of Analog Layout Using Machine Learning Models

Tonmoy Dhar[1], Jitesh Poojary[1], Yaguang Li[2], Kishor Kunal[1], Meghna Madhusudan[1], Arvind K. Sharma[1], Susmita Dey Manasi[1], Jiang Hu[2], Ramesh Harjani[1], and Sachin S. Sapatnekar[1]
[1] University of Minnesota, Minneapolis, MN, USA [2] Texas A&M University, College Station, TX, USA

ABSTRACT

Placement algorithms for analog circuits explore numerous layout configurations in their iterative search. To steer these engines towards layouts that meet the electrical constraints on the design, this work develops a fast feasibility predictor to guide the layout engine. The flow first discerns rough bounds on layout parasitics and prunes the feature space. Next, a Latin hypercube sampling technique is used to sample the reduced search space, and the labeled samples are classified by a linear support vector machine (SVM). If necessary, a denser sample set is used for the SVM, or if the constraints are found to be nonlinear, a multilayer perceptron (MLP) is employed. The resulting machine learning model demonstrated to rapidly evaluate candidate placements in a placer, and is used to build layouts for several analog blocks.

CCS CONCEPTS

• **Hardware → Electronic design automation**; **Physical design (EDA)**; **Analog and mixed-signal circuit optimization**; • **Computing methodologies → Machine learning**.

KEYWORDS

Analog layout, machine learning, performance analysis

ACM Reference Format:
Tonmoy Dhar, Jitesh Poojary, Yaguang Li, Kishor Kunal, Meghna Madhusudan, Arvind K. Sharma, Susmita Dey Manasi, Jiang Hu, Ramesh Harjani, and Sachin S. Sapatnekar. 2021. Fast and Efficient Constraint Evaluation of Analog Layout, Using Machine Learning Models. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431547

1 INTRODUCTION

The performance of analog circuits is very susceptible to layout parasitics, and a predictor that determines whether a particular layout configuration during place-and-route meets constraints or not has great utility. Several notable efforts in this direction have been made in the past. In [1], performance constraints are mapped to bounding constraints of parasitics using linear approximation of sensitivities of each interconnect to the performance. The work in [2] also leverages layout effect sensitivities which indirectly account for interconnect parasitics in layout placement. However, as these sensitivities of interconnect or layout effects are determined independently, such incorporation may not reflect the correlation among the interconnect parasitics. Moreover, the interconnect parasitic correlation may not be linear over the search space, as presumed in these works.

Another widely-used class of methods for tackling the problem of interconnect parasitics is to use a database of prior designs, collected from human designers, directly or to train ML models. The work in [3] searches through a design repository containing legacy designs and pulls out the best match for a target design for layout generation. The work [4] uses artificial neural networks trained on past designs to generate placements. In [5], a variational autoencoder based ML scheme is utilized to extract layout strategies from prior designs for use in routing. However, there are two problems with such methods. First, a database with well-crafted analog layouts by expert designers is hard to find. Second, layout styles change substantially with technology nodes and knowledge across nodes is often not transferrable, particularly for FinFET nodes with restricted design rules and high via/wire resistances. Recent work [6] leverages netlists from an automatic layout generator to train a 3D-CNN model that predicts placement quality. This also requires a large set of post-layout netlists as a training set.

In this paper, we propose a framework where, given a circuit, its performance specifications, and testbenches that simulate the circuit to extract its performance, we extract the correlation among all the sensitive interconnect parasitics automatically using machine learning (ML), and use them to build compact ML-based models for each constraint. The ML models that are built by our approach can be easily trained and used for rapid inference in a place and route engine. During the layout process, given the RC parasitics for a candidate layout, our models can predict whether the layout will meet specifications or not. Thus, our ML-based constraint modeling approach can steer a layout engine away from the part of the design space where constraints may not be met. Our ML models are simple to evaluate, and impose negligible overhead on the layout engine.

The proposed framework can extract both linear and nonlinear correlations among all the sensitive parasitics, over a multidimensional search space of RC parasitics, and is not dependent on a design database. We use as simple a model as possible: a fast evaluation determines whether a linear model would work, and following that, the linear model is refined, or a nonlinear model is employed only if needed. Separate models are built for each performance constraint. This can be applied to any analog circuit.

We define the problem space and the framework in Section 2. The framework is illustrated from Section 3–6. In Section 7 we demonstrate the approach on several operational transcondutance amplifier (OTA) circuits and a voltage-controlled oscillator (VCO), and show layouts generated using our approach. Finally, we conclude in Section 8.

2 PROBLEM FORMULATION

Consider a netlist C to be placed and routed, with a set of performance specifications $P = \{p_k \mid k \in \mathbb{N}^+\}$. The netlist consists of:

- a set of nets $N = \{n_i \mid i \in \mathbb{N}^+\}$,
- a set of modules $M = \{m_j \mid j \in \mathbb{N}^+\}$.

During place-and-route iterations, the movement of modules relative to each other changes the RC parasitics of interconnects between modules. We build ML models of electrical constraints on wires, which can be used to incorporate performance constraints accurately and efficiently.

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431547

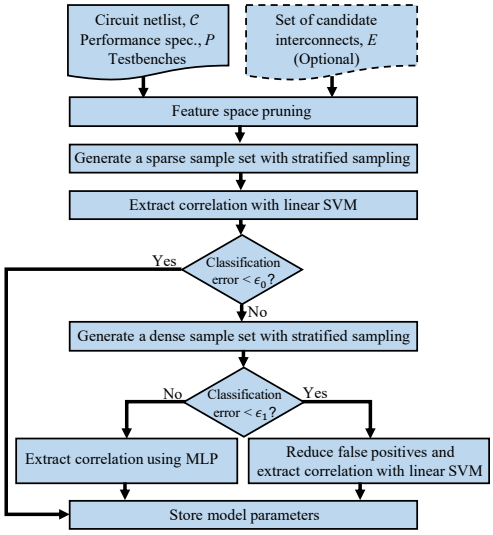

Figure 1: Our ML-based constraint evaluation framework.

Nets that are connected to multiple devices are modeled by a star configuration, where a multipin net, n_i, is represented by the star point, connected to each terminal of module m_j on the net by a two-pin interconnect, e_{ij}. Two-terminal nets do not need an additional star node. The star model is widely used and has been reported to be a good representation for parasitics in analog circuits [7], and this is consistent with our findings.

We annotate a set $E = \cup_{i,j} e_{ij}$ that either includes all interconnects, or a user-defined subset of all interconnects present in the circuit C. For each interconnect, e_{ij}, we annotate two variables, r_{ij} and c_{ij}, to represent its resistance and capacitance, respectively. Finally, we define $Z = \cup_{i,j \in E}\{r_{ij}, c_{ij}\}$ as the set of resistances and capacitances of all interconnects in E.

We use the most computationally simple model, the linear support vector machine (SVM) with sparse sampling. If this is inaccurate, we use dense sampling to obtain either a linear SVM with reduced false positives, or a nonlinear multilayer perceptron (MLP) if the data are not linearly separable.

An overview of the approach is outlined using the flowchart in Figure 1. The inputs to the method are the netlist C, with testbenches for evaluating its performance metrics, and circuit performance constraints P. The elements of set Z constitute the dimensions of a feature space, S. Our approach determines constraints that demarcate the *feasible region* of S, where performance constraints P are satisfied, and consists of the following steps:
(1) Feature space pruning: This step reduces the dimension of S by (a) identifying variables that the performance constraints are insensitive to, and (b) by range reduction, which determines upper bounds on the parasitics in the network. Practically, this step shrinks the dimension of S by reducing the dimension of Z, and by limiting the range of each variable so the feasible region covers much of S, setting the stage for sampling to be successful.
(2) Sparse sampling and labeling: Next, a sparse sample set in the updated feature space is generated using stratified sampling based on the Latin hypercube method. These samples are labeled for each performance constraint $p_k \in P$. A sample is labeled as *positive* (+1) if it satisfies the constraint p_k and *negative* (−1) otherwise.
(3) Classification using a linear SVM: For each $p_k \in P$, the approach employs a support vector machine (SVM) with a linear kernel to extract correlations among the features. To verify the accuracy of SVM, we check if the classification error falls below a user-specified threshold, ϵ_0: if so, the sample is linearly separable, and the constraint for p_k is modeled.
(4) Dense sampling and labeling: If the error exceeds ϵ_0, we generate a larger number of samples to drive higher accuracy.
(5) Model refinement: We use another user-specified threshold, $\epsilon_1 > \epsilon_0$, to determine whether to persist with a linear SVM or to use a nonlinear model. If the error of the linear SVM classifier is below ϵ_1, the linear model is considered redeemable by using dense sampling. In this case, we create a new SVM model and reduce the number of false positives. We focus on reducing false positives (rather than false negatives, though we report both): in a place-and-route engine, it is essential for the model to reject layouts that do not meet constraints. If the error of the linear SVM trained over the sparse sample set exceeds ϵ_1, an MLP is used on the dense sample set.

At the end of this process, each constraint $p_k \in P$ is described either using a linear SVM or nonlinear MLP model, and this can be used by a placer engine to rapidly evaluate layout feasibility.

3 FEATURE SPACE PRUNING

The efficiency and effectiveness of machine learning (ML) strongly depend on the number of features and the size of feature space [8]. In this problem, the interconnect parasitics, r_{ij} and c_{ij}, are the features for the ML based model. We initialize the feature space, S, by setting a worst case bound for each feature of Z. The worst case bound for the resistive and capacitive components in Z, r_{max} and c_{max} respectively, are based on:
Maximum net length bounds: A global pessimistic limit on a net length in the circuit is based on an estimate of the layout semiperimeter. This estimate is typically available for any analog design since standard circuit topologies have been built either at the current technology node, or in a previous node (from which the dimensions can be scaled). It is important to point out that our algorithm can work with an approximate and pessimistic estimate: e.g., when no estimate is available, the chip area may be used, which only means that the feature space pruning step may require more CPU time. However, a typical analog block (OTA, VCO, etc.) will be much smaller than the chip area, and in practice, it is easy to obtain a reasonable coarse area estimate.
Per unit wire resistances: If the maximum per-unit wire resistance, over all metal layers, is r^{pu}_{max}, the wire resistance is upper-bounded by $(r^{pu}_{max} \cdot l^{e}_{max})$.
Via resistances: The via resistance, r^{via}_i, to the next higher layer on metal layer M_i is particularly important in recent technologies, where via resistances are large, this discourages the use of too many metal layers. Since analog designs are very sensitive to interconnect resistances (e.g., because they can degrade the effective g_m of a transistor, or because the can cause undue voltage degradation due to IR drop which may change the operating region of a transistor), typically a small number ($\mathcal{V}$=2–4) of bends with vias may be used on any wire. Such a number is conservative because in practice, it is common to use parallel wires with parallel vias (in FinFET technologies with restricted design rules) or wider wires with larger/parallel vias (in bulk technologies) to reduce the resistance. However, an upper bound, r^{via}_{max}, can be calculated using these guidelines by adding up via resistances up and down the metal stack, and multiplying by $\mathcal{V}$. To our knowledge, the via resistance issue is not considered in prior works on constraint generation.

The sum of the maximum wire and via resistances yields a bound, r_{max}, for resistive components in Z. A worst-case bound for the capacitances in Z, c_{max}, is also similarly computed, ignoring negligible via capacitance.

$$\begin{aligned} r_{max} &= r^{via}_{max} + r^{pu}_{max} \times l^{e}_{max} \\ c_{max} &= c^{pu}_{max} \times l^{e}_{max} \end{aligned} \quad (1)$$

Hence, the initial feature space is a $|Z|$-dimensional hypercube, extending from 0 to r_{max} for resistive elements of Z, and 0 to c_{max} for capacitive elements. It is important to note that loose upper bounds are adequate at this stage. The feature space pruning step will then obtain tighter bounds.

This creates a large search space, S, and training a model over this entire space may be both challenging and unnecessary. For example, an analog circuit requires all transistors to be appropriately biased: if a large IR drop along a wire knocks a transistor away from its operating region, the circuit will not satisfy specifications, and it is possible to shrink S to eliminate this region, and save the effort of training an ML model over a clearly redundant subspace. We present two stages for pruning the feature space:

Feature elimination: We compact the search space by eliminating variables in Z as follows: (1) When two wires must be symmetric, as is common in analog layout, we use a single r and c variable for the wires. (2) The r and c variables associated with wires that are insensitive to the performance parameter p_k are removed from consideration. This sensitivity can be calculated through sensitivity analysis (e.g., using computationally cheap adjoint sensitivities) about the operating point.

Range reduction: The choice of upper bounds in (1) is purely based on layout considerations, but some parasitics may violate the performance constraints P at values well below the bound. We seek a tighter bound for these features by separately considering each feature in the reduced feature space, using a binary search to find thresholds, r_{ij}^t and c_{ij}^t, for resistive and capacitive components respectively, that maintain all performance specifications in P. To maintain pessimism for the bound, we assume zero parasitics for all other features, only during this computation.

At the range reduction step, we consider one variable at a time, without the consideration of interactions between variables. The goal here is merely to conservatively reduce the size of the space S. The next step of stratified sampling and ML is the step that builds ML models for electrical constraints, and it *explicitly considers interactions between variables.*

The outcome of the feature pruning step is a tighter set of bounds for each feature in a reduced feature space, given by

$$S = \{r_{ij}, c_{ij} \in Z \mid 0 \le r_{ij} \le r_{ij}^t, 0 \le c_{ij} \le c_{ij}^t\} \tag{2}$$

At the end of this step, the search space is reduced to a smaller hypercube. From this point, we denote elements of Z as z_ι irrespective of the feature type (resistance or capacitance), such that $1 \le \iota \le |Z|$.

As we describe the steps of the algorithm in detail in the next sections, we use a simplified version of the feature space in two dimensions for visualization. This is illustrated in Figure 2(a), with feature set $Z = \{z_1, z_2\}$, and the critical bounds indicated by the box $Z^t = \{z_1^t, z_2^t\}$. We will use this figure as a running example through the paper as we explain our method.

4 STRATIFIED SAMPLING

4.1 Latin Hypercube Sampling

Our sparse and dense sampling methods in Steps (2) and (4) in Section 2, are based on Latin hypercube sampling (LHS) [9], a robust stratified sampling technique that generates quasi-random sampling distributions. The advantage of LHS over random sampling is that LHS can representatively cover

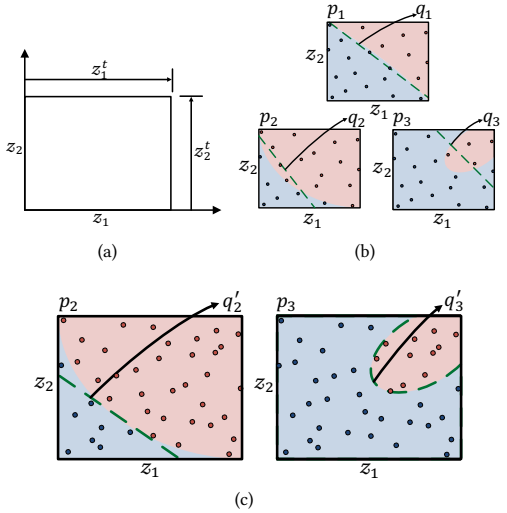

Figure 2: (a) Search space after feature space pruning. (b) Stratified sampling and classification with linear SVM + sparse sample set. (c) A denser sample set to reduce false positives for the SVM, or using MLP for nonlinear classification.

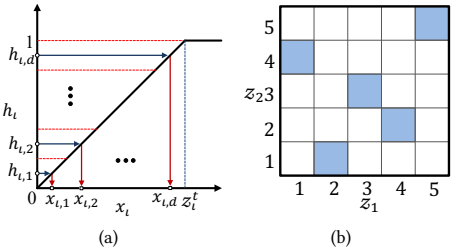

Figure 3: (a) Search space quantization, (b) LHS sampling.

a sampling space with a much smaller sample size than random sampling. Hence, LHS is used to increase the efficiency of Monte Carlo analysis.

The principle of LHS is based on uniformly sampling from an underlying probability distribution. If the probability distribution function of wire parasitics is available, it could be used for this purpose. Digital circuits follow a Rent's rule distribution for wire lengths, which may be used to generate distributions of RC parasitics, but for smaller analog circuits, a distribution of wire lengths has not been studied and is unknown. Moreover, even predicting wire lengths is insufficient: to estimate resistance distributions, via counts must be predicted as well. In this work, in the absence of an available distribution, we use a uniform distribution to model all variables. Our experimental results show that this assumption yields good results.

To generate a matrix with d samples using LHS from the $|Z|$-dimensional hyperspace, S, we work with uniform sampling along each dimension. We illustrate LHS for this distribution. The cumulative distribution function of variable x_ι, with a support of $[0, z_\iota^t]$, can be represented as follows:

$$F_\iota(x_\iota) = \begin{cases} 0 & \text{for } x_\iota < 0 \\ x_\iota / z_\iota^t & \text{for } 0 \le x_\iota \le z_\iota^t \\ 1 & \text{for } x_\iota > z_\iota^t \end{cases} \tag{3}$$

The LHS procedure has two primary steps:
(1) *Divide and sample*: The objective of this step is to generate d samples along each dimension z_ι of $|Z|$ so that they represent the entire distribution along that dimension. To do so, the CDF curve of each dimension is partitioned into d non-overlapping intervals with equal distribution (Figure 3(a)). Next, for each of the d intervals, a probability $h_{\iota,i}$ within the interval can be chosen randomly or deterministically (in our work, we deterministically choose the center point of the interval). These d values of probability are mapped to d samples along the dimension using the inverse transformation:

$$x_{\iota,i} = F_\iota^{-1}(h_{\iota,i}) \tag{4}$$

The d sample values along dimension z_ι can be assembled as a vector:

$$\mathbf{x}_\iota = [x_{\iota,1} \;\; x_{\iota,2} \;\; \dots \;\; x_{\iota,d}] \tag{5}$$

Sampling along each dimension results in $|Z|$ such vectors.
(2) *Permutation*: In this step, d items of each vector $\mathbf{x}_\iota$ are permuted randomly over the $d!$ possible permutations. The permutation along a dimension is independent from the permutation along all other dimensions. The permutation results in a mapping of vector $\mathbf{x}_\iota$ to $\mathbf{x}'_\iota$. Note that $\mathbf{x}_\iota$ was built to be in increasing order by design, and the random permutation to $\mathbf{x}'_\iota$ ensures that each element of the vector appears in randomized order. Next, all of the permuted vectors along each direction are merged into the matrix

$$X = \left[\mathbf{x}_1'^{\mathrm{T}} \; \mathbf{x}_2'^{\mathrm{T}} \; \dots \; \mathbf{x}_{|Z|}'^{\mathrm{T}}\right] \in \mathbf{R}^{d \times |Z|} \tag{6}$$

Each row of X is a sample, $\mathbf{x}_d$, in the $|Z|$-dimensional hypercube.

Figure 3(b) illustrates LHS in a space $Z = \{z_1, z_2\}$ with $d = 5$. Figure 3(b) shows that along each dimension, the selected samples exhibit the classical Latin hypercube pattern, with one sample in each row and each column.

4.2 Applying LHS to Sample the Feature Space

Using LHS, the flow generates a sample set, X, systematically sampled from the $|Z|$-dimensional feature space, as described above. This sample

set represents a set of RC values for the interconnects, and is used to train and test a linear SVM. For each performance specification $p_k \in P$, the circuit is simulated at each sample point using a commercial SPICE-like circuit simulator. Based on the results of simulation, the samples in X are partitioned into binary classes: *(a)* those that satisfy the performance requirement, p_k, which are labeled as positive (+1) and *(b)* those that fail the specification, which are labeled as negative (−1). This is graphically illustrated in Figure 2(b) in a simplified 2D feature space for $P = \{p_1, p_2, p_3\}$. The samples with positive labels, as well as the corresponding positive sample subspace, are colored blue; the negatively labeled samples and their sample subspace are shown in red. If $\mathbf{y}_k$ denotes a vector that records the binary class for each sample, for the performance specification p_k, this procedure yields a dataset $\{X, \mathbf{y}_k\}$.

As outlined in Step (2) in Section 2, an initial sparse sample set for a small value of d is first generated and used to train a linear SVM. If the accuracy of the linear SVM is unsatisfactory, Step (4) of the flow generates another sample set, $X' \in S$, which is a denser sample set compared to X. The sample set in Figure 2(c) represents X'. This denser set is used either to reduce false positives in the classification with linear SVM, or to train a multilayer perceptron (MLP) in case the samples are not linearly separable.

5 LINEAR CONSTRAINTS USING SVM

An SVM [10] for classification is a supervised learning model that constructs a separating hyperplane between positive and negative data in a feature space X *for each performance constraint.* The SVM hyperplane maximizes the distance among the nearest training samples of different classes.

As stated above, the dataset $\{X, \mathbf{y}_k\}$ represents the labeled sample values, where the elements $y_{k,i}$ of vector $\mathbf{y_k}$ are labeled as +1 or −1, depending on whether the sample satisfies constraint p_k or not. The SVM method generates a hyperplane in the $|Z|$-dimensional feature space characterized by weight vector $\mathbf{w_k}$ and bias b_k for each performance parameter p_k that builds the best linear separation between the positively and negatively labeled data. Together, the set $\{(\mathbf{w_k}, b_k) \forall k\}$ represents the model parameters of the linear SVM that defines a hyperplane for each $p_k \in P$.

For each performance specification p_k, SVM solves an optimization problem, represented below in primal form [10]:

$$\min_{\mathbf{w}_k, b_k} \frac{1}{2}\mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k + C\sum_{i=1}^{d}\xi_i \tag{7}$$

$$\text{subject to:} \quad y_{k,i} \times (\mathbf{w}_k^{\mathrm{T}}.\mathbf{x}_i + b_k) \geq (1 - \xi_i); \;\; \xi_i \geq 0; \;\; 1 \leq i \leq d$$

The objective function maximizes the separation between positive and negative labels, with C being a penalty term for negatively labeled samples that encroach on the positive side of the hyperplane, up to distance ξ_i; the extent of encroachment is defined by the constraints.

Our flow leverages a linear SVM to classify the sample space with dataset $\{X, \mathbf{y}_k\}$ corresponding to each performance specification p_k, through generating a $|Z|$-dimensional hyperplane that separates *positive* and *negative* samples. For $P = \{p_1, p_2, p_3\}$ and the hypothetical 2D feature space in Figure 2(b), q_1, q_2, and q_3 represent separating hyperplanes generated by the linear SVM for the sparse sample set.

The classification error of the linear SVM denotes whether the positive and negative samples are linearly separable. As described in Figure 1, if the classification error is below ϵ_0, the flow considers the sample space as linearly separable, and stores the model parameters of the linear SVM, which now define the ML model for constraint p_k. This is seen for the constraint p_1 using the parameters of q_1 for the SVM model (Figure 2(b)).

If the classification error is above ϵ_0 but below a tolerance ϵ_1, the flow still trains a linear SVM, prioritizing the reduction of misclassification of negative samples using the denser sample set X' and stores the respective model parameters. This event is illustrated with sample space of p_2 in Figure 2(b) and (c): q_2 is the set of model parameters generated with linear SVM for sparse sample set, X, and q_2' indicates the set of model parameters of linear SVM that trains over dense sample set, X', to reduce false positives.

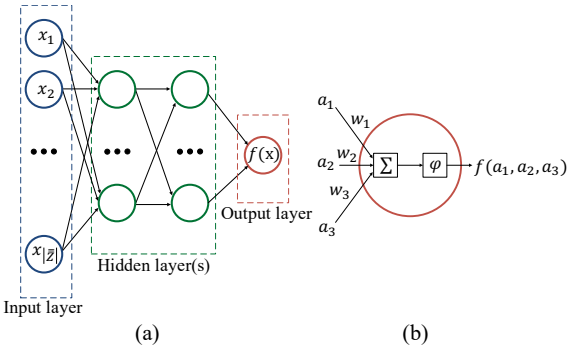

Figure 4: (a) MLP network. (b) A neuron of an MLP.

6 NONLINEAR CONSTRAINTS USING MLP

In our flow, if the classification error of the linear SVM with sparse sampling is high ($\geq \epsilon_1$), it is an indication that denser sampling cannot overcome the limitations of a linear SVM, i.e., the constraint is inherently nonlinear. In this case, our approach uses a multilayer perceptron to model the nonlinear constraint. In Figure 2, the feature space for p_3 represents such a case. In this illustration, the line of separation for positive and negative samples is visibly nonlinear for p_3. As a result, a separating hyperplane generated by linear SVM q_3 cannot separate the sample space with satisfactory accuracy (Figure 2(b)). Algorithmically, we detect this when the error of the linear SVM exceeds threshold ϵ_1. Figure 2(c) shows the MLP-generated line of separation after training with the dense set, X'.

A multilayer perceptron (MLP) is a supervised learning algorithm, trained using back-propagation, that can learn to model nonlinear functions, with the help of an underlying neural network, for data classification. An MLP consists of an input layer, an output layer, and at least one hidden layer in between the two (Figure 4(a)). Each layer consists a set of neurons (Figure 4(b)), where each neuron in the input layer represents an input feature.

An MLP is fully connected: each node of a layer is connected to all nodes of the following layer with a weight. Each neuron transforms values of the previous layer in two steps. First, a weighted linear summation is generated using all values of the previous layer. Next, the sum is transformed again using a nonlinear activation function. The operation in a neuron is illustrated in Figure 4(b) that includes three inputs a_1, a_2, a_3 with the weights of w_1, w_2, w_3 respectively. The symbol "$\sum$" indicates calculation of weighted sum of inputs and "φ" indicates the activation function. With these transformations, an input feature sample $\mathbf{x}_i$ propagates through hidden layers (Figure 4(a)). For binary classification, output layer consists of a single neuron and generates a value between 0 to 1 based on sample $\mathbf{x}_i$. If the output for a sample is ≥ 0.5, it is labeled as positive; otherwise it is negative.

In Figure 2(c), the set of model parameters for MLP that includes number of hidden layers and number of neurons per layer, is denoted with q_3'. We use a rectified linear unit (ReLU) for activation.

7 RESULTS

We have implemented the proposed framework within a Python/C++ environment. The core framework is implemented using Python and the scikit-learn Python library for machine learning tasks. We use a simulated annealing driven placer based on [11], programmed in C++, for placement and routing, and Calibre PEX for parasitic extraction from the layouts. All evaluations are carried out in a Linux server consisting of Intel Xeon(R) 2.20GHz Silver 4114 processors. All testcases of this section are designed using a commercial 12nm PDK and simulated with Cadence Spectre.

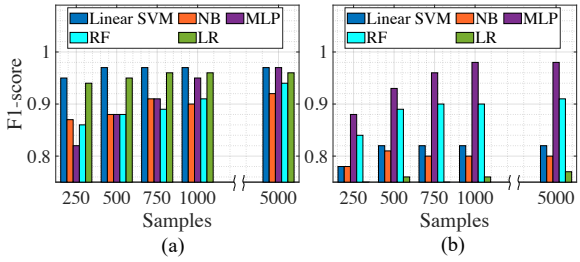

Figure 5: Comparison of the F1-score of binary classifiers for (1) bandwidth of 5T OTA, (2) gain of two-stage OTA.

To illustrate the classification quality of the framework, we use three metrics: precision, recall, and F1-score. For a set of test samples, these metrics can be defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}; \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}; \quad \text{F1-score} = \frac{2.\text{Precision}.\text{Recall}}{\text{Precision} + \text{Recall}}$$

Here, TP, FP, and FN stand for the number of true positives, false positives, and false negatives, respectively. In order to determine whether to use a linear SVM with sparse/dense samples, or an MLP, the framework compares $(1 - \text{F1-score})$ with the user-defined thresholds, $(\epsilon_0, \epsilon_1) = (0.05, 0.1)$.

7.1 Constraint Modeling with Our Framework

OTA testcases: We first evaluate the proposed framework with three different OTA topologies: (1) A five-transistor (5T) OTA, (2) a single-ended telescopic OTA, (3) a two-stage OTA. All OTAs contain transistors as well as C passives. The specifications for each design in a 12nm technology are listed in Table 1. The evaluation of these specifications involves multiple simulation setups: here, five such testbenches, labeled AC_1, AC_2, AC_3, DC, and TRAN, are used. For each testcase, samples are generated after feature space pruning (Sec. 3), with LHS sampling (Sec. 4.1). The framework randomly partitions 80% samples for training, and uses the rest for testing.

Recall that our philosophy is to go from the simplest to increasingly complex constraint models. We justify the structure of our framework by analyzing the sample spaces for two example performance constraints: (1) the bandwidth (BW) of 5T OTA, and (2) the gain of two-stage OTA. We compare the quality of results and runtimes using five binary classifiers: linear SVM, naive Bayes (NB), MLP, random forest (RF), and logistic regression (LR) [12].

From our observation, the boundary between *positive* and *negative* samples for BW of 5T OTA is highly linear. Figure 5(a) compares mean F1-score after k-fold cross validation (k=5), for the samples generated for 5T OTA and labeled according to whether they satisfy the BW specification or not. The sampling was carried out from the pruned feature space with LHS. For this set of samples, we see that as we vary the number of samples, the linear SVM consistently achieves the highest F1-score of all classifiers. The only other competitive classifier is the MLP, which achieves comparable scores at 5000 samples. In contrast, the linear SVM shows a high mean F1-score of 97% with only 500 samples, while MLP requires 5000 samples to equal this. Thus, if the constraint boundary is linear, the use of SVM with a linear kernel will result in a fast and efficient classification.

Next, we perform a similar comparison for the sample space of the gain of two-stage OTA. This boundary is found to be nonlinear for this feature space. Figure 5(b) compares the mean F1-score of the five binary classifiers, and we see that MLP outperforms the competition with a F1-score of 98% with 5000 samples and a score 96% at 1000 samples. This shows why we choose MLP as the nonlinear classifier.

Having justified the choice of classifiers, Table 2 now compares the training time and mean F1-score for three cases for the OTA testcases, summed up over all eight performance parameters, using (1) the proposed framework, (2) the linear SVM, trained on a sparse sample set, and (3) the MLP trained with a dense sample set. As expected, the linear SVM is the fastest for all three cases, but not as accurate as the MLP. Our framework shares the benefit of both models, with an F1-score similar to the MLP, but with much lower runtime as it uses the linear SVM when appropriate.

Table 3 summarizes the classification quality of the proposed framework for each $p_k \in P$, presenting the Precision, Recall, and F1-score. The classifier that is used for each circuit, for each performance metric, is also shown.

VCO testcase: We consider the testcase in Figure 7(a) of a voltage controlled oscillator (VCO) with 24 transistors (12 inverters used as analog elements) and 10 passives (resistors), again designed in a 12nm technology. The VCO operates with the control voltage range of $0 - 0.5$V in the frequency range of 2GHz–56GHz. We consider the input voltage range as its performance specification, and it is set to 0.2V–0.5V: a sample is positive if it can generate oscillation for the given input voltage range and negative otherwise. We achieve a classification F1-score of 91%, with a training time of 4.2s.

Importance of feature space pruning: Table 4 shows the contribution of feature space pruning to dimension reduction. It is evident that substantial reductions are achieved. In addition, within each dimension, range reduction is used to reduce the upper bounds on parasitics. The high F1-scores for

Table 1: Performance specifications P for three OTA circuits.

Performance specifications	Analysis type	Critical values		
		5T OTA	Telescopic OTA	Two-stage OTA
Gain (dB)	AC_1	≥ 19	≥ 42	≥ 25
BW (MHz)	AC_1	≥ 100	≥ 5	≥ 40
UGF (GHz)	AC_1	≥ 1	≥ 0.7	≥ 0.95
PM (^{o})	AC_1	≥ 60	≥ 60	≥ 60
CMRR (dB)	AC_2	≥ 48	≥ 64	≥ 30
PSRR (dB)	AC_3	≥ 19	≥ 42	≥ 25
SR (V/μS)	TRAN	≥ 150	≥ 400	≥ 300
ICMR (V)	DC	$0.60 - 0.75$	$0.55 - 0.85$	$0.60 - 0.75$

Table 2: Training time and F1-score comparison

	Criteria	Proposed framework	Linear SVM with sparse sample set	MLP with dense sample set
5T OTA	Training time	3.74s	0.18s	23.87s
	F1-score (mean)	0.97	0.92	0.96
Telescopic OTA	Training time	2.89s	0.27s	21.67s
	F1-score (mean)	0.97	0.94	0.98
Two-stage OTA	Training time	11.45s	0.25s	26.55s
	F1-score (mean)	0.97	0.87	0.98

Table 3: Classification quality for the OTA testcases. $LSVM_1$ ($LSVM_2$) = linear SVM with sparse (dense) sample set. $\mathcal{P}$ = Precision, $\mathcal{R}$ = Recall, F1 = F1-score. We set $C = 2$ in Eq. (7).

P	5T OTA				Telescopic OTA				Two-stage OTA			
	$\mathcal{P}$	$\mathcal{R}$	F1	Classifier	$\mathcal{P}$	$\mathcal{R}$	F1	Classifier	$\mathcal{P}$	$\mathcal{R}$	F1	Classifier
Gain	0.95	0.84	0.90	$LSVM_2$	0.95	0.91	0.93	$LSVM_2$	0.99	0.99	0.99	MLP
BW	0.98	0.97	0.98	$LSVM_1$	0.99	0.99	0.99	$LSVM_1$	0.98	0.97	0.98	MLP
UGF	0.99	0.96	0.98	$LSVM_1$	0.99	0.98	0.97	$LSVM_1$	0.95	0.95	0.95	$LSVM_1$
PM	0.99	0.99	0.99	$LSVM_1$	0.99	0.99	0.99	$LSVM_1$	0.99	0.98	0.99	$LSVM_1$
CMRR	0.92	0.95	0.94	MLP	0.91	0.89	0.90	MLP	0.99	0.98	0.98	$LSVM_1$
PSRR	0.99	0.99	0.99	$LSVM_1$	0.98	0.99	0.99	$LSVM_1$	0.95	0.91	0.93	$LSVM_2$
SR	0.99	0.99	0.99	$LSVM_1$	0.98	0.97	0.98	$LSVM_1$	0.99	0.98	0.98	$LSVM_1$
ICMR	0.99	0.99	0.99	$LSVM_1$	0.99	0.99	0.99	$LSVM_1$	0.89	0.93	0.91	MLP

Table 4: Dimension reduction using feature space pruning.

	5T OTA	Telescopic OTA	Two-stage OTA	VCO
$\lvert Z \rvert$ before pruning	23	33	28	95
$\lvert Z \rvert$ after pruning	15	19	18	50

Table 5: Post-layout performance of the OTA testcases.

Performance specifications P	5T OTA		Telescopic OTA		Two-stage OTA	
	With framework	Without framework	With framework	Without framework	With framework	Without framework
Gain (dB)	20.57 ✓	19.09 ✓	42.13 ✓	38.12 ✗	26.57 ✓	24.38 ✗
BW (MHz)	103.26 ✓	126.20 ✓	5.49 ✓	7.64 ✓	46.84 ✓	41.22 ✓
UGF (GHz)	1.17 ✓	1.14 ✓	0.70 ✓	0.61 ✗	1.00 ✓	0.92 ✗
PM (^{o})	110.33 ✓	116.77 ✓	133.41 ✓	106.50 ✓	94.43 ✓	82.05 ✓
CMRR (dB)	52.08 ✓	52.92 ✓	69.15 ✓	62.14 ✗	32.71 ✓	38.27 ✓
PSRR (dB)	21.39 ✓	18.47 ✗	42.45 ✓	53.52 ✓	26.94 ✓	24.37 ✗
SR (V/μS)	156.62 ✓	156.63 ✓	414.24 ✓	424.23 ✓	408.19 ✓	386.07 ✓
ICMR (V)	0.60-0.75 ✓	0.60-0.75 ✓	0.55-0.85 ✓	0.55-0.85 ✓	0.60-0.75 ✓	0.60-0.75 ✓

all performance specifications of all testcases are facilitated by the feature space pruning in two ways: (1) In the absence of the feature space pruning step, the SVM/MLP are forced to fit the constraint boundary over a larger domain/more dimensions, resulting in larger runtimes and generally lower quality, as a smaller fraction of samples lie near the constraint boundary. (2) By reducing variable ranges for constraint p_i when *any* constraint p_j is violated. Thus, limits from constraint p_j help to clip out nonlinearities in the constraint boundary for p_i, making linear SVM modeling feasible over a reduced domain. Thus, our approach incorporates mutual interactions between both variables and constraints.

7.2 Application in an Analog Placement Engine

We apply the constraints generated by our framework to guide the placement of the above OTA and VCO circuits within ALIGN [13]. The cost function of the simulated annealing based placer, based on [11], originally used a weighted sum of the normalized area, A, and normalized wirelength, W. We alter it by adding a penalty for violating a performance constraint:

$$\min \alpha \cdot A + \beta \cdot W + \gamma \sum_{k=1}^{|Q|} q_k(\mathbf{x}_i) \tag{8}$$

where α, β, and γ are weighting coefficients. Here, Q is the set of the sets of model parameters generated by the framework for all $p_k \in P$. For a given placement, $\mathbf{x}_i$ represents the vector of RC parasitics in the hyperspace S, and $q_k(\mathbf{x}_i)$ denotes the cost predicted by the framework for the sample $\mathbf{x}_i$ and performance p_k. For a linear SVM, $q_k(\mathbf{x}_i)$ evaluates $(\mathbf{w}_k^{\mathbf{T}} \cdot \mathbf{x}_i + b_k)$ and outputs 1 (−1) if the result is ≥ 0 (< 0). For MLP, the model that outputs a value between zero to one. Thresholding this value with a user-tunable parameter (e.g., 0.5 as default) assigns q_k to 1 if the predicted probability is greater than the threshold, and −1 otherwise. Hence, points that meet the specification are rewarded, and those that do not are penalized. The ML models are simple to evaluate and require only a few arithmetic operations, and cause negligible overhead to the placement engine.

To illustrate the application of our flow, we incorporated the proposed framework to a custom analog layout generator and synthesized the OTA designs (Figure 6). All layouts were verified to be LVS-correct. The performances of the layouts, extracted from post-layout analyses, are summarized in Table 5. All three of the automatically generated layouts maintain the required design constraints, indicated by the "✓" sign in the table. Contrarily, layouts generated without the framework fail to meet some of the requirements, indicated by the "✗" sign. For the VCO, Figure 7(b) shows the circuit layout, and it meets all specifications.

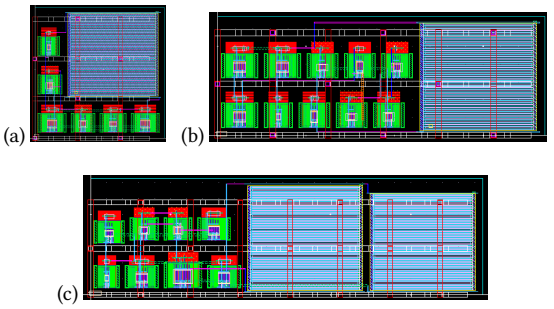

Figure 6: Automated layouts of the (a) 5T OTA (9.63μm × 9.60μm), (b) Telescopic OTA (6.85μm × 18.65μm), and (c) two-stage OTA (7.42μm × 24.49μm) with constraints generated by our framework. [Not drawn to scale]

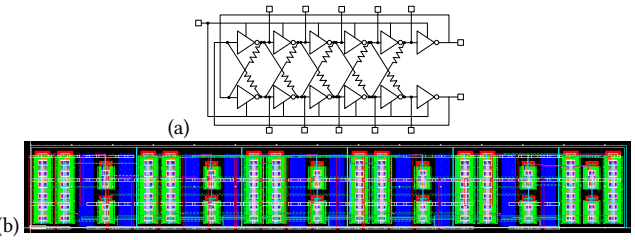

Figure 7: VCO (a) schematic, (b) layout (10.11μm × 72.78μm) using our method.

8 CONCLUSION

This paper automatically generates ML-based performance models from prelayout netlists, capturing correlations among wire parasitics. The feature space initially includes all interconnect parasitics as independent entities, and then this space is pruned. Linear SVM or MLP models are then trained based on efficient LHS-based sampling. When integrated into a layout engine, the models are effective in generating layouts that meet specifications.

REFERENCES

[1] U. Choudhury and A. Sangiovanni-Vincentelli, "Automatic Generation of Parasitic Constraints for Performance-Constrained Physical Design of Analog Circuits," *IEEE T. Comput. Aid D.*, vol. 12, no. 2, pp. 208–224, 1993.
[2] K. Lampaert, *et al.*, "A Performance-Driven Placement Tool for Analog Integrated Circuits," *IEEE J. Solid-St. Circ.*, vol. 30, no. 7, pp. 773–780, 1995.
[3] P.-H. Wu, *et al.*, "A Novel Analog Physical Synthesis Methodology Integrating Existent Design Expertise," *IEEE T. Comput. Aid D.*, vol. 34, no. 2, pp. 199–212, 2015.
[4] D. Guerra, *et al.*, "Artificial Neural Networks as an Alternative for Automatic Analog IC Placement," in *Proc. SMACD*, pp. 1–4, 2019.
[5] K. Zhu, *et al.*, "GeniusRoute: A New Analog Routing Paradigm Using Generative Neural Network Guidance," in *Proc. ICCAD*, pp. 1–8, 2019.
[6] M. Liu, *et al.*, "Towards Decrypting the Art of Analog Layout: Placement Quality Prediction via Transfer Learning," in *Proc. DATE*, pp. 496–501, 2020.
[7] B. Shook, *et al.*, "MLParest: Machine Learning based Parasitic Estimation for Custom Circuit Design," in *Proc. DAC*, pp. 1–6, 2020.
[8] R. Liu, *et al.*, "Search Space Preprocessing in Solving Complex Optimization Problems," in *Proc. BigData*, pp. 1–5, 2014.
[9] K.-T. Fang and R. Li and A. Sudjianto, *Design and Modeling for Computer Experiments.* CRC Press, Boca Raton, FL, 2005.
[10] B. Scholkopf and A. J. Smola, *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, 2001.
[11] Q. Ma and L. Xiao and Y.-C. Tam and E. F. Y. Young, "Simultaneous Handling of Symmetry, Common Centroid, and General Placement Constraints," *IEEE T. Comput. Aid D.*, vol. 30, pp. 85–95, Jan. 2011.
[12] E. Alpaydin, *Introduction to Machine Learning.* MIT press, Cambridge, MA, 2020.
[13] K. Kunal, *et al.*, "ALIGN: Open-Source Analog Layout Automation from the Ground Up," in *Proc. DAC*, pp. 1–4, 2019.