Advanced Reinforcement Learning Algorithms to Optimize Design Verification

Zahra Aref
Dept. of ECE, Rutgers University
New Jersey, USA
zahra.aref@rutgers.edu

Rohit Suvarna
VerifAI Inc.
California, USA
rohit@verifai.ai

Bill Hughes
VerifAI Inc.
California, USA
bill@verifai.ai

Sandeep Srinivasan
VerifAI Inc.
California, USA
sandeep@verifai.ai

Narayan B. Mandayam
WINLAB, Rutgers University
New Jersey, USA
narayan@winlab.rutgers.edu

ABSTRACT

Given the increasing complexity of integrated circuits, the utilization of machine learning in simulation-based hardware design verification (DV) has become crucial to ensure comprehensive coverage of hard-to-hit states. Our paper proposes a deep deterministic policy gradient (DDPG) algorithm combined with prioritized experience replay (PER) to determine the stimulus settings that result in the highest average FIFO depth in a modified exclusive shared invalid (MESI) cache controller architecture. This architecture includes four FIFOs, each corresponding to a distinct CPU. Through extensive experimentation, DDPG coupled with PER (DDPG-PER) proves to be more effective than DDPG with uniform experience replay in enhancing average FIFO depth and coverage within the DV process. Furthermore, our proposed DDPG-PER framework significantly increases the occurrence of higher FIFO depths, thereby addressing the challenges associated with reaching hard-to-hit states in DV. The proposed DDPG-PER and DDPG algorithms also demonstrate a larger average FIFO depth over four CPUs, requiring considerably less execution time than Bayesian Optimization (BO).

KEYWORDS

Deep Reinforcement Learning, Hardware Design Verification, Deep Deterministic Policy Gradient, Prioritized Experience Replay, Functional Coverage, Stimulus Generation

ACM Reference Format:
Zahra Aref, Rohit Suvarna, Bill Hughes, Sandeep Srinivasan, and Narayan B. Mandayam. 2024. Advanced Reinforcement Learning Algorithms to Optimize Design Verification. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3657365

1 INTRODUCTION

The increasing complexity of computer systems, driven by high performance demands and diverse requirements, poses significant challenges for chip verification. Integrated circuits (ICs) and accelerators have gained popularity over traditional multipurpose processors, making chip verification an essential and complex task. In fact, the cost of validating an IC constitutes a significant portion of the overall development cost [12].

Simulation-based verification is a widely used approach for the functional verification of complex hardware designs. It offers scalability and aims to cover a wide range of hard-to-hit states in the design, thereby achieving maximum coverage goals. However, as IC complexity continues to grow, simulation-based functional verification has become a barrier in hardware design verification.

In this paper, we address this challenge by leveraging reinforcement learning (RL) for stimulus or knob settings generation, aiming to achieve optimal verification coverage with a significantly reduced number of simulation cycles. Specifically, we focus on MESI cache controller [13], a cache coherence architecture for hardware verification. Within the MESI InterSection Controller (ISC), the input First In First Out (FIFO) queues are allocated only in the case of address conflicts in incoming CPU transactions, which are unlikely in random transaction streams with random addresses. A transaction with an address mismatch will also often have a short lifespan in the FIFO since it will typically be written back to memory. As a result, filling the FIFOs becomes a challenging task. However, by maximizing the FIFO depth, previously inactive parts of the design become active, enabling the discovery of hard-to-hit states in the MESI Cache Controller design.

Reinforcement learning, a specialized category of machine learning algorithms, offers a promising approach to identify the optimal set of stimuli required to achieve maximum coverage [2, 6]. In our proposed model, we utilize DDPG [9], which has proven highly effective performance in very large discrete action spaces. We employ the Prioritized Experience Replay (PER) algorithm [15] as part of the experience replay in our DDPG framework to increase the average FIFO depth across four CPUs and thus maximizing the coverage of hard-to-hit hardware verification states.

Contributions made by this work include:

- We propose the first known application of deep RL for stimulus generation in hardware design verification using MESI ISC architecture as the Design Under Test (DUT).
- We encode memory address ranges into a set of blocks that provide finite discrete actions for RL from a large continuous action space.

- We apply RL to a multi-arm bandit scheme by defining a single-state Markov decision process.
- We propose a DDPG algorithm with the incorporation of prioritized experience replay (DDPG-PER), resulting in enhanced hardware design coverage and a higher occurrence of deep FIFOs compared to DDPG with uniform experience replay.
- We demonstrate that the proposed DDPG-PER and DDPG algorithms achieve a larger average FIFO depth across four CPUs, with a considerably smaller execution time than BO model.

The rest of this paper has the following structure. A discussion of related work is provided in section 2, while a presentation of simulation-based design verification is provided in section 3. The proposed model is described in detail in section 4. Experimental results are provided in section 5, followed by the conclusion and future work in section 6.

2 RELATED WORK

Recent years have witnessed an increase in the use of machine learning to improve functional verification [3, 4, 8, 19]. The authors in [3] implemented a long short term memory algorithm and a random forest classifier in a quad-core MESI cache architecture. They examined coarse- and fine-grained machine learning model training in stimulus optimization through test-level and transaction-level experiments, respectively. It was demonstrated that in some common coverage categories, transaction-level optimization could decrease the CPU time required for verification coverage by around 70%, whereas test-level optimization fails to achieve significant gains. The work in [19] presents a three-layer artificial neural network model for coverage-directed test generation with the aim of increasing the chance of selecting high-coverage test cases during functional verification of a basic CPU DUT. It is demonstrated that simulation time is reduced and assertion coverage is enhanced. The authors argue that deep convolutional neural networks may be ineffective because of the small number of stimuli required. Clustering is used in [4] to condense a vast pool of tests into those that would most likely lead to higher coverage of an ALU DUT. Filtering allows for the efficient removal of many duplicate tests while ensuring optimum final coverage. The results indicate a four-fold increase in speed. Machine learning algorithms including support vector machine and Sigmoid activated neural network have been proposed in [6, 8] to find hard-to-hit states in the verification of complex integrated circuits using input stimuli that perform significantly better than constrained random inputs. The model was developed based on the MESI cache controller and the RISCV-Ariane core architectures.

Reinforcement learning has been proven effective in functional hardware verification, with the aim of detecting hard-to-hit states, and achieving high coverage goals [1, 14]. According to [14], a mechanism for automating shared memory verification is presented that combines RL with a recurrent neural network. Applying a near-miss tracking approach improves the capability of reaching hard-to-hit functional states and uncovering bugs. The authors assert that the tests created through RL have greater coverage values than those provided by genetic algorithms. Deep RL is utilized in [1] to generate test vectors for dynamic memory device verification using a K-NN policy gradient. The authors demonstrate that they can accomplish greater levels of coverage in DRAM by comparing their proposed framework to tests provided by engineers.

3 SIMULATION-BASED DESIGN VERIFICATION

Design verification, or functional verification, is a technique used in hardware design to demonstrate a design's correct functionality. Despite being one of the most expensive processes, it usually receives the most focus for efficiency improvement. In functional verification, the main objective is to ensure that the design implementation meets the specifications without containing any bugs. Specifically, when the hardware description language (HDL) fails to represent the required functionality or when the hardware designer fails to find hard-to-hit states, it is the responsibility of the hardware verification engineer to identify those bugs.

Functional verification is primarily performed by formal and simulation-based techniques. Formal verification uses math to prove or disprove the correctness of a design specification by exhaustively exploring the state space. The state space also increases exponentially as the number of flip-flops in the design increases. On the other hand, simulation-based verification continues to be widely applied in industrial manufacturing due to its scalability and ease of use. However, it requires a considerable amount of time, labor, and a large number of test vectors, which is still insufficient to guarantee the correct behavior of deep states of the design. In simulation-based verification, input vectors, or knob settings, are generated first, and then the outputs are determined according to the input vectors [2].

Coverage directed test generation is the primary method for modern verification. A large number of test cases are generated using random stimulus generators and coverage is used as the primary metric, allowing the verification team to determine whether all interesting scenarios have been tested. It requires an in-depth understanding of the DUT as well as expertise in the directives provided to the test generator. For each coverage scenario, the constraint-driven engine provides a set of directives to the test generator that maximize the probability of hitting those states, for each coverage event. [10].

RL can be used as an effective method to generate precise stimuli and thus improve design coverage. A set of constraints would serve as features, and information on how much a desirable condition is reached would serve as the reward to be optimized.

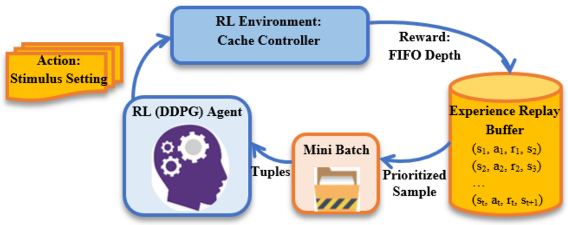


Figure 1: Proposed model.

4 PROPOSED MODEL

In this paper, we introduce a stimulus generation optimization model that utilizes the deep deterministic policy gradient RL algorithm combined with prioritized experience replay within a functional verification environment. Our study focuses on the design under test, which is an open-source Verilog RTL for a multiprocessor MESI ISC. The proposed model is illustrated in Fig. 1.

4.1 DUT-Cache controller design

MESI cache controllers play an important role in ensuring the consistency of local and memory-cached data. The MESI ISC design is based on the MESI protocol [13] which maintains cache coherence in systems with multiple cores and local caches using a write-back policy. Fig. 2 illustrates a MESI ISC with a quad-core coherency configuration, where each core represents a processor. Nowadays, a quad-core configuration is considered commonplace. In this setup, each CPU is equipped with its own FIFO queue, capable of holding a maximum of 16 transactions. Incoming CPU transactions pass through an arbitration unit before being directed to the respective caches. This arbitration unit is responsible for determining which accesses can proceed to the cache and which ones need to be temporarily stalled for later processing. To resolve conflicts between load and store operations, four FIFOs, associated with the four CPU ports, are employed. When two CPUs attempt to access the same address simultaneously, the CPU with higher priority is granted access, while the CPU with lower priority inserts the corresponding transaction into its FIFO to be handled subsequently. As a result, deep FIFOs are considered as hard-to-hit states since only the address conflicts could lead to an increase in the queue depth. The MESI ISC architecture is chosen as a proof of concept because the input FIFOs are only allocated in the case of an incoming transaction address conflict, which is relatively uncommon in a random transaction stream with random addresses. Additionally, when a transaction is loaded due to an address mismatch, it typically remains in the FIFO for a short duration before being written to memory. Consequently, in the presence of a stream of random input transactions with random addresses, the occupancy of the FIFOs is expected to be very low.

When FIFO queues reach their capacity and become filled, previously inactive components of the design are activated. This enables the detection of bugs that may occur in deep states of the MESI cache controller design. Therefore, maximizing the FIFO depth can be valuable in identifying elusive bugs that are challenging to discover through conventional testing methods. To ensure the correct functionality of the cache controller, a dedicated DV environment is proposed for each CPU port. The CPU transactions are appropriately weighted, and the transaction memory addresses are constrained based on the tag, index, and offset address of the target cache.

4.2 Deep Deterministic Policy Gradient

RL involves training an intelligent agent to determine the optimal actions that maximize the cumulative reward when interacting with an environment. The behavior of the agent is modeled as a Markov decision process, where a policy π maps states to probability distributions over actions: $\pi : S \longrightarrow P(A)$. In a fully-observable environment with discrete episodes, the RL model is characterized by a state space S, an action space $A = \mathbb{R}^N$, transition probabilities $p(s_{t+1}|s_t, a_t)$, and a reward function $r_t(s_t, a_t)$ in each episode t.

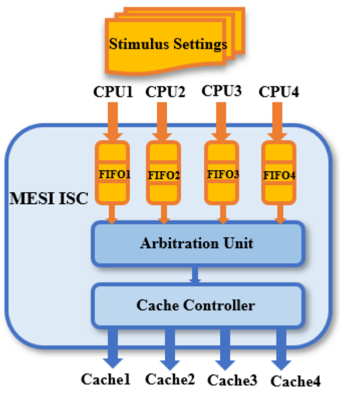


Figure 2: Cache controller design.

To model the reinforcement learning problem, we adopt a multi-arm bandit scheme [18] with a single state, where the state is represented by a constant integer value $s_t = 1$, with an initial state of $s_0 = 0$.

The action vector a_t denotes the DV stimulus setting in episode t which has a size of 32, representing eight knobs per CPU. There are five knobs corresponding to instruction control and three knobs associated with memory address control settings for each CPU. The full list of DV stimulus settings presented in our design can be found below.

- The instruction control indicates how often each type of instruction is executed by each CPU:
  - Percentage of Read
  - Percentage of Write
  - Percentage of RdBroadcast
  - Percentage of WrBroadcast
  - Percentage of No Operation
- Memory address control breaks down the memory address for each CPU into Tag, Index, and Offset fields. In each case, a range is specified as follows:
  - Tag field specified by memory address bits $16 : 31$ is forced to a maximum range as:$\{0 : 0x0001,\ 1 : 0x0002,\ 2 : 0x0004,\ 3 : 0x0008,\ 4 : 0x000F,\ 5 : 0x00FF,\ 6 : 0x0FFF,\ 7 : 0xFFFF\}$.
  - Index field represented by address bits $4 : 15$ is forced to a maximum range as:$\{0 : 0x000,\ 1 : 0x001,\ 2 : 0x002,\ 3 : 0x004,\ 4 : 0x008,\ 5 : 0x00F,\ 6 : 0x0FF,\ 7 : 0xFFF\}$.
  - Offset field defined by address bits $0 : 3$ is forced to a maximum range as:$\{0 : 0x1,\ 1 : 0x2,\ 2 : 0x4,\ 3 : 0x8,\ 4 : 0xA,\ 5 : 0xB,\ 6 : 0xC,\ 7 : 0xF\}$.

The action vector a_t is defined as follows:

$$a_t = \left\{ [\rho_{ij}, \sigma_{ik}]_{1 \le i \le 4, 1 \le j \le 5, 1 \le k \le 3} \middle| 0 \le \rho_{ij} \le 100, 0 \le \sigma_{ik} \le 7 \right\} \tag{1}$$

where, ρ_{ij} and σ_{ik} denote the instruction control and memory address control setting of CPU i, respectively. A CPU's instruction control is specified by an integer in the range $[0, 100]$, which represents the percentage of the executed commands. Furthermore, each field in the memory address control settings is defined as an integer within the range of $[0, 7]$, which indicates the memory address range.

In our proposed model, the reward r_t is defined as the average FIFO depth across four CPUs in episode t. MESI Cache Controller has four FIFO queues, one for each CPU. Up to 16 entries can be stored in each FIFO queue. The aim is to maximize the number of entries in each FIFO, so that the deeper the average FIFO across all four queues, the greater the likelihood of finding hard bugs.

Our proposed framework introduces a discrete action space by dividing the memory addresses of tag, index, and offset into blocks. As a result of the definition of a discrete action space, the large continuous action space can be replaced with a smaller discrete one.

The objective is to identify a policy that maximizes the return, which is defined as the sum of discounted future rewards: $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_t(s_i, a_i)$, with $\gamma \in [0, 1]$ as a discounting factor. The action-value function in reinforcement learning algorithms is defined by the Bellman equation [18]:

$$Q^*(s_t, a_t) = \mathbb{E}[r_t(s_t, a_t) + \gamma \max_{a'_{t+1}} Q^*(s_{t+1}, a'_{t+1})] \tag{2}$$

Directly applying Q-learning is not feasible in large action spaces, as finding a greedy policy requires optimizing a_t in each episode in a vast space, which is computationally slow for practical implementation. To address this, we employ DDPG algorithm [9]. DDPG is a powerful RL algorithm specifically designed for learning in high-dimensional or continuous action spaces. It combines deep neural networks and the actor-critic model, building upon the deterministic policy gradient [16].

The action-value network $Q(s_t, a_t; w)$ and the actor network $\mu(s_t; v)$ are deep neural networks used to approximate the action-value function $Q(s_t, a_t)$ and the actor function $\mu(s_t)$, respectively. The parameters w and v represent the network parameters, and $\mu(s_t)$ maps a state s_t to a deterministic action a_t. Deep reinforcement learning algorithms combine traditional reinforcement learning algorithms, such as Q-learning, with deep neural networks. Based on the DPG policy gradient algorithm [16], the actor network parameters are updated using the following equation:

$$\begin{aligned} &\nabla_v Q(s, a; w)|_{s=s_t, a=\mu(s_t; v)} \\ &= \nabla_a Q(s, a; w)||_{s=s_t, a=\mu(s_t; v)} \nabla_v \mu(s; v)|s = s_t \end{aligned} \tag{3}$$

In deep RL, neural networks can pose challenges due to the assumption of independent and identical samples made by optimization algorithms. However, this assumption is violated when samples are generated sequentially in an environment. The DQN algorithm addresses this issue by using a replay buffer, also known as the experience replay buffer. Experience tuples (s_t, a_t, r_t, s_{t+1}) are gradually added to the replay buffer as the agent interacts with the environment. A common approach is to use a fixed-size buffer, where new data replaces the oldest experience at the end of the buffer. In each episode, a minibatch is sampled uniformly from the replay buffer, allowing DDPG to learn from uncorrelated transitions.

Through continuous interaction with the environment, the actor and critic networks tend to converge, leading to optimal stimulus settings that maximize the average FIFO depth.

4.3 Prioritized Experience Replay

The prioritized experience replay algorithm is designed to prioritize experience tuples in RL, allowing for more frequent replay of significant transitions and facilitating better learning [15]. The underlying concept of prioritized experience replay is to replay experiences related to highly effective or poor performance more frequently. To determine the criterion for replaying experiences, a ranking based on temporal-difference (TD) errors can be chosen. TD errors reflect the unexpectedness of a transition or how much the value deviates from the expected value in the next step. Experiences with large TD error magnitudes are assumed to be advantageous and their expected action values are forcefully modified. Conversely, experiences with large negative TD error magnitudes indicate poor behavior by the agent and the need to learn the underlying states. By replaying these experiences more frequently, the agent can learn more effectively. The absolute TD error, δ_t^i, is used as an index to measure the value of experiences and is defined as follows:

$$\delta_t^i = r_t(s_t, a_t) + \gamma \max_{a'_{t+1}} Q(s_{t+1}, a'_{t+1}, w) - Q(s_t, a_t, w) \tag{4}$$

To determine the likelihood $P(i)$ of sample experience i, a probability distribution is created using the priorities of the transitions as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{5}$$

where $p_i = |\delta_t^i| + \epsilon$ represents the priority of transition i, and ϵ is a small positive constant. The value of α determines the priority given to each experience, with $\alpha = 0$ indicating uniform prioritization. However, prioritized replay introduces bias into the distribution of sampled tuples, which can be detrimental to neural network training. This bias is eliminated by using importance-sampling (IS) weights. The final importance-sampling weights w_i are calculated as follows:

$$w_i = \left(\frac{1}{NP(i)}\right)^\beta \tag{6}$$

where β determines the compensation for $P(i)$ and N is the size of the minibatch. These weights, multiplied by δ_i, are incorporated into the Q-learning update. To ensure stability, the weights are always normalized to $1/\max_i w_i$.

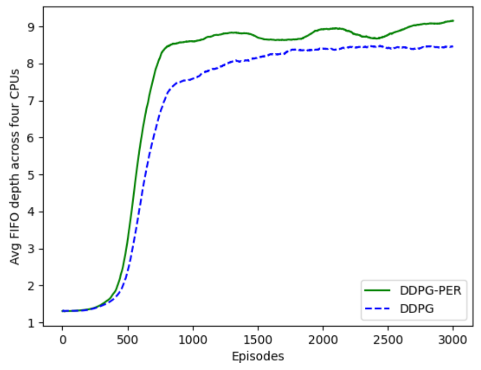
4.4 Bayesian Optimization

As our problem, determining the stimulus settings that maximize the average FIFO depths in MESI ISC architecture, can be viewed as the optimization of a black-box function, we employ the Bayesian Optimization (BO) [11] method as a baseline. It has been demonstrated that BO is a highly effective global optimization algorithm of black-box functions [7]. In BO, the objective is to solve the global optimization problem as follows:

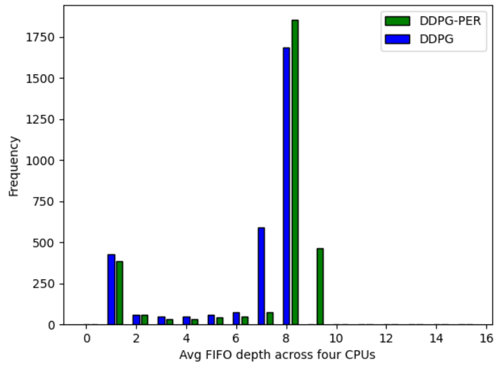
$$x_{max} = \operatorname*{argmax}_{x \in \mathcal{X} \subseteq \mathbb{R}^d} f(x) \tag{7}$$

where, x_{max} denotes the DV stimulus settings of size 32, corresponding to eight knobs per CPU, that maximizes $f(x)$ in our problem. The $f(x)$ denotes the average FIFO depth across four CPUs. Assuming $f(x)$ is a continuous unknown function, a noisy observation in episode t is defined by $y_t = f(x_t) + \epsilon_t$.

It is assumed that the function $f(x)$ is drawn from a Gaussian process (GP) prior and the observations are of the form $\{x_t, y_t\}_{t=1}^N$,



(a)



(b)

**Figure 3: a) Average FIFO depth.
b) Frequency of average FIFO depth.**

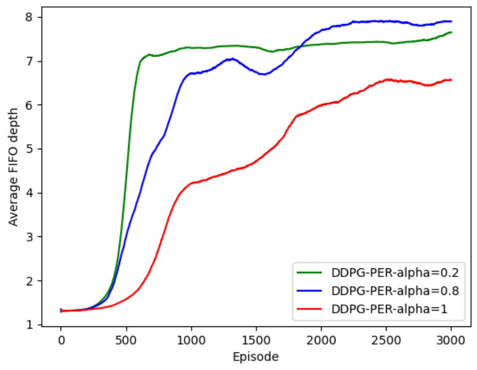


Figure 4: Effect of α on average FIFO depth in DDPG-PER.

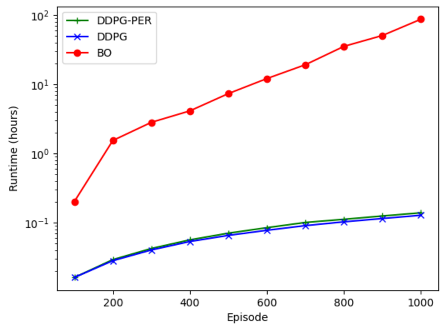


Figure 5: Runtime in hours vs. episodes.

where $y_t \sim \mathcal{N}(f(x_t), \sigma^2)$ with variance of noise σ^2 in the observations. The posterior distribution, specified by $p(f|\{x_t, y_t\}_{t=1}^N) \propto p(\{x_t, y_t\}_{t=1}^N|f)p(f)$, represents the updated beliefs on the unknown objective function f. It is derived by multiplying the prior distribution $p(f)$ with the likelihood function $p(\{x_t, y_t\}_{t=1}^N|f)$ which defines the likelihood of the observed data given f, assuming Gaussian noise on the observations. The posterior distribution, is considered as the acquisition function for determining the next optimal point in the space $\mathcal{X}$. As the acquisition function, we utilize GP-Hedge [5], which includes N bandits, each associated with an acquisition function. GP-Hedge has demonstrated superior performance compared to using a single acquisition function. In GP-Hedge, one of the lower confidence bound (LCB), expected improvement (EI), and probability of improvement (PI) acquisition functions is selected by a softmax probability. Then, the reward r_t^i of acquisition function i for x_t^i is calculated by the expected value of the GP model, i. e., $r_t^i = \mu_t(x_t^i)$ in episode t.

5 EXPERIMENTAL RESULTS

Our experiments were carried out on an AWS instance with an Intel Xeon E5-2686 v4 (4 cores, 2.30GHz, 16 GB RAM) running Amazon Linux 2. The system had 200 GB storage with 30 GB free. Experiments utilized CPU resources, as no GPU acceleration was available. Furthermore, in order to measure the associated FIFO depths of four CPUs using the stimulus vectors generated by our RL-based architecture, we utilized the open-source Verilator simulator [17]. The deep deterministic policy gradient algorithm combined with prioritized experience replay was implemented to maximize the average FIFO depth across the four CPUs and increase the occurrence frequency of deeper FIFOs.

To evaluate the performance of our approach, we conducted experiments with two variants: DDPG with uniform experience replay (DDPG) and DDPG with prioritized experience replay (DDPG-PER). We performed a total of 3000 episodes to maximize the cumulative reward, which represents the average FIFO occupancy across the four CPUs obtained by optimal stimulus settings from DDPG and DDPG-PER. As part of the training process, DDPG was trained on a dataset of prior random test cases. In our simulations, we set the parameters of prioritized experience replay as $\alpha = 0.5$ and $\beta = 0.5$, and the minibatch size was set to 128. Additionally, the FIFO depth was assumed to be an integer within the range of 1 to 16. The figures presented in this section show the average of the previous 100 FIFO depths.

Fig. 3a illustrates the average FIFO depth achieved by our proposed model. It clearly demonstrates that DDPG-PER achieves a higher reward in fewer episodes compared to DDPG with uniform replay. As a point of clarification, we collected the average of 100 previous rewards during the code execution. Fig. 3b presents the distribution of average FIFO depth across the four CPUs, derived from the stimulus settings generated by DDPG and DDPG-PER. The

Metric / Algorithm	DDPG-PER	DDPG	BO
Avg Runtime (hours)	0.047	0.045	86.7
Best Avg FIFO Depth	10.27	10.26	9.82

Table 1: Evaluated metrics in 1000 episodes.

histogram depicts the increasing frequency of higher FIFO depths, indicating the effectiveness of our approach. DDPG-PER performs better in terms of reaching larger FIFOs more frequently compared to DDPG with uniform experience replay.

In Fig. 4, we investigated the effect of α on the average FIFO depth achieved by DDPG-PER, while keeping β constant. The graph shows the average FIFO depth over the four CPUs for different values of α, representing the priority given to the transitions. The results indicate that DDPG-PER with $\alpha = 0.8$ achieves a greater average FIFO depth compared to very small values like 0.2 or the maximum value of 1. Interestingly, DDPG-PER with $\alpha = 1$ and $\beta = 0.5$ does not yield the maximum average FIFO depth, suggesting that using only TD-error as the priority criterion might not be sufficient in our environment.

The runtime of our proposed models is compared to the BO algorithm in Fig. 5. We ran BO with 20 initial samples, observation noise of 0.5, and exploration-exploitation balance parameters of $x_i = 0.2$ and $\kappa = 3$. The results demonstrate that DDPG-PER is executed significantly faster than BO. This is due to the fact that BO is more complex. It uses GP model as a surrogate function. The complexity of GP models increases and updating these models becomes more expensive as more data points are collected. The inversion of the covariance matrix results in GP inference scaling cubically with data points. In addition, in order to determine the next point, BO must optimize the acquisition function, GP-Hedge, at each episode. Due to a more complex response surface with more local optima as the number of episodes increases, optimizing this acquisition function may become more challenging. It is also observed that DDPG-PER runs slightly longer than DDPG as prioritizing the tuples affects the runtime.

Table 1 represents two metrics evaluated in 1000 episodes. In comparison with DDPG-PER and DDPG, BO is executed at a significantly slower speed. Additionally, DDPG-PER and DDPG yield greater average FIFO depth over four CPUs than BO, presumably because BO may be less efficient on very high-dimensional inputs.

6 CONCLUSION AND FUTURE WORK

We presented a novel application of reinforcement learning for stimulus generation in design verification using the MESI cache controller topology. To handle the challenge of a large continuous action space, we encoded the memory address ranges into specific blocks, enabling the use of RL algorithms in a finite discrete RL action space. By utilizing a multi-arm bandit configuration and specifying constant states of the Markov decision process, we successfully applied RL algorithms to our design verification framework. Furthermore, we proposed the incorporation of prioritized experience replay in the DDPG design verification process. This enhancement increased the hardware design coverage by effectively reaching more hard-to-hit states and increasing the frequency of deeper FIFOs compared to DDPG with uniform experience replay. Our experimental results demonstrate the superior performance of the DDPG-PER algorithm, achieving higher rewards in fewer episodes compared to the DDPG algorithm with uniform replay. Moreover, we demonstrated that DDPG-PER and DDPG achieve greater average FIFO depth over four CPUs than BO, with a substantial improvement in execution speed.

For future work, we plan to extend our RL-based design verification models to the open-source RISC-V Ariane 64-bit architecture. This extension will allow us to evaluate the effectiveness of our approach in reaching hard-to-hit states and further improving hardware coverage in a different architectural context. By exploring the applicability of our models in different scenarios, we aim to validate their effectiveness and potential for broader practical applications in the field of design verification.

REFERENCES

[1] Hyojin Choi, In Huh, Seungju Kim, Jeonghoon Ko, Changwook Jeong, Hyeonsik Son, Kiwon Kwon, Joonwan Chai, Younsik Park, Jaehoon Jeong, et al. 2021. Application of Deep Reinforcement Learning to Dynamic Verification of DRAM Designs. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 523–528.

[2] Alexandru Dinu and Petre Lucian Ogrutan. 2022. Reinforcement Learning Made Affordable for Hardware Verification Engineers. *Micromachines* 13, 11 (2022), 1887.

[3] Saumil Gogri, Aakash Tyagi, Michael Quinn, and Jiang Hu. 2022. Transaction Level Stimulus Optimization in Functional Verification Using Machine Learning Predictors. In *2022 23rd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 71–76.

[4] Onur Guzey, Li-C Wang, Jeremy R Levitt, and Harry Foster. 2009. Increasing the efficiency of simulation-based functional verification through unsupervised support vector analysis. *IEEE transactions on computer-aided design of integrated circuits and systems* 29, 1 (2009), 138–148.

[5] Matthew Hoffman, Eric Brochu, Nando De Freitas, et al. 2011. Portfolio Allocation for Bayesian Optimization.. In *UAI*. 327–336.

[6] William Hughes, Sandeep Srinivasan, Rohit Suvarna, and Maithilee Kulkarni. 2019. Optimizing design verification using machine learning: Doing better than random. *arXiv preprint arXiv:1909.13168* (2019).

[7] Donald R Jones. 2001. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization* 21 (2001), 345–383.

[8] Maithilee Rajendra Kulkarni et al. 2019. *Improving coverage of simulation-based design verification using Machine Learning techniques*. Master's thesis. The University of Texas at Austin.

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[10] Ashok B Mehta. 2018. Constrained random verification (crv). In *ASIC/SoC Functional Design Verification*. Springer, 65–74.

[11] Jonas Mockus. 1998. The application of Bayesian methods for seeking the extremum. *Towards global optimization* 2 (1998), 117.

[12] Andreas Olofsson. 2017. Intelligent design of electronic assets (idea) & posh open source hardware (posh). *Mountain View: DARPA* (2017).

[13] Mark S Papamarcos and Janak H Patel. 1984. A low-overhead coherence solution for multiprocessors with private cache memories. In *Proceedings of the 11th annual international symposium on Computer architecture*. 348–354.

[14] Nícolas Pfeifer, Bruno V Zimpel, Gabriel AG Andrade, and Luiz CV dos Santos. 2020. A reinforcement learning approach to directed test generation for shared memory verification. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 538–543.

[15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[16] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*. PMLR, 387–395.

[17] Wilson Snyder. 2004. Verilator and systemperl. In *North American SystemC Users' Group, Design Automation Conference*.

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

[19] Fanchao Wang, Hanbin Zhu, Pranjay Popli, Yao Xiao, Paul Bodgan, and Shahin Nazarian. 2018. Accelerating coverage directed test generation for functional verification: A neural network-based framework. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. 207–212.