Stimulus Diversification and Coverage Closure of 3D NAND Flash with Artificial Intelligence

Goutham Pallela
Micron Technology, Inc.
San Jose, CA, USA
gpallela@micron.com

Peiyao Shi
Micron Technology, Inc.
San Jose, CA, USA
peiyaoshi@micron.com

Srinivas Deshmukh
Micron Technology, Inc.
San Jose, CA, USA
sdeshmukh@micron.com

Vaishnavi Venkatesh
Micron Technology, Inc.
San Jose, CA, USA
vvenkatesh@micron.com

Bill Hughes
VerifAI Inc.
Palo Alto, CA, USA
bill@verifai.ai

Rohit Suvarna
VerifAI Inc.
Palo Alto, CA, USA
rohit@verifai.ai

Sandeep Srinivasan
VerifAI Inc.
Palo Alto, CA, USA
sandeep@verifai.ai

***Abstract*- Given the ever-increasing complexity of 3D NAND Flash designs and tighter tape-out timelines, applying machine learning (ML) algorithms in pre-silicon design verification (DV) has become essential to ensure satisfactory coverage within the project schedule. This work proposes using reinforcement learning (RL) algorithms that use several deep neural networks (DNN) coupled with Gradient Ascent approach to determine desired stimulus settings that cover the highest number of unit coverage bins. A feedback loop is established to train the model from the coverage database to the RL agent and from the RL agent to stimulus generation. The RL agent is trained in several iterations without the actual design-under-verification (DUV), and the trained model is later used on actual DUV to generate the best stimulus that can reach the highest coverage.**

***Index Terms*: Reinforcement Learning, 3D NAND Flash, Deep Neural Networks, Functional Coverage, Gradient Ascent**

I. Introduction

Verifying 3D NAND Flash[1] (NAND) full-chip functionality by meeting coverage goals is a requirement for design tape-out. With rapidly increasing NAND design complexity driven by high-performance demands and diverse requirements, achieving coverage goals before tape-out becomes a severe challenge to the design verification (DV) team. This paper uses machine learning (ML) techniques to generate stimulus or knob settings, aiming to achieve optimal verification coverage[2, 3, 4] with a significantly reduced number of verification cycles. Some specialized categories of ML algorithms, such as Reinforcement Learning (RL), Multi-Arm Bandit algorithms, Gradient Ascent approach and advanced neural net architectures are used to identify optimal settings of stimuli to maximize the coverage with minimal manual intervention. This work aims for faster coverage closure giving more time to the DV team for further expanding the scope of verification.

II. Existing Design Verification Environment

3D NAND Flash (NAND) DV team follows the industry standard Coverage driven Constrained Random Verification methodology to verify NAND designs. DV flow starts with reviewing and understanding NAND Design Specification documents and planning for verification. Later separate verification plans are drafted for each new NAND feature. Sequences, Functional Coverage, and behavioral white-box checks are developed in System Verilog (SV) independently per the verification plan. After test-bench (TB) development, the DV team runs regressions, debugs failures, and analyzes coverage gaps using the coverage reports that are generated by the simulator tool. This cycle repeats till satisfactory coverage goal are reached. Fig. 1 illustrates this process, the part highlighted in green will be explained in later sections.

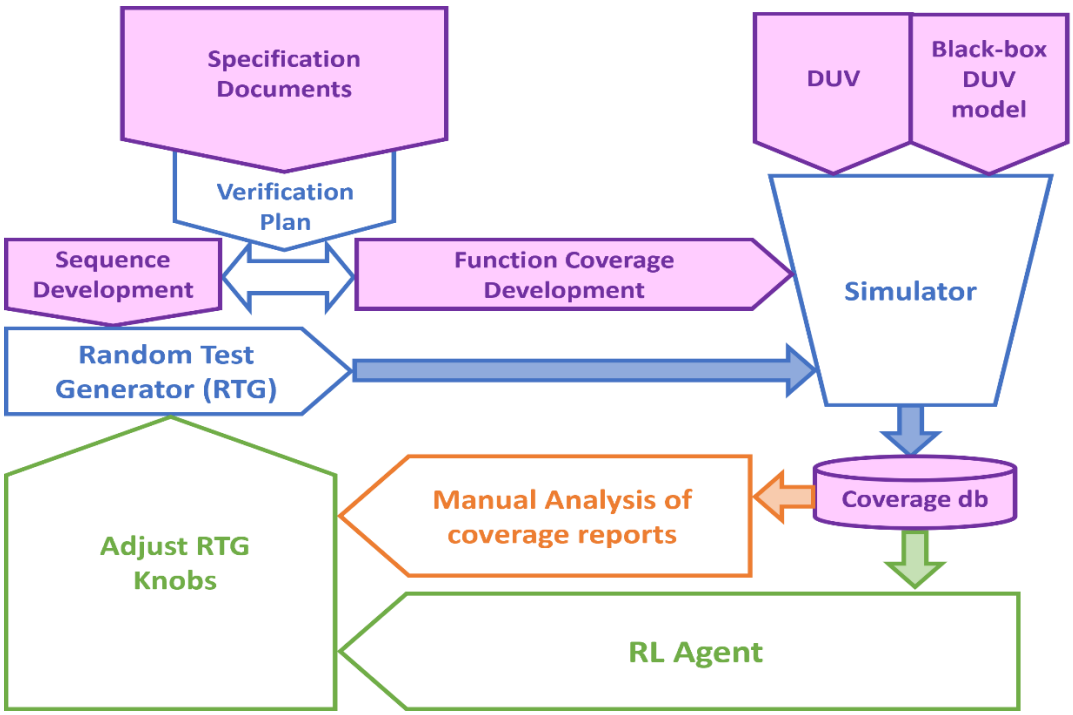

Figure 1. Machine Learning in DV flow

Functional Coverage is a key metric to measure the verification, and it is widely used across the industry to sign-off verification with confidence. Function Coverage is a collection of cover points that are grouped into cover groups by their relevance. Each cover point is manually written by DV engineer to represent each feature, operation, or a state of NAND. Many cover points are often crossed together to cover valid feature, register, or operation combinations. Coverage bins count grows exponentially when cover points are crossed together, resulting in exceedingly high number of bins to be covered before sign-off. The current NAND TB has around 500k bins that need to be covered by random sequence generation. For highest confidence, coverage accumulated throughout the verification cycle is reset to zero for the final tape-out netlist which makes closing coverage more challenging.

The current test-bench issues random operations in random order to NAND by enabling random configurations to cover wide range of command, address, data, and feature requirements. Randomization in SV is enormously powerful that it often needs to be controlled by writing constraints to direct the randomization. Sequences are developed to have knobs (SV parameters) to control the stimulus by steering random SV constraints without modifying the sequences or tests. The details of these knobs will be discussed in the later sections. There are several hundred knobs in the current TB which are used to adjust the stimulus. During the coverage analysis, verification engineers use their best judgment to adjust the values of these knobs and run targeted regressions to fill coverage holes. Considering the indirect or direct relationships between the knobs that can nullify each other's impact, and complex correlation from knobs to coverage bins, this is a very tedious and arduous process. This activity can sometimes yield undesirable results, meaning that the targeted bins are not hit even with multiple targeted regressions. This process can often take multiple weeks to fully converge on the coverage and accounts to 10%-20% of entire verification cycle.

III. Proposed Model

A Reinforcement learning (RL) agent that uses a Deep Neural Network (DNN) is incorporated into existing DV infrastructure (highlighted in green in Fig. 1). Reinforcement learning [5, 6] is a specialized category of machine learning algorithms, which aims to maximize the reward by continuously modelling and optimizing the mathematical relationship between inputs and the reward. RL algorithms can be leveraged in the current DV flow to accurately model the relationship between input knobs that are used to generate stimuli to maximize the reward, which is the coverage score. A framework is developed to integrate RL agent into existing testbench coverage flow independent of design-under-verification (DUV), building a feedback loop from coverage database to the input knobs via RL agent. RL agent will use coverage reports to find correlations and relationships from input knobs to the reward as well as correlations between input knobs. Then, RL agent aims to produce an optimized new set of knobs that can maximize

the coverage score. There are four key requirements to accomplish this, viz: stimulus controlling knobs, a block-box model of DUV, feedback path from coverage database to RL agent, and an RL Agent.

The RL Agent aims to produce an optimal value for each of the knobs, which collectively can maximize the coverage score. So, it is essential that the TB's random sequence generation is controllable by external knobs. There are several concepts such as SystemVerilog parameters, command line $test$plusargs or UVM command line processor that can be leveraged for controlling the stimulus generation either from command line or by modifying the global parameter file. An example of SV distribution/weighted constraint and parameters is shown in Fig. 2 to illustrate how knobs can control stimulus. Here SV parameters '*MODE_X_EN,'* '*MODE_Y_EN'* and '*MODE_Z_EN'* can control the random variable '*mode.'* If *MODE_X* targeted for coverage, '*MODE_Y_EN'* and '*MODE_Z_EN'* can be changed to 0 so that '*mode*' will always take *MODE_X* as desired. After RL agent generates the knob values, an automation is required to update the knob values in the TB and start simulations. In this example, the parameters can be moved to a global parameter file which can be updated automatically. As described in section II, Micron NAND test-bench is developed systematically to have these constraint controlling parameters in a global file which is updated automatically after RL agent generates new knob settings.

```
parameter MODE_X_EN = 2;
parameter MODE_Y_EN = 2;
parameter MODE_Z_EN = 6;

typedef enum {MODE_X, MODE_Y, MODE_Z} mode_type_e;

class foo_test;
  rand mode_type_e mode;

  constraint mode_c {
    mode dist {
      MODE_X := MODE_X_EN,
      MODE_Y := MODE_Y_EN,
      MODE_Z := MODE_Z_EN
    };
  }
 ....
 ....
endclass
```

Figure 2. An example of knobs controlling the stimulus

As TB is changing rapidly throughout the verification cycle, RL agent needs to be trained periodically to accommodate newly added knobs and coverage bins. A typical RL agent is trained in several iterations with each iteration involving hundreds of design simulations. This can extremely delay the training process as design simulations are slow in general. The process of learning correlations from stimulus to coverage bins is resided to the test-bench environment, which is independent from the actual DUV, so it is highly recommended that actual DUV be removed from this process. A black-box model or a bus functional model (BFM) mimicking the behavior of DUV can be used to replace the actual DUV. This would also save costs related to huge computational resources and simulation licenses that actual DUV requires.

Final requirement in the DV environment is to establish a feedback path from coverage database to the RL agent. Any industry standard simulator can generate coverage reports per simulation in text and HTML formats. An automation is required to be developed to parse the coverage reports, extract the coverage scores and pass that information back to RL agent in a tabular format. Before sending the coverage data to the RL agent, extracted coverage scores are mapped to the knob settings with which they are associated. An example table of knobs and coverage scores that is fed back to the RL agent is shown in Fig. 3.

Knob A	Knob B	Knob C	Knob D	Knob E	Knob F	Coverage Group 1 Score	Coverage Group 2 Score	Coverage Group 3 Score	Coverage Group 4 Score	Total Average Coverage Group Score
16	19	14	1	2	3	23.77	28.63	43.99	37.34	33.43
16	19	15	1	2	3	40.64	48.22	25.9	50.03	41.20
16	19	14	2	1	2	26.12	54.81	27.71	44.04	38.17
16	19	14	2	2	3	41.15	51.17	45.28	56.37	48.49
16	16	15	1	1	2	41.86	51.88	36	41.1	42.71
16	19	15	2	1	2	31.28	51.16	45.85	41.56	42.46
15	16	15	2	1	3	38.07	54.59	47.54	51.7	47.98
16	19	15	1	1	3	41.09	47.48	25.11	49.34	40.76
16	16	14	2	1	2	16.7	45.82	44.86	50.97	39.59
15	19	15	2	1	2	39.11	51	44.38	50.98	46.37
16	16	15	1	2	3	33.75	61.58	41.22	47.73	46.07
15	19	14	2	1	3	34.57	44.95	47.99	40.57	42.02
15	19	14	2	1	3	32.59	52.5	45.16	51.62	45.47
15	19	15	2	1	2	32.63	56.46	46.56	50.53	46.55
16	16	14	2	1	3	33.08	55.15	40.16	41.59	42.50
16	19	15	1	1	3	30.4	50.89	39.27	49.16	42.43

Figure 3. An example table of knobs and coverage scores

Reward to the RL agent is customized to be either the absolute coverage score, or the relative coverage score with respect to the already hit coverage, using knobs as input. RL agent can be more effective when reward is relative coverage since it aims to increase the newly hit bin count. While developing cover points, they are grouped into cover groups based on their relevance for each feature, operation, or algorithm. Independent RL agents can be trained parallelly to improve each cover group by changing the reward from total average coverage score to a particular cover group score. These feature specific trained models can be extremely important to reach newly introduced feature coverage goals to ensure highest design quality.

To minimize the risk of potential over-reliance on artificial intelligence in the proposed DV model, reward should be customized and controlled very carefully by DV engineers. Additionally, DV engineers should review training knob settings and coverage scores before using them on real DUV. Authors think that the chances of complete failure of artificial intelligence is near zero with careful human involvement at each necessary step in the process. In the worst case of complete failure of this approach, RL agent would have at least produced unique combinations of knob settings (than traditional manual settings) that may uncover corner design bugs with less or no coverage improvement.

IV. Reinforment Learning Agent

The RL agent used for this work is developed by VerifAI Inc. Reinforcement Learning is a technique in which an agent takes action in an environment. The environment returns a reward and a representation of the state of the environment to the agent. The agent takes the next action in the environment based on the reward and the state of the environment. In applying reinforcement learning to the design verification problem, the environment is DV environment, and the reward is typically coverage. The RL agent can be a deep neural network (DNN) that learns to take the best actions based on the reward. RL agent used for this work utilizes advanced RL and robust optimization algorithms, such as ‘gradient ascent approach,’ and dist-perturbation technique. It also employs sophisticated neural network architectures including wide networks with batch normalization and Leaky ReLU (A Leaky Rectified Linear Unit) networks.

The RL agent is given an optimization problem where we have a set of input features X (input knobs), which may constrain relations between them if they represent relative weights. The input knobs can be continuous within a range or have an enumerated type, i.e., have values from a list of possible values. RL agent is configured with an output/label column representing the objective metric we want to optimize. To use reinforcement terminology, we can call this the ‘reward.’ RL agent first fits a DNN that has enough capacity to capture the relationship between input knobs and the output metric. Then it uses gradient ascent on the output of the DNN (Y) with respect to the input features (X) to find the best knob settings that maximize the output of the DNN subject to the constraints defined by the penalty terms. A sample gradient ascent function is plotted in Fig. 4.

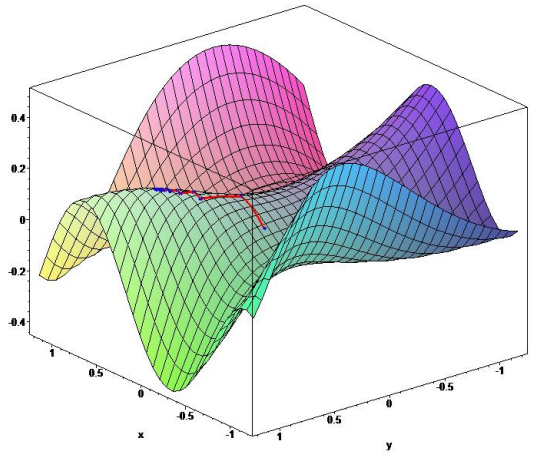

Figure 4. A sample gradient ascent surface

The fundamental idea behind gradient ascent is to update the parameters or variables iteratively in the direction of the steepest climb of the function. The update rule for the parameters at each iteration can be represented as (1) where '**θ'** represents the parameters of the function '**f**' that we want to maximize, '**η'** is the learning rate, a positive scalar determining the size of the step to take in the direction of the gradient. **∇f(θ)** is the gradient of the function 'f' at '**θ**,' which points in the direction of the steepest ascent.

$$\theta_{new} = \theta_{old} + \eta \nabla f(\theta) \tag{1}$$

Gradient ascent process starts with an initial set of parameters θ(0), later gradient ∇f(θ(t)) is evaluated at the current parameter values. In the next step, parameters are adjusted in the direction of the gradient to move toward the maxima as shown in (2). This process is repeated until the changes are sufficiently small or a maximum number of iterations is reached. This can also include conditions like the gradient being close to zero, indicating a potential maximum. This process is illustrated in Fig. 5.

$$\theta(t+1) = \theta(t) + \eta \nabla f(\theta(t)) \tag{2}$$

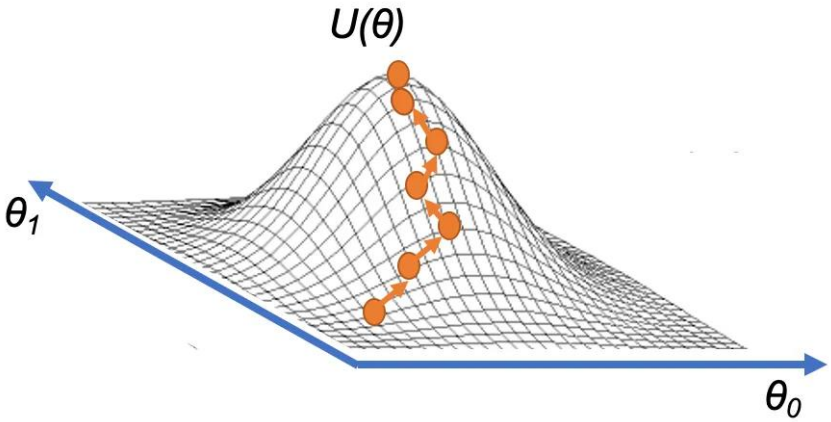

Figure 5. Ascending a function along the gradient

The loss function used in the gradient ascent technique as penalty can be expressed as (3) where '**f(X)**' is the function being maximized (the model's prediction), '**P(X)**' is the penalty function, which computes the penalty based on how far '**X**' deviates from the desired constraints. '**λ**' is the penalty weight, a scaling factor that determines the influence of the penalty term relative to the predictive term.

$$L(X) = -f(X) + \lambda \cdot P(X) \tag{3}$$

The negative sign before f(X) is being maximized, the loss L used in gradient calculations is technically being minimized (a common approach in optimization algorithms). The penalty function P(X) could be structured in many ways depending on the constraints. For example, if we want the sum of certain variables in X to equal 1 (a common constraint in probability distributions or normalized settings), P(X) might be defined as (4) where P(X) penalizes any deviation of the sum of components of X from 1, squaring the difference to ensure the penalty is always positive and more pronounced for larger deviations.

$$P(X) = \sum_{i=1}^{n} (X_i - 1)^2 \tag{4}$$

Overall process of the RL agent starts with a random plausible setting for input knobs X. Later, it iteratively adjusts the settings using gradients of the DNN output Y, with respect to the input, which includes penalties for constraint violation. RL agent attempts to maximize the network's output by adjusting the knob based on the computed gradients. Optimal settings and the corresponding network output after the specified number of iterations.

V. RL Agent Training

Before the training is started, RL agent is configured with testbench knobs, type, and range of each knob. This RL based model is trained in 'n' iterations. Each iteration begins with model generating 'r' unique knob sets. With each knob set, 's' number of NAND simulations are run on the NAND model, and coverage is accumulated per knob set. Then, each knob set is ranked based on how diverse the stimulus is, .i.e., how many unreachable bins that it can hit. This ranking is completely customized based on the end goal which is coverage closure in this case, and this is what will be fed back to agent as reward. In each iteration, RL agent builds and optimizes neural networks to understand and model the relationship between input knobs and output reward. The agent then generates next knob sets which aim to further improve the reward. After all the iterations are finished, last iteration is not necessarily the one with highest coverage due to RL agent being not fully trained, or RL agent is learning negative correlations. So, best knob sets that have highest coverage among the 'n' iterations are handpicked to run regressions with actual DUV. This training process is illustrated in Fig. 6.

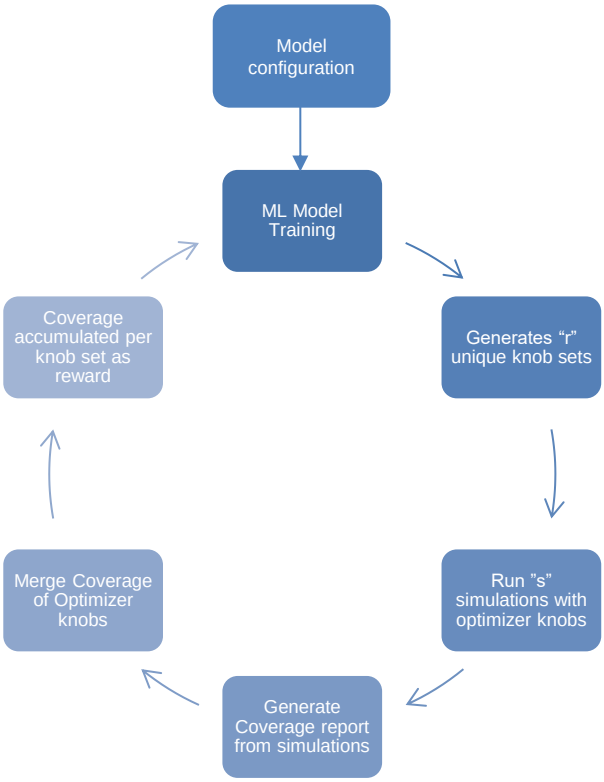

Figure 6. RL Agent training process

VI. Results and Future Work

Fig. 7 shows the coverage plot from one of our ongoing proof-of-concept project training iterations. This plot illustrates the improvement that this work has made.

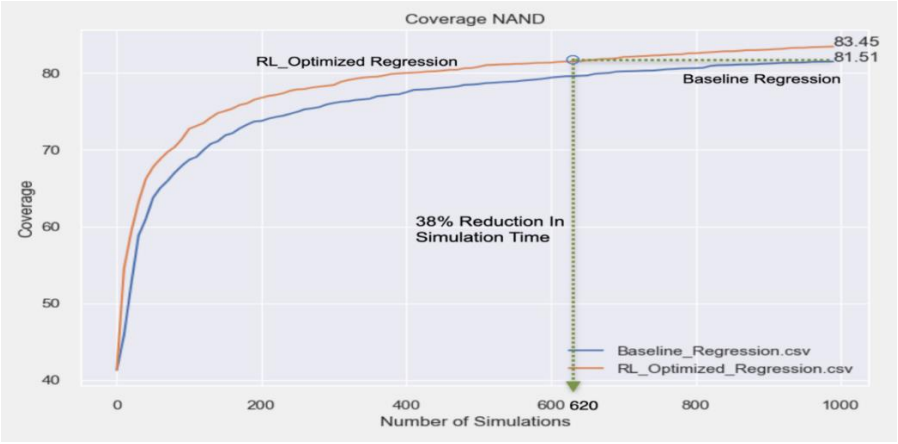

Figure 7. Comparison of Coverage ramp

With the ML methodology, a higher Coverage of **~2%** can be reached with the same number of **1000** simulations, and the same amount of existing coverage can be achieved with **38%** fewer simulations. Given the manual efforts spent, these results are very encouraging in pursuing AI/ML to improve design verification. Additionally, this increased coverage trajectory and reduced simulation time give the verification team more time to further improve verification to find more design bugs. In this work, there is more room for fine-tuning rewards, and finding the correct balance between 'r' – number of unique knobs sets and 's' - number of seeds run per unique knob set to maximize the Coverage. More experiments are being run by adjusting reward metrics, using different combinations of 'r' and 's,' and by using more advanced RL algorithms to reduce training iterations and improve coverage further.

VII. Conclusion

An automated RL agent framework has been developed and incorporated into the existing DV flow. RL agent aims to generate optimal knob settings to steer the random stimulus generation to reach maximum verification coverage. RL agent will be trained on DUV Model or a BFM iteratively throughout the project cycle to learn the effect of each knob on coverage score and on other knobs. Later, trained RL agent is used on real DUV to cover highest number of unit coverage bins in least simulations possible. Results of this work on a proof-of-concept project are promising, and we see an opportunity to deploy it for future NAND products to aid the DV team in improving stimulus and closing coverage.

Efforts will continue to further fine-tune the model, improve the results, and potentially expand its use on verification of future NAND products.

References

[1] K. Kawai, et al., "*A 1Tb Density 3b/Cell 3D-NAND Flash on a 2YY-Tier Technology with a 300MB/s Write Throughputs*," *IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, USA,* February 2024.

[2] M. R. Kulkarni, et al., "*Improving coverage of simulation-based design verification using machine learning techniques,"* 2019, Master's thesis, University of Texas at Austin.

[3] Z. Aref, R. Suvarna, W. Hughes, S. Srinivasan, and N. Mandayam, "*Advanced Reinforcement Algorithms to optimize Design Verification*," *IEEE Design Automation Conference (DAC), San Francisco, USA,* June 2024, pp. 1-6.

[4] W. Hughes, S. Srinivasan, R. Suvarna, and M. R. Kulkarni, "*Optimizing Design Verification using Machine Learning: Doing better than Random*," unpublished.

[5] R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An introduction*, 2018, MIT press.

[6] D. Silver, et al., "*Deterministic Policy Gradient Algorithms*," *International conference on Machine Learning (ICML), Beijing, China,* June 2014, pp. 387-395.