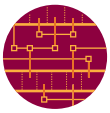
Self-Learning Tuning for Post-Silicon Validation

Milestone Report – GS-IMTR

Peter Domanski

October 5, 2022



GS-IMTR
Graduate School
Intelligent Methods for Test and Reliability

Advisor / main examiner: Prof. Dirk Pflüger

Co-examiner: Prof. Bin Yang

Mentor: Raphaël Latty

1 Introduction

Electronic devices occupy every aspect of our daily life. Therefore, it is crucial to verify the correctness and reliability of these devices, especially in safety-critical applications. Post-Silicon Validation (PSV) refers to validation efforts of manufactured circuits (prototypes) in the real application environment before sending the design to mass production [1, 2]. Ensuring functional correctness, checking security properties, and meeting performance constraints under operating conditions are important tasks in PSV. Moreover, robustness, e.g., against process variations in semiconductor manufacturing, is crucial for high yields in mass production [3]. Note that we use the terms circuits and devices interchangeably to describe the in-silicon manufactured prototype.

PSV is a crucial challenge for modern circuits with highly complex designs. Due to the increasing complexity, PSV is getting an expensive bottleneck because existing techniques cannot cope with the complexity of modern and future circuits [4, 5]. According to the study in [6], PSV can take up to 70% of the time, effort, and resources used during the design process. Moreover, only a few systematic techniques exist to solve the tasks in PSV. A particularly challenging task in PSV is robust performance tuning. It is necessary because designs rely on tuning to compensate impacts of process variations and non-ideal design implementations. Tuning implies setting variables and registers, so-called "tuning knobs" (e.g., bias settings or adjustable currents and voltages)[7]. The number of such knobs is rising because affordability is increasing in modern technologies. The aim is to set the tuning knobs such that circuit parameters stay within specification limits and optimize performance goals. Furthermore, the tuning knobs may be adjusted as a function of operating conditions, e.g., temperature or operating mode of the circuits.

Robust performance tuning remains a complicated, bound-constrained, and mixed-type optimization task. A first naive approach is to either use state-of-the-art gradient-based or derivative-free optimization algorithms. Derivative-free methods are relevant in black-box optimization (e.g., PSV) because we typically have no access to an analytical expression describing the behavior of manufactured circuits (similar to black-box functions) [8]. Therefore, we cannot compute higher-order moments of the underlying objective. However, applying state-of-the-art derivative-free optimization strategies to solve mixed-type optimization problems results in approximate, point-wise solutions. Given strict timing requirements in PSV, point-wise solutions remain inefficient because we need to re-run the optimization process for changing operating conditions. Due to the complex nature of the optimization task tuning each run of the optimization strategy can take a significant amount of time. Especially if we are interested in adjusting tuning knobs for many different operating conditions, the total runtime of state-of-the-art optimization strategies gets unreasonably large [7]. To avoid these pitfalls, we have proposed a novel self-learning tuning approach that aims to learn a mapping from (operating) conditions to tuning knob configurations. In this thesis, we call the mapping tuning law. We have proposed a combination of robust surrogate-based modeling of the objective function tailored to tuning [9] and Reinforcement Learning to train a flexible, robust, and efficient tuning law (optimization strategy) applicable in circuit-specific or circuit-independent tuning setups.

2 Background

2.1 Surrogate Modeling for Black-Box Optimization

In many real-world optimization tasks, objective functions are black-boxes that are very expensive to evaluate, e.g., complex computer simulation programs. In such settings, analytical or derivative information are not available or impractical to calculate.

A common approach to reduce computation times is to approximate the computationally expensive (black-box) function with a more efficient model, so-called surrogates. However, we do not know which surrogate model type will perform best for a specific application. Popular surrogate model types include radial basis functions, kriging, support vector machines, Bayesian approaches, and Neural Networks. Independent of the specific model type, the surrogate is usually used along with an optimization algorithm to search for a (global) optimal solution to the optimization task. However, algorithms to solve surrogate-based optimization problems are scarce and often designed for a specific application. Therefore, choosing a suitable optimization algorithm is difficult and depends on many factors, including data types, complexity

(e.g., number of function evaluations), and the formulation of the optimization task (e.g., constraint and unconstrained tasks). A common choice in black-box optimization are derivative-free algorithms because derivative information is not available, and the computational overhead is lower. For a k-dimensional problem, computing the gradient and the Hessian of a function $f(x)$ is, respectively, k-times and k^2-times as expensive as computing $f(x)$ [10].

The general formulation of a mixed-type optimization problem is given in Def. 2.1. For these problems types, which are especially relevant in real-world applications, only few approaches exist (e.g., [11, 12]). Additionally, existing work is mainly designed for a specific application increasing the difficulty of selecting a suitable surrogate model type and optimization algorithm for different applications.

Definition 2.1 (Mixed-type bound-constrained optimization problem) *Let the objective function $f : \mathbb{R}^{D_1} \times \mathbb{Z}^{D_2} \to \mathbb{R}$ denote the (expensive) black-box function with optimization variables $\vec{z}^T = \left(\vec{z_1}^T, \vec{z_2}^T\right)$, where $\vec{z_1} \in \mathbb{R}^{D_1}$ denote continuous variables and $\vec{z_2} \in \mathbb{Z}^{D_2}$ denote discrete variables.*

The bound-constrained optimization task (here: minimization task; maximization can be reformulated by minimizing the negative objective function) is defined as:

$$
\begin{aligned}
&\min_{\vec{z}} \; f\left(\vec{z}\right) \\
s.t. \quad & c_j\left(\vec{z}\right) \leq 0 \; \forall j = 1, \ldots, m \\
& -\infty < z^l_{1,k} \leq z_{1,k} \leq z^u_{1,k} < \infty \; \forall k = 1, \ldots, D_1 \\
& -\infty < z^l_{2,l} \leq z_{2,l} \leq z^u_{2,l} < \infty \; \forall l = 1, \ldots, D_2 \\
& \vec{z_1} \in \mathbb{R}^{D_1}, \; \vec{z_2} \in \mathbb{Z}^{D_2}, \; \vec{z}^T = \left(\vec{z_1}^T, \vec{z_2}^T\right), \; \textit{Dimensionality } D = D_1 + D_2
\end{aligned}
$$

2.2 Learning to Optimize

Learning to optimize (L2O) is a field of research that aims to leverage machine learning to develop new optimization algorithms or improve upon existing methods. Additionally, automatizing the laborious, iterative process of hand engineering optimization methods significantly reduces design efforts allowing the implementation of task-specific algorithms that can improve convergence speed (given a similar amount of computing budget) or final objective value. Instead of a theory-driven design for a specified class of optimization problems, L2O uses a data-driven approach to learn the algorithm by observing its execution (optimization performance) and shaping the optimizer according to the type of problem (e.g., task distribution). Therefore, we cannot specify (theoretical) performance guarantees for the learned optimization algorithms, which is the main reason for the current lack of a solid theoretical basis for L2O [13].

L2O has the potential to break the limits of analytical optimization methods. It is especially beneficial if we repeatedly solve certain types of optimization problems with similar data distributions. However, L2O typically falls short on out-of-distribution tasks. Applications of L2O include ℓ_1-minimization [14], neural network training [15], black-box optimization [16], and combinatorial optimization [17]. Moreover, many application areas could benefit from L2O methods, ranging from computer vision to computational biology. Nonetheless, existing works mainly focus on (unconstrained) continuous optimization algorithms. Training of L2O approaches usually implies a time-consuming offline stage. After training, the (online) application of the autonomous algorithm is typically more efficient concerning the total time consumption of the optimization process [18].

A popular approach to L2O is to use Reinforcement Learning (RL) to train an optimization algorithm that is represented by a RL policy, e.g., in [13]. The general structure of an optimization algorithm is given in Alg. 1. The optimizer is specified as a function π that depends on the function values or history of points in the domain of f. In the RL setting, we reward algorithms that converge quickly and penalize those that do not.

Related research areas to L2O are hyperparameter optimization, AutoML, algorithm configuration, learning to learn (meta-learning), and learning-augmented algorithms. Many approaches in these areas either try to determine optimal configurations for existing algorithms (search over different instantiations of an algorithm) or automatize steps in the ML life cycle. However, L2O is the only approach that allows

Algorithm 1 Optimization methods with RL, see [13]

Require: Analytical description or model of objective function f
  $x^{(0)} \leftarrow$ random point in the domain of$f$
  for $i = 1, 2, \ldots$ **do**
    $\Delta x \leftarrow \pi\left(f, x^{(0)}, \ldots, x^{(i-1)}\right)$
    if stopping condition is met **then**
      return $x^{(i-1)}$
    end if
    $x^{(i)} \leftarrow x^{(i-1)} + \Delta x$
  end for

searching the space of all possible algorithms directly. Thus, the autonomous algorithm may identify patterns in the task distribution data and discover more effective strategies than classical algorithms.

2.3 Deep Reinforcement Learning

Reinforcement Learning (RL) is an area of machine learning designed for sequential decision-making tasks. It is about a decision maker, the so-called agent, who interacts with an environment to learn an optimal behavior from information collected by trial and error [19]. The RL problem setup is fundamentally different from supervised and unsupervised learning because it is usually impractical to obtain examples of the desired behavior that are correct and representative of all possible situations [20]. Moreover, RL is goal-directed, meaning we aim to learn from interaction instead of trying to find hidden structures in the data. In the RL setting, the agent receives a state s_t in a state space $\mathcal{S}$ and selects an action a_t from an action space $\mathcal{A}$ following a policy $\pi\left(a_t|s_t\right)$ (e.g., mapping from state s_t to action a_t that describes agent's behavior) at each discrete time step t. Due to the selected action, the agent receives a feedback, e.g., scalar reward $r_t \in \mathbb{R}$ and the next state $s_{t+1} \in \mathcal{S}$ of the environment. Defining a reward signal to formalize the agent's goal is a key idea of RL and allows one to acquire new behaviors and skills incrementally [21, 22]. The agent's goal is to maximize the (discounted) cumulative reward, also called return R_t over time. Formally, the RL setting is defined as a Markov Decision Process (MDP) see Def. 2.2. In general, a process has the Markov property if the future of the process only depends on the current state of the environment. A so-called trajectory is a sequence of state, action, and reward pairs.

Definition 2.2 (Markov Decision Process) *A MDP is a discrete time stochastic control process defined as a 5-tuple* $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ *where:*

- $\mathcal{S}$ *is the state space*
- $\mathcal{A}$ *is the action space*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is the transition function (set of conditional transition probabilities between states, e.g.,* $P\left(s_{t+1}|s_t, a_t\right)$*)*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to R$ *is the reward function, where* R *is a continuous set of possible rewards* $R_{max} \subseteq \mathbb{R}$ *(e.g., reward distribution* $P\left(r_t|s_t, a_t\right)$*)*
- $\gamma \in [0, 1)$ *is the discount factor (e.g., scale down future rewards after each step such that total sum remains bounded)*

The formulation as a MDP provides an abstract and flexible framework because time steps are not fixed intervals, and actions can be high- or low-level controls of environmental parameters. Similarly, states can include low-level sensations or high-level descriptions of the environment's state. Additionally, the formulation allows continuous and discrete state and action spaces. The agent aims to maximize the expectation of the long-term return (cumulative discounted future reward) given in Eq. 1. Thus, a function that predicts the expected return given the state s_t for the current policy is often used to learn the optimal policy. In addition to this so-called V-value function (see Eq. 2), the Q-value function is often of interest because it allows to obtain the optimal policy π^* directly, see Eq. 3 and Eq. 4 respectively. The Q-value function describes the expected return for selecting action a_t in state s_t following the current

policy π.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi\right] \tag{2}$$

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi\right] \tag{3}$$

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q^*(s,a) \tag{4}$$
$$\text{with } Q^*(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a)$$

The success of modern RL approaches is mainly due to the combination of Deep Neural Networks (DNNs) and RL, so-called Deep RL (DRL). The usage of DNNs (e.g., Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs)) allows to automatically extract features and exploit the expressiveness and flexibility of DNNs, e.g., to address the curse of dimensionality and to reduce domain knowledge significantly. Most commonly, DNNs are used to approximate the V-value function, the Q-value function, and the policy π (optionally state transition function $\mathcal{T}$ and reward function $\mathcal{R}$) which allows to directly operate in high-dimensional state and action spaces. These abilities enable agents to achieve outstanding performance in many real-world decision-making problems.

Today, many (Deep) RL algorithms exist. In the following, we only focus on on-policy learning. It has stronger convergence properties compared to off-policy approaches but usually lower sample efficiency because on-policy algorithms discard experience (e.g., stored in a replay buffer) that is not collected with the current policy. A popular class of methods is policy optimization. Algorithms like REINFORCE [23] or Proximal Policy Optimization (PPO) [24] directly optimize the parameterized policy $\pi(a|s,\Theta)$ and update the parameters Θ by gradient descent on $\mathbb{E}[R_t]$. DRL approaches have proven successful in many areas, including games, robotics, finance, industrial automation, etc. Specific applications range from the real-time adjustment of operation parameters on a petroleum refinery, automating cooling of data centers to save energy, autonomous robots on assembly lines, financial trading, to self-driving cars. Recently, DRL approaches have also been used to automatize the design process of optimization algorithms (learning to optimize)[13].

Despite the success, Deep RL still has unique challenges and open problems: Firstly, the fundamental exploration vs. exploitation trade-off. The main question is how to exploit the current best action to maximize rewards vs. exploring the environment to identify better actions. Secondly, in contrast to other areas of ML it is unclear how to transfer learned knowledge to different tasks (e.g., using transfer learning to generalize to similar contexts). Additionally, sample efficiency, stability of training, multi-objective reward functions, and catastrophic inference can be challenging aspects for Deep RL approaches. Finally, the reward definition and the representation of actions and states can strongly affect the performance increasing the difficulty designing suitable (Deep) RL approaches [25, 26].

3 Problem Definition

The problem of robust performance tuning in PSV is very intricate, especially with the increasing complexity of modern circuits. In our application, we aim to optimize a user-defined (scalar) figure-of-merit (FOM) that encodes aspects of the circuits, e.g., margin to specification limits, absence of errors, and minimum power consumption. Starting point is a PSV characterization data set including 9 (prototype) circuits with 100k (uniformly distributed) data points per circuit. The circuits under test (CUTs) are representative of the process spread. A more detailed description of the data is given in Tab. 1.

Given the characterization data set, various quantities must be modeled. Firstly, we need to model the FOM for a specific circuit d. Secondly, we need a tuning model to compute optimal tuning configurations as a function of the tuning conditions. Additionally, in production, a circuit-independent FOM model

is of interest. The overall problem definition is given in 3.1. It typically includes multiple aggregations, followed by (mixed-type) surrogate-based optimization.

Definition 3.1 (Robust performance tuning) *Defining the overall problem, we are using the following notation:*

- ***Device variables*** d*, e.g. device IDs*
- ***Test case input variables*** $\vec{x}$*, e.g., operating conditions like voltages, temperature, device modes, and tuning knobs*
- ***Test case output variables*** $\vec{y}$*, e.g., measurements from external instruments, status information, and on-chip measurements values*
- ***Figure-of-merit*** f*, e.g., computed from measurement outputs which quantifies how well a specific test case performs*
- ***Aggregated FOM*** $F\left(d, \vec{c}, \vec{t}\right)$
- ***Achievable worst-case robust FOM*** F^*

The test case inputs are split according to Eq. 5 where $\vec{x}^-$ *are the remaining input variables.*

$$\vec{x} = \left(\vec{c}, \vec{t}, \vec{x}^-\right) \tag{5}$$

Combining the aggregation and optimization steps, we get the following formulation:

$$\begin{aligned} \vec{t}^* &= \vec{t}^*\left(d, \vec{c}\right) \\ &= \underset{\vec{t}}{\operatorname{argmax}}\ F(d, \vec{c}, \vec{t}) \\ &= \underset{\vec{t}}{\operatorname{argmax}}\ \min_{\vec{x}^-} f\left(d, \vec{c}, \vec{t}, \vec{x}^-\right) \end{aligned} \tag{6}$$

The goal is to design an autonomous training and hyperparameter optimization approach with minimal user intervention. Additionally, the proposed framework should minimize computing time and enable optimization of the FOM metric as a function of tuning conditions in high-dimensional settings (up to 100 variables).

Features	Data Types	Values	Description
Device ID d	integer	0 to 8	Unique ID; identifies devices
Corner	categorical	{TT, 3FF, 3SS}	Process corner of devices
$c_1, \ldots, c_4$	continuous	-1.0 to 1.0	Tuning condition variables
t_1	integer	2 to 13	Tuning knobs for configurable register 1
t_2, t_3	integer	0 to 15	Tuning knobs for configurable register 2, 3
t_4, t_5	integer	0 to 3	Tuning knobs for configurable register 4, 5
t_6, t_7	integer	1 to 2	Tuning knobs for configurable register 6, 7
y_1	continuous	-21.765 to 8.777	Measured result 1 of devices
y_2	continuous	-21.915 to 7.912	Measured result 2 of devices
y_3	boolean	{False, True}	Measured result 3 of devices
y	continuous	-20.602 to 7.387	Aggregated FOM value (here: aggregation of y_1, y_2, y_3)

Table 1: Description of a real-world PSV characterization data set

4 Methodology

4.1 Soft-Aggregation Surrogate Model

In PSV, available data sets typically consist of a limited set of prototype circuits representative for the spread in the manufacturing process. Here, the task is to approximate the underlying function of the data with multiple learning algorithms, each trained on a circuit-specific subset. Furthermore, we expect an unknown number of subsets to describe functions with very different characteristics, e.g., due to

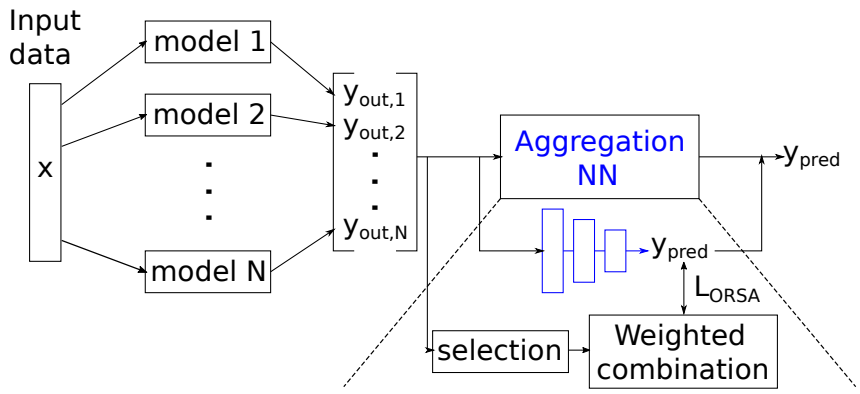


Figure 1: Outlier Robust Stacked Aggregation. The overview shows the selection, the combination process, and the loss calculation (see Eq. 7).

process variation or defects. Corresponding ensemble members, so-called outliers, can heavily influence the approximation. Thus, we have proposed a novel ensemble learning technique called Outlier Robust Stacked Aggregation (ORSA) [9]. It aims to find an approximation that is robust to outliers and represents a non-pessimistic worst-case or non-optimistic best-case approximation, respectively. Therefore, we define a 'soft-min' or 'soft-max' function in place of a maximum or minimum operator. We train a NN to learn this 'soft-function' in a two-stage process. First, we select a subset of ensemble members with the best or worst predicted performance. Second, we combine these members and define a weighting that uses the properties of the Local Outlier Factor (LOF) [27] to increase the influence of non-outliers and to decrease outliers. The weighting ensures robustness to outliers and makes sure that approximations are representative for the majority of circuit-types. Fig. 1 shows an overview of the approach.

$$\begin{aligned} L_{ORSA} &= \sum_{i=0}^{k} w_i * (y_{out}^i - y_{pred})^2 \\ &= \frac{1}{\sum_{j=0}^{k} LOF_j} \sum_{i=0}^{k} \frac{1}{LOF_i} * (y_{out}^i - y_{pred})^2 \end{aligned} \tag{7}$$

The combination process is very flexible because it allows arbitrary selections of different types for the circuit-specific models (model 1, ..., model N in Fig. 1). The loss L_{ORSA} given in Eq. 7 is used to train the (stacked) Aggregation NN. It is a weighted least-square formulation with a weighting that uses properties of the LOF to ensure robustness. The weights w_i in Eq. 7 are calculated as the reciprocal of the LOF and normalized such that $\sum_i w_i = 1$. Note that there is no ground truth for the approximation task. Thus, ORSA is an unsupervised ensemble learning method. Conventional approaches to mitigate faulty circuits depend on outlier detection and neglecting faulty circuits. In contrast, our approach does not discard whole circuits and thus can exploit more data than possible otherwise [9].

4.2 Self-Learning Tuning with Reinforcement Learning

In this section, we introduce a novel approach to the problem of robust performance tuning. The self-learning method is based on DRL and aims to automatize the tuning process. Additionally, it is scalable to high-dimensional tasks - which is especially important to cope with the increasing complexity of modern circuits - and flexible because we learn a mapping from given conditions to optimal tuning configurations. In our work, we call this mapping tuning law. Learning a tuning law goes beyond current state-of-the-art approaches that only approximate point-wise solutions, which are very inefficient given the tight schedule in PSV. However, the longer training time of learning-based methods is affordable for the benefit of fast inference (prediction) in later stages. The proposed approach applies to circuit-specific and circuit-independent tuning settings.

We are following the path of learning to optimize and leverage DRL methods as in [16, 28, 29, 30] to train a problem-specific optimization algorithm. We define robust performance tuning as a RL task to enable learning of optimization methods for mixed-type problems (see Def. 2.1 and Sec. 2.1). Similar to [16],

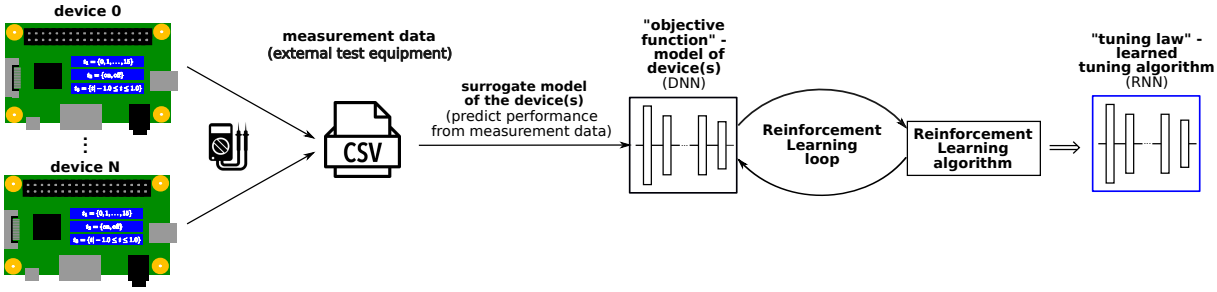


Figure 2: Self-learning tuning in PSV. The tuning law is learned iteratively within the RL loop. We use a performance measure (aggregated from measurement outputs, e.g., FOM) of the circuits to train and evaluate a tuning algorithm in form of a RNN.

the objective function is a black box. In our application, the objective is solely described by the given data set and the surrogate model, respectively. Instead of optimizing network parameters we aim to find inputs that produce the best outputs (in terms of specification goals) without relying on gradients or approximations of gradients resembling the scenario of black-box adversarial attacks, e.g., in [31]. Fig. 2 shows an overview of our approach. We iteratively train an optimization algorithm on a surrogate model in the outer reinforcement learning loop. The trainable optimization model itself iteratively improves the performance measure of the surrogate in the inner loop. Performance improvement is used as feedback (e.g., reward) to the outer training loop. The surrogate model allows us to formulate tuning as an iterative optimization process and train the tuning law efficiently in the reinforcement learning loop. The final goal is to find the solution to Eq. 6 (see Sec. 3). The reward definition we use reflects specified optimization goals of the circuits, e.g., FOM. The computations of our methodology are summarized in Alg. 2:

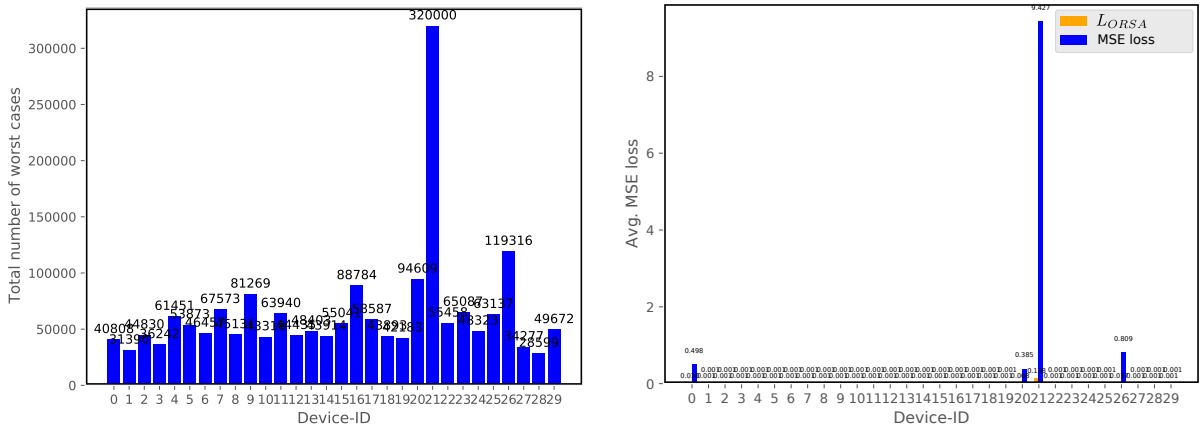
Algorithm 2 Self-Learning Tuning

1. Given current state $\vec{s}_{t,\mathrm{NN}} := \{(\vec{x}_{t-1}, f(\vec{x}_{t-1}))\}$ or
   $\vec{s}_{t,\mathrm{RNN}} := \{(\vec{x}_{t-1}, f(\vec{x}_{t-1}), \vec{h}_{t-1})\}$
   propose update $\vec{x}_t$
   ($\vec{h}_{t-1}$: internal state of RNN at time step $t-1$)
2. Observe response of environment, e.g., $f(\vec{x}_t)$
3. Update internal statistics to produce $\vec{s}_{t+1}$ and r_{t+1}

The update rule (e.g., described by a DRL agent) in 1. step is defined by a (R)NN parameterized by Θ, such as $f_{\theta,(\mathrm{R})\mathrm{NN}} := \vec{s}_t \rightarrow \vec{x}_t$. The architecture consists of a two-layer LSTM network with tanh activation units. The parameters Θ are updated by applying reinforcement learning algorithms, e.g., REINFORCE, to maximize the cumulative reward. For similar computing budgets, our approach can have superior performance showing a much faster convergence speed than classical methods, especially for difficult optimization problems. Thus, learning to optimize has the potential to overcome the limits of current analytical methods in PSV [7].

5 Experiments and Results

In the following, we present the experiments and results of our methodology. We will compare the results of the real-world data set (see Sec. 3) with artificially generated data because we typically have no ground truth about defective device IDs or optimal achievable performance (e.g., FOM) in the real-world scenario.



(a) Results of the selection process for a robust, non-pessimistic worse-case approximation.

(b) Comparison of MSE loss with our weighted loss formulation L_{ORSA} (see Eq. 7).

Figure 3: Results of the selection and weighted combination process for the artificial data set.

5.1 ORSA

5.1.1 Artificial data set

This section summarizes the results of our ensemble learning technique ORSA using an artificially generated data set. We generate the data with the expertise of Advantest and the following properties:

- Data of 30 circuits with different characteristics, 10k samples per circuit
- 4 outlier circuits (IDs: 0, 20, 21, 26) representing different defective behaviors, e.g.,
  1. Constant offsets
  2. Smooth, Gaussian-shaped noise (non-zero in random area of the parameter space)
  3. Probabilistic offsets (constant offset with probability $p_1 < 1$ and no offset with $1 - p_1$)
  4. Probabilistic offsets with randomly scaled amplitudes

Fig. 3 shows a qualitative analysis of our method. Fig. 3a visualizes the frequency with which ensemble members are selected for k worst circuits. We observe that the outlier circuits are among the most selected devices. Moreover, the effects of the different outlier types get more subtle from 1. to 4. and thus more difficult to detect. This effect is visible in terms of the total number of worse cases as well as in Fig. 3b showing how much our outlier robust loss formulation L_{ORSA} in Eq. 7 reduces the loss contribution of individual circuits. The more subtle the effects of outlier get, the smaller the reduction in loss contribution. However, it is visible that our approach can detect the effects of outliers and reduce their contribution in the worst-case approximation. Fig. 4 visualizes the prediction of the Aggregation NN for different hyperparameters. It is noteworthy that our method can learn a robust approximation between the absolute minimum and the average of circuit outputs depending on the value of k. For $k = 1$, we get the minimum, and for $k = 30$, we get the average (k defines how many circuits to select in each step).

5.1.2 Real-world data set

Fig. 5 and Fig. 6 visualize the same analysis as in the previous section. Instead of artificial data, we use a real-world data set (see Sec. 3) provided by Advantest. However, we have no ground truth of outlier circuits and their characteristics. Whereas the output prediction of the Aggregation NN in Fig. 6 looks similar w.r.t. robust worse-case approximation, we observe different circuit characteristics in Fig. 5. Fig. 5a does not clearly show outlier circuits. Instead, we may have different classes of devices that represent bad, typical, and good performance. Together with the individual loss contributions in Fig. 5b, we may conclude that ID 6 is an outlier similar to constant offset outliers. Other circuits with reduced loss show subtle effects similar to outliers with probabilistic offsets. Again, this aligns with the expertise of Advantest.

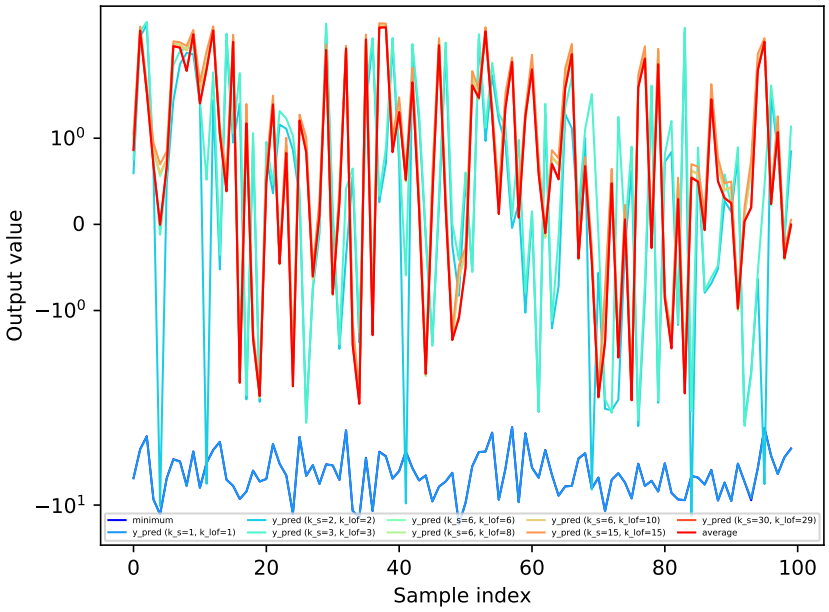
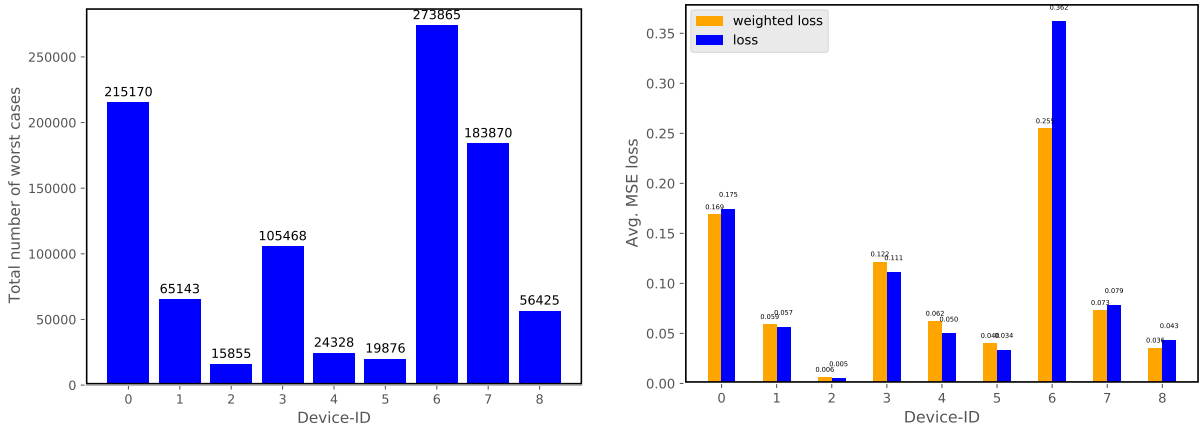


Figure 4: Output prediction of the Aggregation NN (see Fig. 1) for 100 random input samples of the artificial data set. We compare different hyperparameter settings of our method with the minimum and average of the circuit outputs. Depending on k, our method can learn an optimistic and representative worst-case approximation with a suitable trade-off between minimum and average.



(a) Results of the selection process for a robust, non-pessimistic worse-case approximation.

(b) Comparison of MSE loss with our weighted loss formulation L_{ORSA} (see Eq. 7.)

Figure 5: Results of the selection and weighted combination process for the real-world data set.

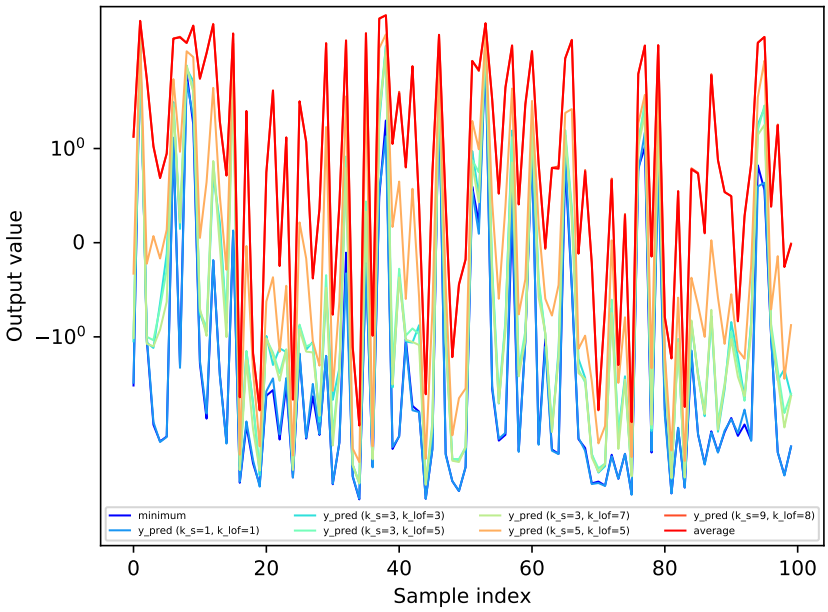


Figure 6: Output prediction of the Aggregation NN (see Fig. 1) for 100 random input samples of the real-world data set. We compare different hyperparameter settings of our method with the minimum and average of the circuit outputs.Depending on k, our method can learn an optimistic and representative worst-case approximation with a suitable trade-off between minimum and average.

5.2 Tuning Law

In general, multiple tuning settings are possible. We can either learn a circuit-independent or a circuit-specific tuning law. In both cases, we can define a parameter-dependent (e.g., dependent on conditions $\vec{c}$) or a parameter-independent tuning law. In the following sections, we focus on circuit-independent and parameter-dependent tuning laws because they allow the highest increase in efficiency and flexibility. In addition, they are less expensive than circuit-specific tuning laws but probably suboptimal w.r.t. specific circuit characteristics.

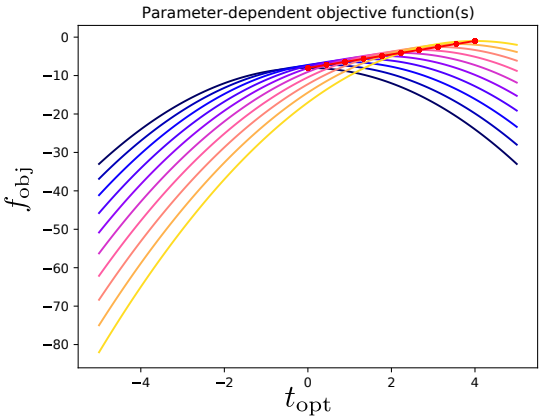
5.2.1 Analytical objective functions

In this section, we present the results of a parameter-dependent tuning law for an analytically known objective function $f(t, c_{free}) = -(t - s_x(c_{free}))^2 + s_y(c_{free})$ with arbitrary functions $s_x(c_{free})$ and $s_y(c_{free})$ that cause a shift in x and y direction dependent on the value of c_{free}. Fig. 7a shows the objective for $s_x(c_{free}) = -4(c_{free} - 1)^2 + 4$ and $s_y(c_{free}) = 8 - 7c_{free}$. The goal is to find $t^*(c_{free}) = \underset{t}{\text{argmax}} f(t, c_{free})$ which is shown in red, see Fig. 7a. Fig. 7b visualizes the difference between predicted optimum and ground-truth value. In this low-dimensional setting, we can visually analyze that our tuning law almost perfectly matches the true optimum values after 10k training iterations.

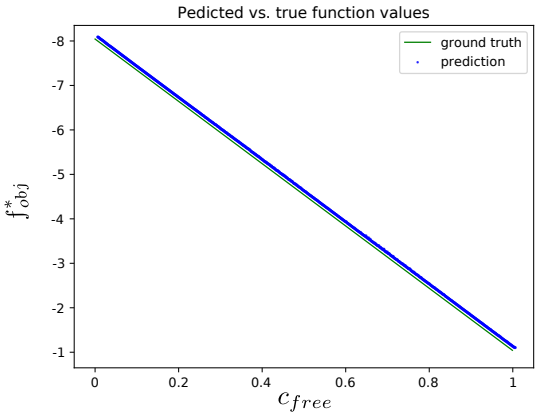
5.2.2 Real-world black-box objective functions

In the following experiments, the objective function is no longer a closed-form expression but a black box given by the real-world data set (e.g., test measurements of the circuits). As described in previous sections, we use this data set to learn a surrogate model (here: soft-aggregation model) of the black-box objective and a corresponding tuning strategy. Fig. 8a and Fig. 8b visualize the results after 10k training iterations of a circuit-independent but parameter-dependent tuning law for one or respectively two tuning conditions c_1 and c_2.

Both plots in Fig. 8 show almost constant performance w.r.t. the achievable FOM. Moreover, tuning results are almost independent of the tuning conditions c_1 and c_2. The analysis of the remaining input variables $\vec{x}^-$ in Eq. 6 reveals that worse-case values are distributed in areas around corners of the parameter space. These observations match the expertise of Advantest. Furthermore, both plots show

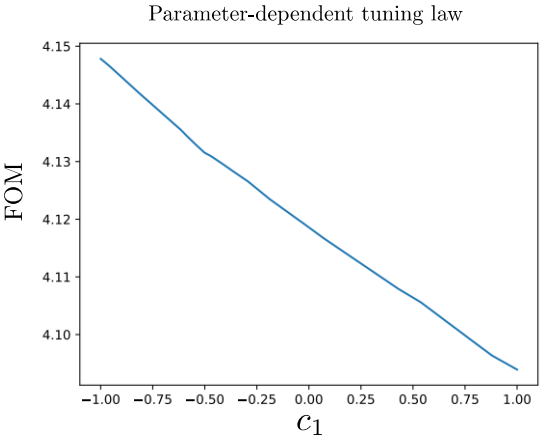


(a) Parameter-dependent quadratic objective function

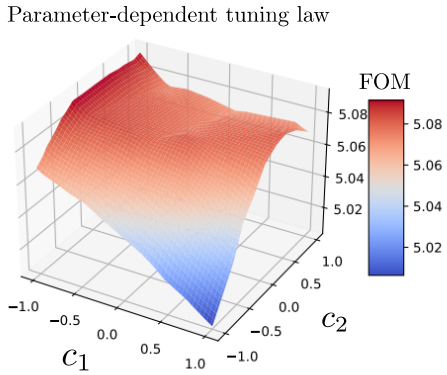


(b) Predicted optimum values vs. ground truth maxima

Figure 7: Results of tuning law for quadratic objective function.



(a) Parameter-dependent tuning law $t^* = t(c_1)$



(b) Parameter-dependent tuning law $t^* = t(c_1, c_2)$

Figure 8: Results of parameter-dependent tuning laws for real-world data.

that the predicted tuning knobs, which are in most parts constant, match the given variations in circuit characteristics well and fulfill the specified requirements, e.g., FOM larger 1. Finally, modeling the tuning law with a DNN or RNN allows a significant speed up after the initial training.

6 Conclusion and Outlook

We conclude that our self-learning tuning approach is suitable in learning a tuning law and solve the task of robust performance tuning in the given real-world setting. The proposed method fulfills the specified requirements: the soft-aggregation surrogate ensures robustness, and represents a non-pessimistic approximation, and the RL formulation allows us to learn parameter-dependent tuning laws that are fast and efficient in inference (e.g., prediction). The next steps of our work include:

- Explore scalability of our methodology w.r.t. high-dimensional optimization tasks (analytical benchmark functions and real-world data sets) and an increased number of operating conditions
- Explore influence of different surrogate models in the subsequent optimization process
- Increase efficiency of the DRL training and explore transferability to similar tuning settings
- Analysis of learned tuning law, e.g., sensitivity analysis or inner working mechanisms
- Apply the proposed approach to different tasks, e.g., calibration of parameterized transistor models (collaboration with P9)

- Integration of our software library into Advantest's Cloud Solutions which allows possibility to include variable selection approach of P6
- Interactive visualizations that allows validation engineers to interact with the learning process of the tuning law, e.g., to include expert knowledge or interrupt the learning process (collaboration with P2)

References

[1] Subhasish Mitra, Sanjit A Seshia, and Nicola Nicolici. Post-silicon validation opportunities, challenges and recent advances. In *Design Automation Conference*, pages 12–17. IEEE, 2010.

[2] Prabhat Mishra and Farimah Farahmandi. *Post-Silicon Validation and Debug*. Springer, 2019.

[3] H Ramakrishnan, S Shedabale, G Russell, and A Yakovlev. Analysing the effect of process variation to reduce parametric yield loss. In *2008 IEEE International Conference on Integrated Circuit Design and Technology and Tutorial*, pages 171–176. IEEE, 2008.

[4] S Yerramilli. Addressing post-silicon validation challenge: Leverage validation and test synergy. In *Keynote, Intl. Test Conf*, 2006.

[5] Priyadarsan Patra. On the cusp of a validation wall. *IEEE Design & Test of Computers*, 24(2):193–196, 2007.

[6] Prabhat Mishra, Ronny Morad, Avi Ziv, and Sandip Ray. Post-silicon validation in the soc era: A tutorial introduction. *IEEE Design & Test*, 34(3):68–92, 2017.

[7] Peter Domanski, Dirk Plüger, Jochen Rivoir, and Raphaël Latty. Self-learning tuning for post-silicon validation. *arXiv preprint arXiv:2111.08995*, 2021.

[8] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. *Introduction to derivative-free optimization*. SIAM, 2009.

[9] Peter Domanski, Dirk Pflüger, Raphaël Latty, and Jochen Rivoir. Orsa: Outlier robust stacked aggregation for best-and worst-case approximations of ensemble systems. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1357–1364. IEEE, 2021.

[10] Igor Griva, Stephen G Nash, and Ariela Sofer. *Linear and nonlinear optimization*, volume 108. Siam, 2009.

[11] Juliane Müller, Christine A Shoemaker, and Robert Piché. So-mi: A surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. *Computers & Operations Research*, 40(5):1383–1400, 2013.

[12] Sun Hye Kim and Fani Boukouvala. Surrogate-based optimization for mixed-integer nonlinear problems. *Computers & Chemical Engineering*, 140:106847, 2020.

[13] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[14] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

[15] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

[16] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, pages 748–756. PMLR, 2017.

[17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[18] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*,

2021.

[19] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[20] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[21] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Deep reinforcement learning: an overview. In *Proceedings of SAI Intelligent Systems Conference*, pages 426–440. Springer, 2016.

[22] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[23] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] Zihan Ding and Hao Dong. Challenges of reinforcement learning. In *Deep Reinforcement Learning*, pages 249–272. Springer, 2020.

[26] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

[27] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[28] Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning*, pages 2247–2255. PMLR, 2017.

[29] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, pages 3751–3760. PMLR, 2017.

[30] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[31] Chuan Guo, Jacob Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Weinberger. Simple black-box adversarial attacks. In *International Conference on Machine Learning*, pages 2484–2493. PMLR, 2019.