Invited: Efficient Reinforcement Learning for Automating Human Decision-Making in SoC Design

Shankar Sadasivam
Qualcomm Technologies, Inc.
ssadasiv@qti.qualcomm.com

Zhuo Chen*
Carnegie Mellon University
zhuochencmu@gmail.com

Jinwon Lee
Qualcomm Technologies, Inc.
jinwonl@qti.qualcomm.com

Rajeev Jain
Qualcomm Technologies, Inc.
rajeevj@qti.qualcomm.com

ABSTRACT

The exponential growth in PVT corners due to Moore's law scaling, and the increasing demand for consumer applications and longer battery life in mobile devices, has ushered in significant cost and power-related challenges for designing and productizing mobile chips within a predictable schedule. Two main reasons for this are the reliance on human decision-making to achieve the desired performance within the target area and power budget, and significant increases in complexity of the human decision-making space. The problem is that to-date human design experience has not been replaced by design automation tools, and tasks requiring experience of past designs are still being performed manually.

In this paper we investigate how machine learning may be applied to develop tools that learn from experience just like human designers, thus automating tasks that still require human intervention. The potential advantage of the machine learning approach is the ability to scale with increasing complexity and therefore hold the design-time constant with same manpower.

Reinforcement Learning (RL) is a machine learning technique that allows us to mimic a human designers' ability to learn from experience and automate human decision-making, without loss in quality of the design, while making the design time independent of the complexity. In this paper we show how manual design tasks can be abstracted as RL problems. Based on the experience with applying RL to one of these problems, we show that RL can automatically achieve results similar to human designs, but in a predictable schedule. However, a major drawback is that the RL solution can require a prohibitively large number of iterations for training. If efficient training techniques can be developed for RL, it holds great promise to automate tasks requiring human experience. In this paper we present a Bayesian Optimization technique for reducing the RL training time.

*This author's contributions to this paper were made while he was employed by Qualcomm Technologies, Inc.

DAC '18, June 2018, San Francisco

ACM ISBN 978-1-4503-5700-5/18/06.
https://doi.org/10.1145/3195970.3199855

ACM Reference Format:
Shankar Sadasivam, Zhuo Chen, Jinwon Lee, and Rajeev Jain. 2018. Invited: Efficient Reinforcement Learning for Automating Human Decision-Making in SoC Design. In *DAC '18: DAC '18:The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3195970.3199855

1 HUMAN-DECISION MAKING IN SOC DESIGN FLOW

Despite advances in design automation over the past 30 years, there remain key tasks in the SoC design and productization flow that have to be done manually by human designers, because the design automation tools are not perfect and cannot ensure that the desired performance is met with desired power targets and cost targets. Therefore, human designers have to step in after the tools have done their job to do manual optimization and design changes to meet the product targets.

The question arises, is this always going to be a fundamental limitation or can we automate the tasks that are still done by human designers? In this section we examine two of these tasks, one in design and one in productization, and understand why designers have to step in.

Figure 1 shows a typical SoC design and productization flow. While tools exist to assist in each task, a limitation of these tools is their inability to guarantee that the requirements, especially timing and power requirements will be met. Lets examine two tasks that involve human decision-making and consume significant time in the overall product release cycle: timing closure (design task) and run-time power optimization (productization task).

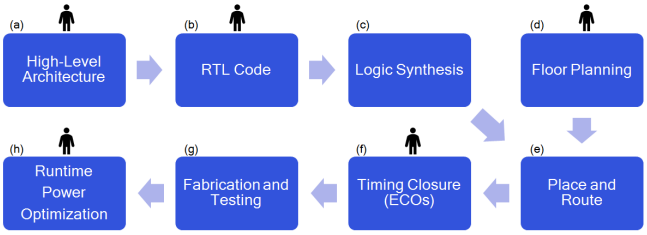

Figure 1: SoC design and productization flow showing human decision-making tasks.

(1) **Run-time power optimization (Figure 1(h)):** To minimize power in mobile devices, various techniques are used on the chips today to dynamically adjust the clock rate based on performance requirements, to get the lowest power with the desired performance. This is known as dynamic clock and voltage scaling or DCVS. Although several DCVS algorithms have been developed in the design automation community to perform the selection of the operating point, the parameters of the algorithms have to be manually adjusted for every chip and for the desired applications [1]. The reason is that the parameter settings are highly dependent on the application, the architecture and the technology node of the chip, which changes from generation to generation. This parameter tuning process relies entirely on human experience and decision-making and is time consuming. This tuning time gates the release of the chip. As technology scales and applications on mobile devices grow, this tuning consumes more time.

(2) **Timing Closure (Figure 1(f)):** Typically, both the logic synthesis and place and route tools attempt to meet specified timing constraints (performance) with minimum power and area. However, in reality, after initial place and route there are still a large number of timing violations that have to be corrected through so-called ECO fixes. While EDA tools exist to assist with these ECOs, the number of violations on large SoCs exceeds the number of violations that can be fixed within the tape-out schedule. Hence the designers have to select a subset of violations that can be fixed within the schedule, while ensuring that all violations are fixed. Currently designers rely on past experience in deciding which subset of violations to provide to the ECO fix tool. It takes several iterations to remove all violations and each iteration can take more than a day. With exponentially growing PVT corners, this process is even more time consuming since violations have to be fixed across all corners.

2 AUTOMATING HUMAN DECISION MAKING

One of the reasons why many these tasks have not been automated is because they rely on human experience and human judgment to resolve the tradeoffs between power, performance and cost. While there have been several attempts to capture human design experience in the form of design automation tools - most of these techniques have leveraged so called rule-based expert systems, where the rules followed by a human designer are captured in a decision tree and the decision tree is implemented in software [2]. The problem with that is that the same rules and decision-making process don't apply uniformly to every chip and any deviation from the rules and decision process followed on a previous chip require human judgment, defeating the purpose of the expert-system.

So, what we really need are design automation techniques that can not only capture the human experience or learn by experience as human designers do, but also make optimal decisions the way a human designer would do. Now a simple way that designers arrive at optimal choices is to play what-if games by generating options to make the design meet requirements, and then based on the evaluation of each option, decide what is the next option they should exercise. In many ways this is like playing a game where there are certain moves that you know you can make to achieve the outcome, which in the case of a game is to win the game. If you're playing against an opponent, and the opponent could be different each time, you have to be able to determine from the available moves, which move will help you win the game. While the set of moves is the same, which sequence of moves works in a particular game depends on your opponent's moves and your experience.

In SoC design the same analogy holds because every chip is different. Although we may have learned a set of rules (moves) from experience, how we apply them depends a lot on the chip and application at hand. In machine learning there's a technique known as reinforcement learning that has gained momentum in the past few years, which can help to mimic and automate this sequential human-decision making process [3]. However, to do as good a job as a human being would do or perhaps even better, the reinforcement learning tools do require that we generate several design experiences so it can learn from those experiences the way a human would. That means that we need to generate many design alternatives and allow the reinforcement learning tool to evaluate the outcomes of all those different alternatives so it knows from experience what choices result in the best outcome, the best here implying that we meet a certain performance target with the lowest power and cost. In this paper we present reinforcement learning techniques that can help in automating human design decisions, while achieving similar results as manual designs. However, we discovered that the limitation is in generating a sufficient number of design alternatives for the tool to learn from within the given schedule. This is known as the training process and is the main focus of this paper: how can we efficiently train RL tools to automate design tasks that depend on human intuition? In general machine learning tools need a much larger set of experiences (tens of thousands) to develop the same 'intuition' that human designers do with much fewer experiences. If we can solve this problem, we can benefit from the predictable execution time that reinforcement learning offers.

Going back to the analogy with game playing, the efficacy of applying reinforcement learning to learning how to win a game has been established by several people, including most recently by researchers at Google Deepmind, in the case of the very difficult game Go [4]. However, in the case of games, it is easy to generate experiences by having the computer play millions of games with itself in a very short amount of time. By contrast generating one chip design option to learn from can take a day or more, so generating 1000s of design options to learn from can take years. Therein lies the problem in the case of design automation with reinforcement learning. To successfully apply reinforcement learning, we need creative methods to cut down the number of experiences the tool needs to learn from. In this paper we present a Bayesian optimization based RL training method to address this problem.

3 REINFORCEMENT LEARNING APPLIED TO RUN-TIME POWER OPTIMIZATION

Most CPU based SoCs have built-in hardware to measure the real-time processing requirements for applications. Based on these measurements run-time tools (i.e., run-time control algorithms) choose clock frequency and voltage settings (e.g., for the CPU and the DDR) that deliver the desired performance while minimizing power [1]. Two major drawbacks with existing tools is that their parameters have to be re-tuned for every new chip based on the SoC architecture and technology node, and secondly the control algorithms are rule-based and the rules may have to be modified based on the applications. The parameter tuning process is usually done manually based on human experience as part of the post-silicon productization. The parameters have to be iteratively adjusted, while measuring power and performance on a variety of applications, to converge on a setting that meets both performance and power requirements. Based on experience, human designers use intuition in guiding this process. We can abstract this problem as shown in the below sections (see Figure 2).

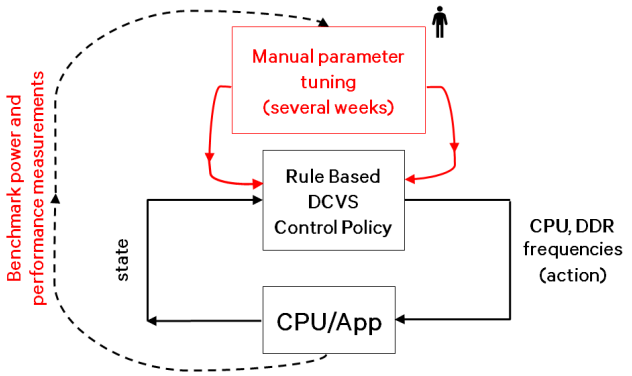

Figure 2: DCVS manual parameter optimization.

3.1 Definitions

- s_i: The state at time t_i is the value of the processor measurements at time t_i
- a_i: Action taken by the DCVS algorithm at time t_i to select the clock frequency for the next time step
- π_θ: Policy followed by the DCVS algorithm to map the state s_i in to the action a_i
- θ: Policy parameters
- r_i: Reward or the outcome at time t_i, corresponding to action a_i, i.e., the measured performance and power

3.2 Problem Formulation

At run time, the DCVS rule-based control algorithm observes state s_i and takes action a_i based on the policy π_θ. Prior to release of every new chip, human designers have to iteratively tune the parameter values (θ) of the policy to ensure desired performance for target applications while meeting power targets. As the power margins get tighter and number of applications grows, this manual tuning process becomes more challenging. Mathematically we can formulate the parameter tuning problem as follows (see Figure 2):

For all applications, given the reward r_i and current value of θ in the i^{th} tuning iteration, and the possible values of θ, select a new value of θ to improve r_{i+1}. When r_i meets or exceeds the product requirements, the value of θ is set.

Generally, designers rely on experience how to adjust θ to get a desired outcome, since there is no direct mathematical expression or model relating θ to power and performance. Thus, this tuning process is time consuming as the designers have to go through several iterations to get the right parameter θ that delivers the performance and power targets. This manual process is repeated for every chip.

3.3 Reinforcement Learning Solution

The efficacy of the DCVS tool is limited by the ability of the human designers to adapt the parameters (θ) of the tool for new chips and applications. The schedule for delivery of the chip is gated by the time to tune θ. With technology scaling, it is more challenging to achieve low power and the tuning process can take longer. The question we explore is whether machine learning can learn the tuning process and do the tuning in a fixed amount of time regardless of technology node or desired power targets.

The above problem statement is a natural fit for the reinforcement learning (RL) technique as illustrated in Figure 3 [1, 5]. The DCVS control algorithm can be represented by a policy that is trained via RL, i.e., optimizes a desired reward r_i, by choosing actions a_i in response to the value of state s_i. The RL training replaces the manual tuning process described above. In the training phase, based on its experience with past iterations, reinforcement learning will discover the optimal policy π_θ for mapping the current state to the next action. In contrast to current rule-based approaches, the RL model itself can be agnostic to the application and automatically be retrained for new applications and new chips as needed. From our experience the training time is mainly gated by the training data collection which can be a fixed amount of time independent of chips and applications.

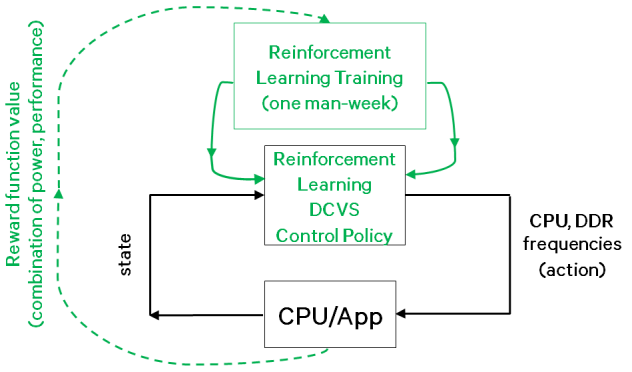

Figure 3: Reinforcement learning based DCVS parameter optimization.

If we can train the RL model in a constant amount of time for any chip (less than the time currently required for manual tuning), while

achieving power and performance similar or better than manual tuning, then we can ensure a predictable schedule for release of the chip for future generations.

The RL training is an iterative process in which the training algorithm has to exercise the DCVS loop of action → state → action multiple times, measuring the state and reward in each loop, and adapting the policy till the desired reward is achieved. Therefore, RL training time is gated by: (1) the number of iterations required to achieve the desired reward, and (2) the time for each iteration.

The time for each iteration is further determined by time to run the applications for which the DCVS control needs to be optimized. Typically, applications run for several minutes. For each application the RL training algorithms will require multiple iterations to converge to the desired reward. Based on other attempts to apply RL to this problem [5] and our own work, it can take approximately 10,000 iterations (each iteration comprising of 100 policy rollouts) to train on a single application, which results in a training time of one month per application. Assuming we need to train on 10 different type of applications to get a robust DCVS model, it can take almost a year to train, which is not practical. To get an idea of the efficacy of the RL approach, we conducted training experiments with a representative set of synthetic applications of shorter duration (few milliseconds each) and discovered that the RL tool can learn and produce DCVS settings that result in similar or lower power for the same performance as manual settings (tested on the same synthetic workloads). The results were reproducible with a predictable schedule. The training algorithm we used was the Cross-Entropy Method (CEM) [6].

4 REDUCING RL TRAINING TIME

To benefit from the RL approach by reducing number of training iterations, we have developed a Bayesian Optimization (BO) method for the RL training. The proposed iterative-BO method for RL-based DCVS algorithm training, results in a speedup in training time up to ∼ 37×, relative to direct RL-DCVS training using CEM. First, we compress the RL-DCVS model (Figure 3) to fewer parameters, which not only speeds up RL training, but also enables the application of BO. Second, we propose an iterative-BO method that uses a dimensionality reduction strategy to enable better BO convergence properties. The improved version of iterative-BO method, which we call *iterative-BO with restart*, additionally leverages a novel history forgetting strategy to achieve an increased speedup of ∼ 37× in RL-based DCVS training.

One of the major challenges in BO is the curse of dimensionality [7, 8]. Several researchers have tried to alleviate this issue, but with strong assumptions that are not applicable to our problem [7, 9–12]. For the DCVS problem specifically, we address the curse of dimensionality issue by (a) compressing the model, and (b) decomposing the BO algorithm to iteratively optimize over subsets of model parameters focusing on different parts of the underlying system. We provide additional details regarding algorithm and methodology in the next section.

4.1 Iterative Bayesian Optimization as a Fast Proxy for RL Training

DCVS parameter tuning can be modeled as a Markov Decision Process [5, 13] that traditionally admit solutions via RL [14]. Since we care about both the performance and power of the system, we define the reward r as the product of a performance reward and a power reward. We define the application execution time (proxy for performance) and average power consumption (proxy for mobile device battery life), with RL-DCVS, as $\mathcal{T}_{RL}$ and $\mathcal{P}_{RL}$, respectively, and the execution time and power under a rule-based DCVS policy as $\mathcal{T}_{rule}$ and $\mathcal{P}_{rule}$. Performance reward $r_{perf} = 1$ when $\mathcal{T}_{RL} <= \mathcal{T}_{rule}$ and $r_{perf} = \mathcal{T}_{rule} - \mathcal{T}_{RL}$, which is a negative value, when $\mathcal{T}_{RL} > \mathcal{T}_{rule}$. Power reward $r_{power} = max(0, 1 - \frac{\mathcal{P}_{RL}}{2\mathcal{P}_{rule}})$ which prefers lower power than $\mathcal{P}_{rule}$. Accordingly, rule-based DCVS has a reward of 0.5, and any algorithm with a reward higher than 0.5 achieves better energy efficiency than rule-based DCVS. As mentioned previously, we denote the state of the system by s, and it comprises of several system state counter values.

Cross-Entropy Method (CEM) is a well-known RL algorithm [6], and we use it as the baseline RL-DCVS method for our problem. We model each device component (e.g., CPU and DDR) control policies separately as they are distinct components in the mobile device, and this also helps dealing with the BO curse of dimensionality. The model we use is:

$$f_{cmpt} = \text{argmax}(\theta_{cmpt} \cdot s)$$

where, f is the index corresponding to the component's selected frequency level, *cmpt* is device component (CPU or DDR), θ_{cmpt} is a two-dimensional parameter matrix. We call this the CEM-1 model, and it has 184 parameters, as in our case, $N_{fcpu} = 13$ CPU frequencies and $N_{fddr} = 10$ DDR frequencies. In our experiments with the CEM-1 model it takes 4,000 iterations of training for the reward with RL-DCVS to surpass the reward with a rule-based DCVS method. In order to speed up training, we first propose a simplified model:

$$f_{cmpt} = round[(N_{fcmpt} - 1) \cdot Sigmoid(\hat{\theta}_{cmpt} \cdot s)]$$

where $\hat{\theta}_{cmpt}$ is a one-dimensional parameter vector. We use $(N_{fcmpt} - 1)$ to scale up the sigmoid output because the frequency index starts from zero. We call this the CEM-2 model and it has only 16 parameters. Our experimental results (excluded due to space constraints) show that CEM-1 and CEM-2 deliver similar energy and performance.

We then apply BO to speed up the training of our RL-DCVS algorithm, that uses the CEM-2 model described above. Although CEM-2 drastically reduces the number of parameters from 184 to 16, it is still not small enough in dimensionality for BO to be effective [7, 10]. To tackle this problem, we propose the iterative-BO method (see Algorithm 1 with *Method* set to *iterative-BO*) to decouple CPU and DDR model optimization steps, so that we effectively only optimize eight parameters at a time, *i.e.*, while the CPU model parameters are being optimized using BO, the DDR model parameters are held fixed to the optimal values from the previous iteration (and vice versa). In each iteration, we optimize each component using $M = 50$ iterations, which is determined by experiments. We use a Gaussian

Process prior for our BO algorithm. Since the choice of acquisition function, covariance kernel and their hyperparameters are problem dependent, we did a grid search to optimize the same.

Algorithm 1 Pseudocode of iterative-BO method (with restart)

Input: $MaxIter$, N_{fcpu}, N_{fddr}, $Method$, M
Output: $\hat{\theta}_{CPUbest}$, $\hat{\theta}_{DDRbest}$, $MaxReward$

//Randomize CPU, DDR model parameters at iteration zero
$\hat{\theta}_{CPUbest} \leftarrow random()$; $\hat{\theta}_{DDRbest} \leftarrow random()$;
$cmpt \leftarrow [CPU,\ DDR]$; $MaxReward \leftarrow 0$;
Initiate BO_{CPU}, BO_{DDR}
for $i \leftarrow 1; i \leq MaxIter; i \leftarrow i+1$ **do**
  for $j \leftarrow 0; j \leq 1; j \leftarrow j+1$ **do**
    if $Method$ == iterative-BO **then**
      $reward, \hat{\theta}_{cmpt[j]} \leftarrow$ resume $BO_{cmpt[j]}$ on $System(\hat{\theta}_{cmpt[j]}, \hat{\theta}_{cmpt[1-j]} \leftarrow \hat{\theta}_{cmpt[1-j]best})$ for M iterations {Resume $BO_{cmpt[j]}$, i.e., keep previous observations}
    else if $Method$ == iterative-BO with restart **then**
      $reward, \hat{\theta}_{cmpt[j]} \leftarrow$ restart $BO_{cmpt[j]}$ on $System(\hat{\theta}_{cmpt[j]}, \hat{\theta}_{cmpt[1-j]} \leftarrow \hat{\theta}_{cmpt[1-j]best})$ for M iterations {Restart $BO_{cmpt[j]}$, i.e., clear previous observations}
    end if
    if $reward > MaxReward$ **then**
      $MaxReward \leftarrow reward$; $\hat{\theta}_{cmpt[j]best} \leftarrow \hat{\theta}_{cmpt[j]}$;
    end if
  end for
end for

When iterating between BO_{CPU} and BO_{DDR}, note that we may want to retain the observations from previous iterations as they may help guide the parameter learning. However, our experiments showed, that because at the beginning of iterative-BO, parameters are close to random, they change drastically after a few iterations. Those early observations therefore quickly become wrong estimations of the reward function and hinder the optimization process. Accordingly, we propose *iterative-BO with restart*, which restarts BO at the beginning of each iteration (see Algorithm 1 with *Method* set to *iterative-BO with restart*). With restart, results show greatly improved speedup of RL-based DCVS training (see Figure 4), confirming that our intuition to forget the historical BO context is helping significantly.

4.2 Experiments and Results

4.2.1 Experiment Setup. We show the effectiveness of our proposed methods by testing on extensive combinations of workloads. The power and performance values of the rule-based DCVS algorithm as well as the features of all workloads are measured from real mobile chipsets. We implement CEM-1 and CEM-2 following [6], and use the BO package: BayesianOptimization [15] for BO optimization. We modify the source code to experiment with different kernel hyperparameters and implement our iterative-BO methods.

4.2.2 Experimental Results. CEM has two hyperparameters: batch size and noise. We performed grid search to find the best value of batch size = 200 and noise = 0.01 that surpasses the rule-based method with fewest iterations. As we pointed out before, the choice of acquisition functions, kernels and their parameters may highly affect the results of BO [16]. We experiment with various choices to determine the best values for them. For the acquisition function, we tried Expected Improvement (EI) and Upper Confidence Bound (UCB), both of which have been widely used [17]. EI has no hyperparameters, while UCB has one hyperparameter κ that trades off between exploitation and exploration. Squared Exponential kernel is often used in BO, however it is considered unrealistically smooth for many engineering problems [17]. As a result, we pick Matern kernel and experiment with hyperparameter $\nu = 0.5,\ 1.0,\ 2.5$ as suggested by [16]. Also, these values compute considerably faster due to the modified Bessel function in Matern [18][16]. Our results show that a GP prior with Matern kernel $\nu = 2.5$ and unit scale length, and UCB with $\kappa = 0.5$ as the acquisition function gives the best results.

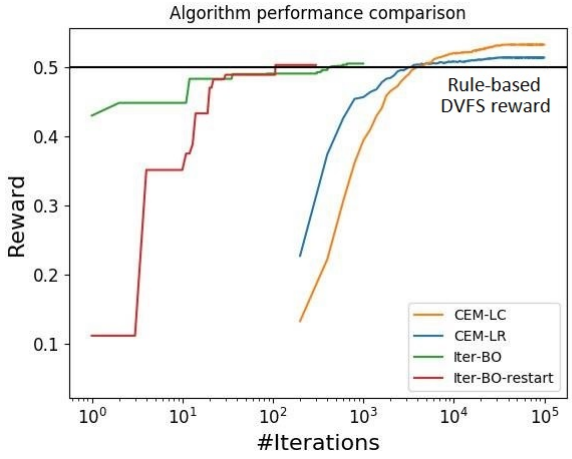

Figure 4: Sample efficiency of various RL based DCVS training methodologies.

CEM and BO solve for the model parameters to maximize the reward r, the indicator of system energy and performance. Rule-based DCVS method has a reward value of 0.5, and this is the value our methods aim to surpass with fewer iterations. Figure 4 shows the reward *vs.* iteration (in log scale) for all four methods under the best parameters chosen above. We can see that all methods surpass rule-based DCVS (horizontal line at 0.5). CEM methods start from 200 iterations because the reward values are evaluated after each batch, which has a size of 200. Compared to CEM-1, model reduction (CEM-2) gives a 1.2× speedup, while iterative-BO delivers 9.1× speedup based on the compressed model. BO on joint CPU+DDR system does not even reach the rule-based DCVS method in the experiment time horizon considered here; we believe this is due to the curse of dimensionality problem for BO. *Iterative-BO with restart* is able to further boost the speedup to

37.4×, which demonstrates that discarding incorrect history helps learn the target function much faster. Initial values do not matter much for *Iterative-BO with restart* as its reward value improves much faster than Iterative-BO.

5 CONCLUSION

Reinforcement Learning can effectively automate manual design tasks that depend on human experience and decision-making provided fast training methods can be found. In the context of chip DCVS control, we have demonstrated how Bayesian Optimization can be used as a fast proxy for Reinforcement Learning to reduce the RL training time. Our future research is aimed at applying this technique to the timing closure and other human decision-making tasks in the SoC design flow.

ACKNOWLEDGMENTS

The authors would like to thank Steve Molloy, Steve Halter, Saravana Kannan, Sejoong Lee, Tauseef Kazi, Dilip Gopalakrishna, Amalendu Iyer, Raju Katari and Premal Shah for their valuable inputs and contributions towards this work.

REFERENCES

[1] X. Chen, Z. Xu, H. Kim, P. V. Gratz, J. Hu, M. Kishinevsky, U. Ogras, and R. Ayoub. Dynamic voltage and frequency scaling for shared resources in multicore processor designs. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–7, May 2013.

[2] In A.F. Schwarz, editor, *Handbook of {VLSI} Chip Design and Expert Systems*. Academic Press, 1993.

[3] Abhijit Gosavi. Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192, 2009.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, and et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354âĂŞ359, 2017.

[5] Zhuo Chen and Diana Marculescu. Distributed reinforcement learning for power limited many-core system performance optimization. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1521–1526. EDA Consortium, 2015.

[6] István Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Learning*, 18(12), 2006.

[7] Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. High dimensional bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning*, pages 295–304, 2015.

[8] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[9] Bo Chen, Rui Castro, and Andreas Krause. Joint optimization and variable selection of high-dimensional gaussian processes. *arXiv preprint arXiv:1206.6396*, 2012.

[10] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, Nando De Freitas, et al. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, pages 1778–1784, 2013.

[11] Josip Djolonga, Andreas Krause, and Volkan Cevher. High-dimensional gaussian process bandits. In *Advances in Neural Information Processing Systems*, pages 1025–1033, 2013.

[12] David K Duvenaud, Hannes Nickisch, and Carl E Rasmussen. Additive gaussian processes. In *Advances in neural information processing systems*, pages 226–234, 2011.

[13] Wei Liu, Ying Tan, and Qinru Qiu. Enhanced q-learning algorithm for dynamic power management with performance constraint. In *DATE*, pages 602–605. European Design and Automation Association, 2010.

[14] Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[15] https://github.com/fmfn/bayesianoptimization.

[16] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[17] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[18] http://scikit-learn.org.