2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

OCTOBER 26-27, 2021

Optimizing Design Verification Using Machine Learning - Doing Better than Random

William Hughes*, Sandeep Srinivasan* , Rohit Suvarna*, Maithilee Kulkarni++

* VerifAI Inc, ++ Xilinx Inc.

accellera SYSTEMS INITIATIVE

Agenda

- Design Verification Challenges
- Code Coverage Example - Doing better than Random
- Deep-Q-Learning for Code Coverage Example
- Deep-Q-Learning for Design Verification
- Test Case #1 - MESI Cache Controller Design
- Test Case #2 - RISCV CVA6 (Ariane) Future Direction
- Future Direction - ML for DV
- Conclusion

Verification Complexity Growing Rapidly

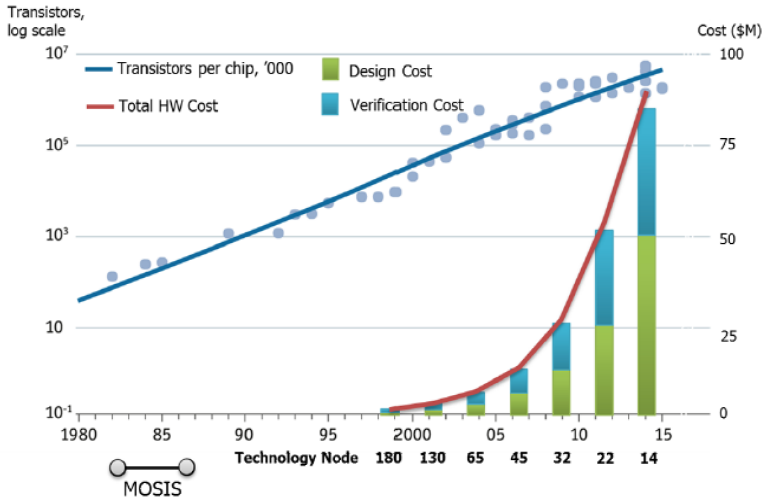


Figure 1. Illustration of the rapid increase in resources required for physical design and verification as Moore's Law has progressed.

****Source:** Page 3 Design Defense Advanced Research Projects Agency

Hardware

- 4 Verification Engineers per Design Engineer*
- Verification costs > 55% of Design Cost & Rising*
- Hard bugs found late in the Project

Software

- 1.5 Verification Engineers per Developer
- Security bugs found late in Production

*Source : Wilson research group study 2020, Siemens EDA

Verification Challenges

Can we do better than Random and humans in lesser time ?

A New Approach is Needed

Code Coverage – Doing better than Random

```
for i in range(num_iterations):
    depth = 0
    i,j,k = 0,0,0
    i = random.randint(1,100)
    if i in range(1,26):
        depth = 1
        j = random.randint(1,100)
        if j in range(26,51):
            depth = 2
            k = random.randint(1,100)
            if k in range(51,76):
                depth = 3
            else:
                depth = 2
                k = 0
        else:
            depth = 1
            j = 0
    else:
        depth = 0
        i = 0
```

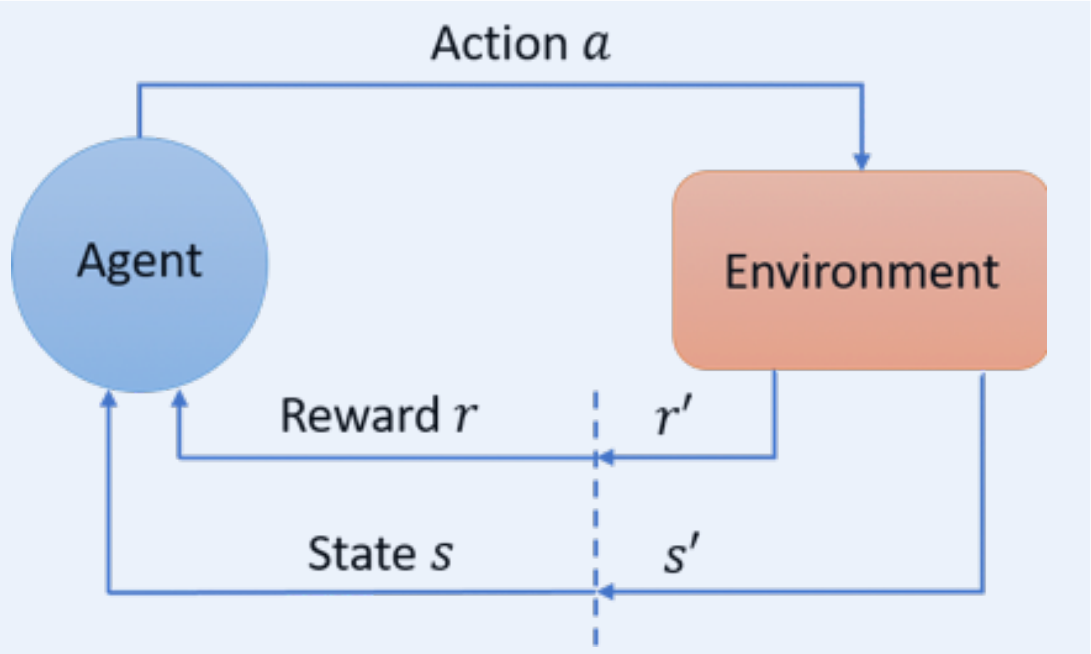
Goal: **Learn** how to reach max code depth consistently
➔ Find best { i , j , k }

- 3 levels in a nested 'if' condition, flows to lower depths only if the values of *{i,j,k}* satisfy a certain range constraint
- Depth =3 is a proxy for a hard to reach condition or state
- Can we train a Neural Network to learn to produce the best *{i,j,k}* values without exhaustive random combinations (fuzzing*) ?
- How does random selection of *{i,j,k}* perform?

Total number of States in this example =	3^8	~= 6561
Total Number of Atoms in the Universe =~	10^{82}	~= 2^{287}
Total number of States in a GO game =	10^{172}	~= 2^{602}
Total Number States in a Small Microprocessor		~= $2^{2\text{million}}$

*The term "fuzz" originates from a fall 1988 class project[2] in the graduate Advanced Operating Systems class (CS736), taught by Prof. Barton Miller at the University of Wisconsin

Code Coverage – Deep-Q-Learning (DQN)



- Q-Learning is a Reinforcement Learning Algorithm (RL)*
- In Q-Learning the goal is to maximize the cumulative discounted reward

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right]$$

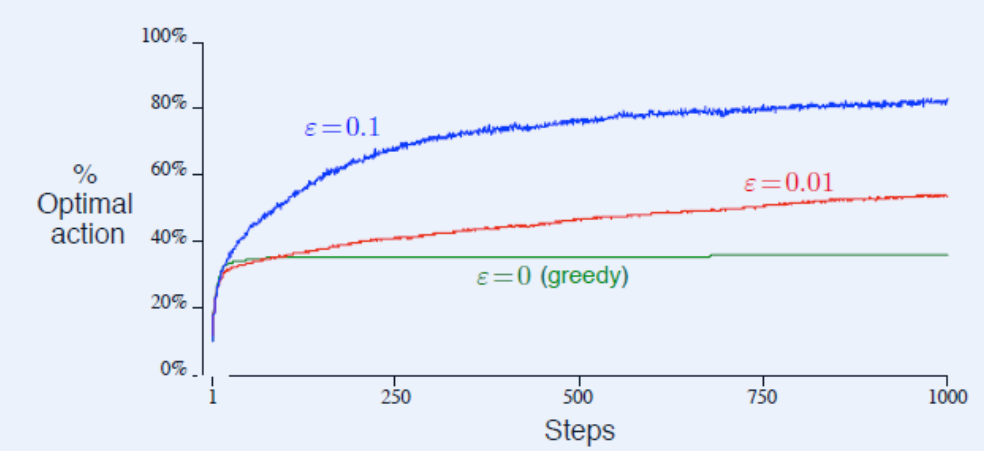
For a given state s, and action a at time t, with the discount factor γ.

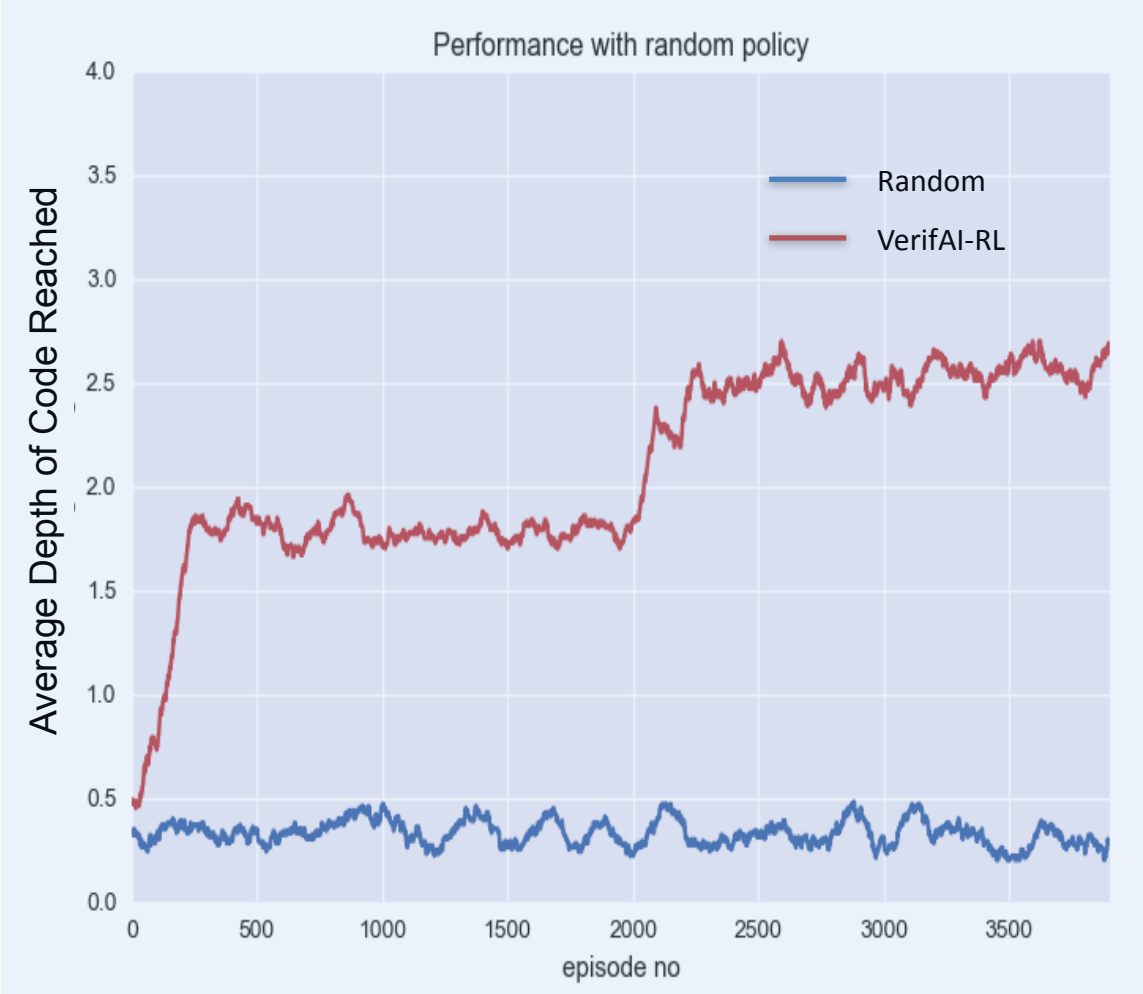
- At the core of DQN are Q-value , where Q represents the discounted future rewards for a particular action from a particular state
- DQN uses Deep Neural Networks (DNN) to estimate these Q-values when the state space becomes too large for simple lookup table approaches

***Reinforcement Learning: An Introduction** Richard S. Sutton and Andrew G. Barto

Code Coverage – DQN Results

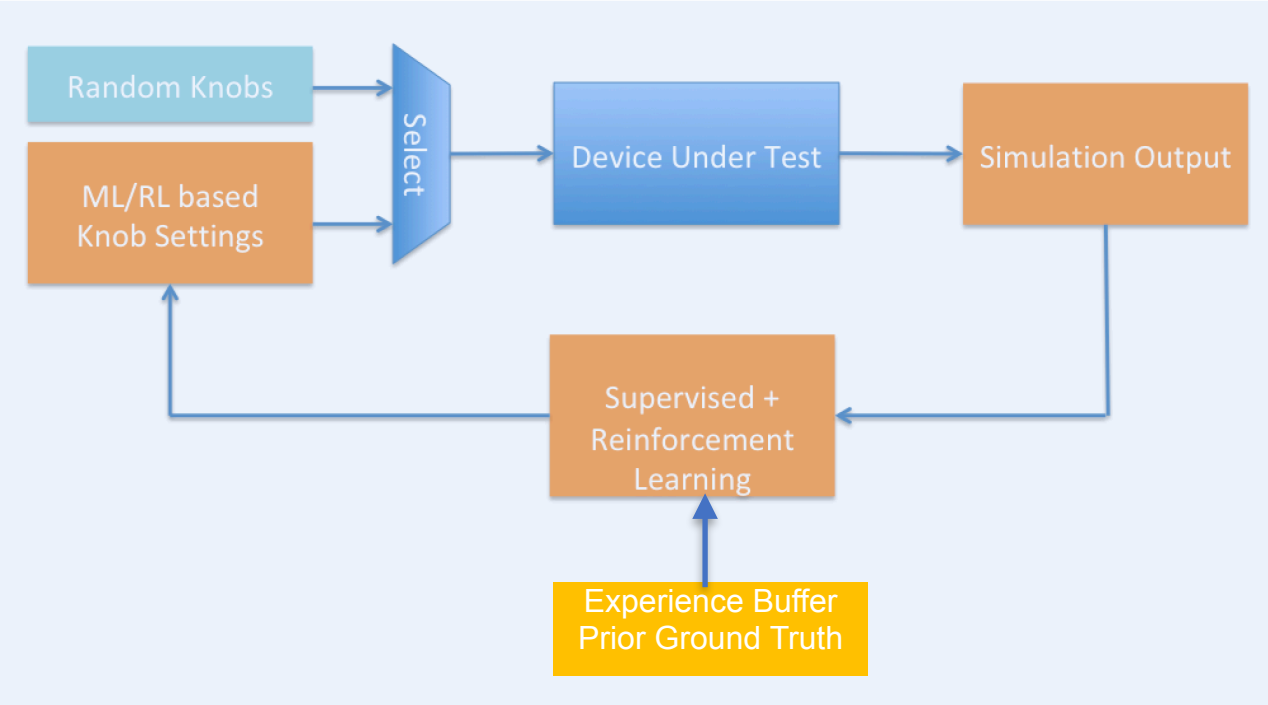
- We applied DQN - The RL agent without any prior knowledge learns by trial and error that *i* should be in *(1,25)*, *j* in *(25,50)* and *k* in *(50,75)*
- The results can be seen on the right with clears points where the agents performance goes to the next level
- Periodic exploration ($\epsilon > 0$) leads to higher cumulative future reward





***Source:** Reinforcement Learning: An Introduction Richard S. Sutton and Andrew G. Barto

DQN for Design Verification



- Use RL to drive the input to a Simulator with a set of controllable knobs
- Optimize a given metric or multiple metrics (Coverage, FIFO Depths, VB Count) that depend on the knobs settings
- Modeled as Markov Decision Process (MDP)
- Use a DQN (Deep Q-Network) to train an RL agent to to explore the action space and learn the Q-values
- Pre-fill experience buffer to speed up learning

Test Case #1 - MESI Cache Controller*

- **MESI Cache Controller- Quadcore CPU design**

- 4-way set associative cache of size 256 kB (one for each CPU)
- 4-bit offset
- 12-bit index
- 16-bit tag

- **Knobs (Input to testbench to generate constrained random sequences)**

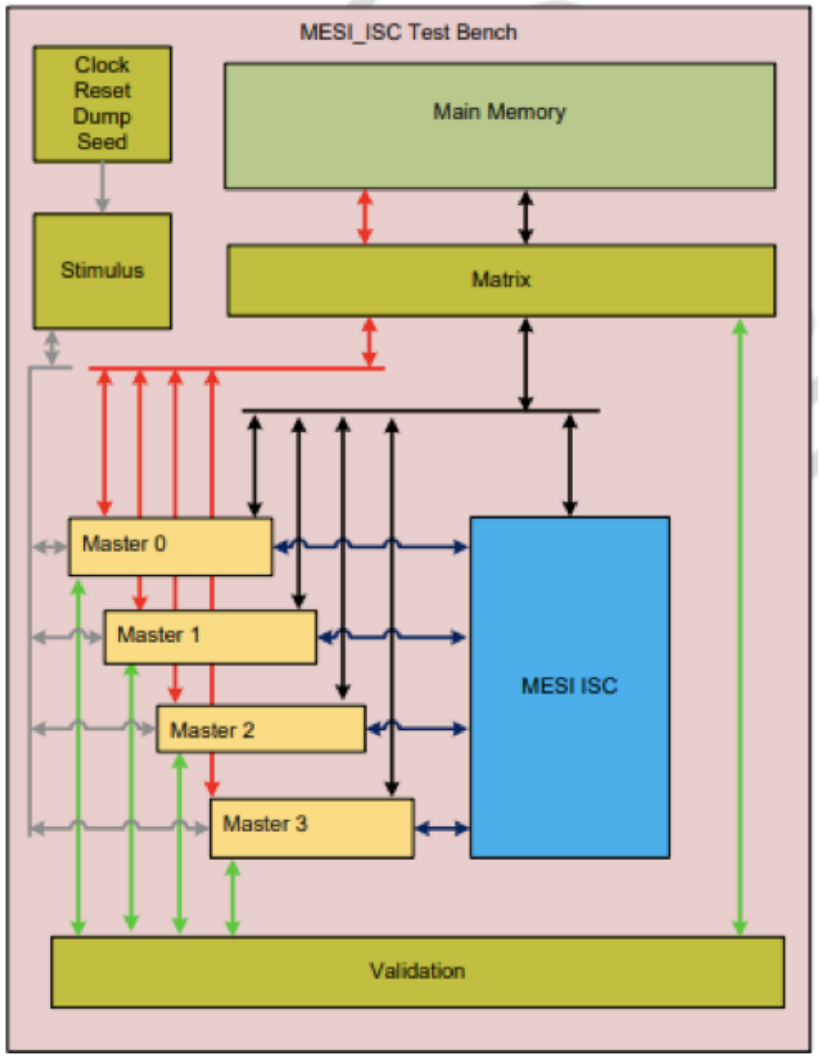
- 16 for distribution of commands (4 for each CPU -(broadcast vs single read/ write)
- 12 for distribution of freedom of address (3 for each CPU - tag/ index. offset)

- **Output captured from testbench**

- FIFO depth for each CPU at each cycle

- **Goal**

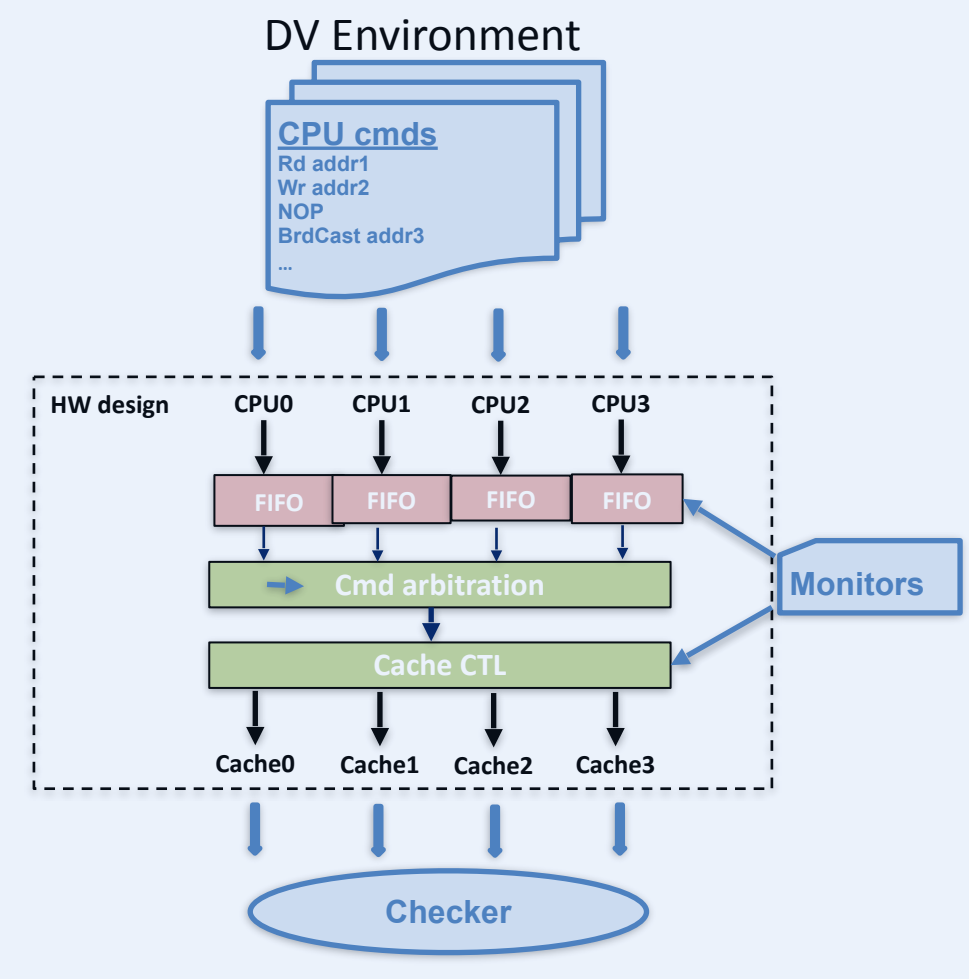
- Increase FIFO depths across all CPU's



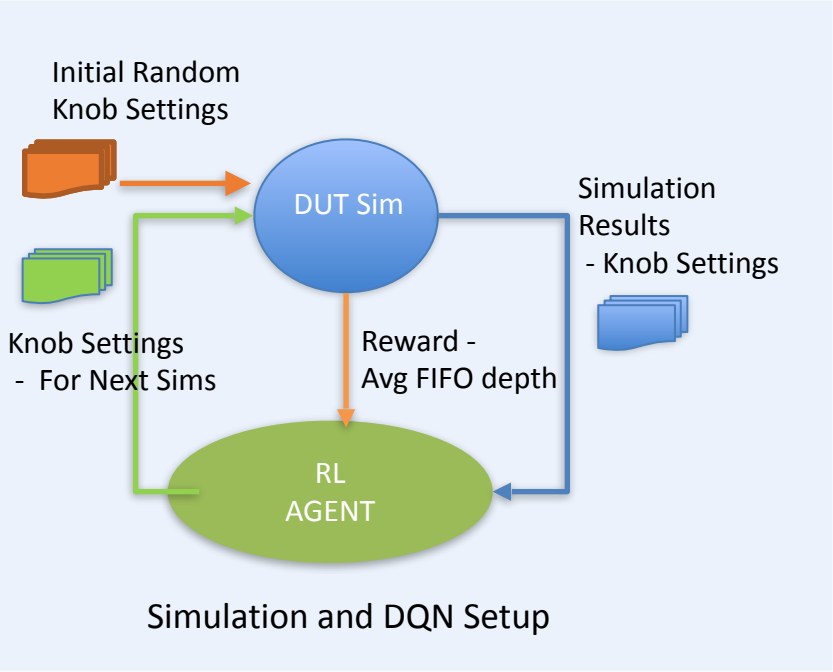
* https://opencores.org/project/mesi_isc

The RL Algorithm for Test Case #1

- Initial Dataset consist of a set of random knobs and their average FIFO depths
- This dataset is fed to a DNN which learns the function that maps the knob settings to the FIFO depth
- The DNN, once trained, can be used to generate new knob settings and prediction for the target variable
- Once trained, the DNN simulates the output for any new unseen setting of input knobs



Test Case #1- Simulation Setup - List of Knobs

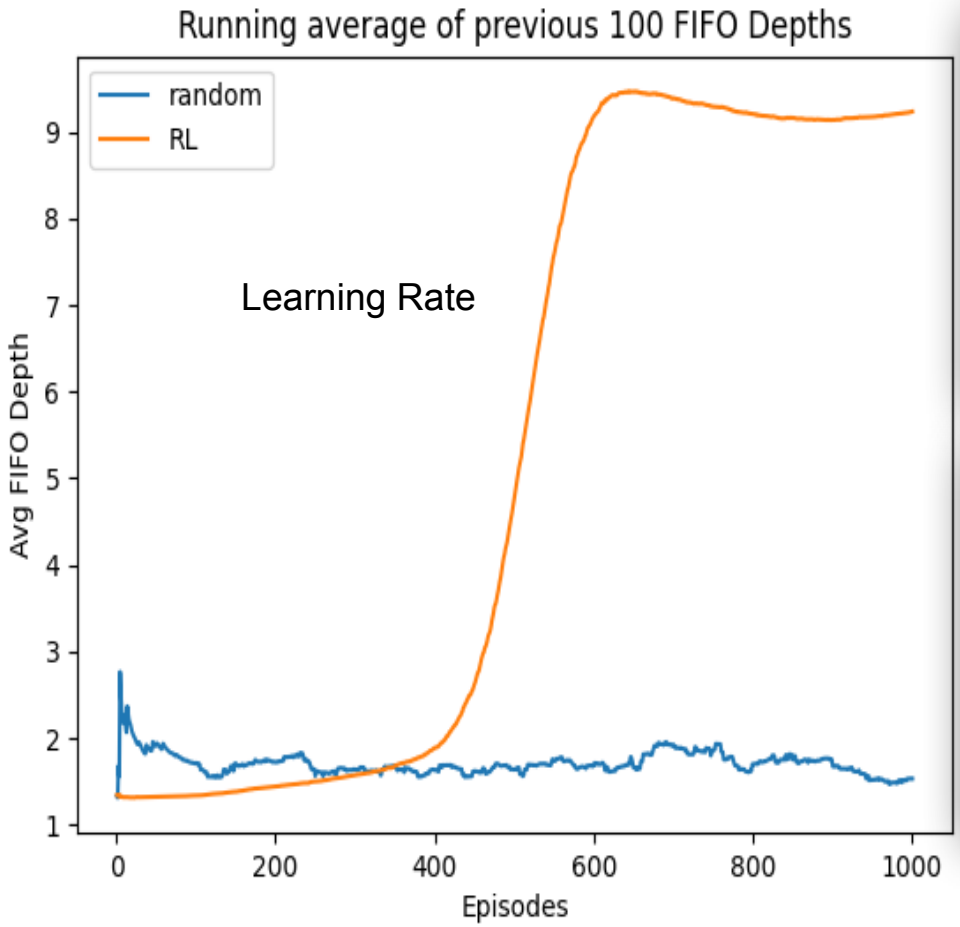


Simulation and DQN Setup

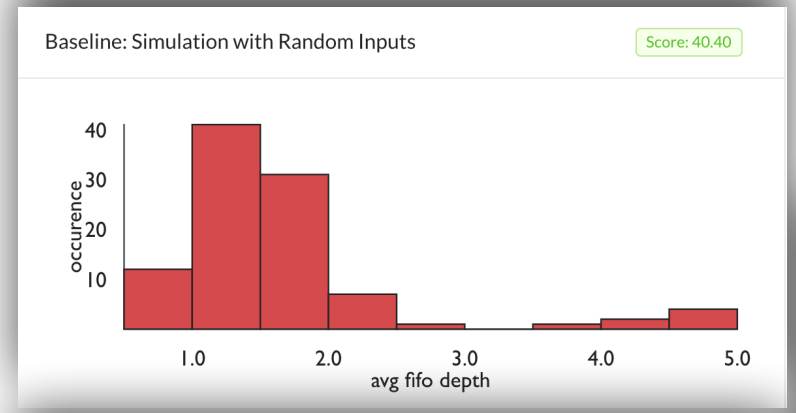
Commands are entered into FIFO when two or more incoming non-NOP requests are to the same address

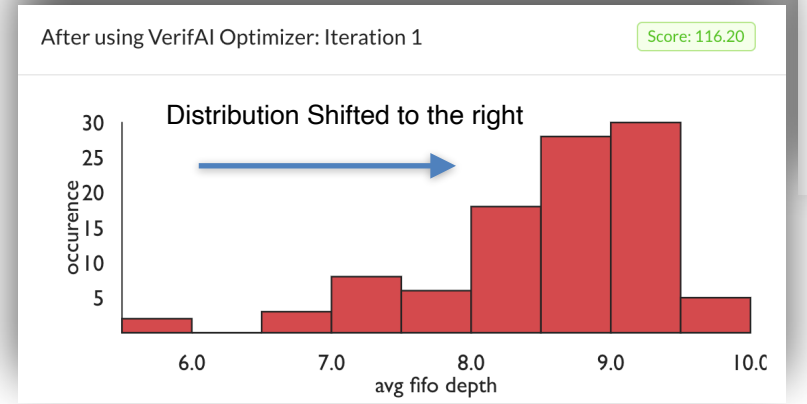
Input Signal	Values	Explanation of type of control
Mbus_cmd_cpu0 (3-bit)	Valid values – 000 to 100	**Knob**: Distribution of commands *Weight* on commands - NOP, WR, RD, WR_BORAD, RD_BROAD
Mbus_cmd_cpu1 (3-bit)	Valid values – 000 to 100	**Knob**: Distribution of commands *Weight* on commands - NOP, WR, RD, WR_BORAD, RD_BROAD
Mbus_cmd_cpu2 (32-bit)	Valid values – 000 to 100	**Knob**: Distribution of commands *Weight* on commands - NOP, WR, RD, WR_BORAD, RD_BROAD
Mbus_cmd_cpu3 (32-bit)	Valid values – 000 to 100	**Knob**: Distribution of commands *Weight* on commands - NOP, WR, RD, WR_BORAD, RD_BROAD
Mbus_addr_cpu0 (32-bit)	Valid range – 0 to 32'hFFFF_FFFF	Index bits are [15:4] for 256kB cache with 16 bytes cache block size and 4-way set associative. tag = [31:16], index = [15:4], offset = [3:0] **Knob**: set individual freedom for each of them.
Mbus_addr_cpu1 (32-bit)	Valid range – 0 to 32'hFFFF_FFFF	**Knob**: set individual freedom for each of them.
Mbus_addr_cpu2 (32-bit)	Valid range – 0 to 32'hFFFF_FFFF	**Knob**: set individual freedom for each of them.
Mbus_addr_cpu3 (32-bit)	Valid range – 0 to 32'hFFFF_FFFF	**Knob**: set individual freedom for each of them.

Test Case#1 Results

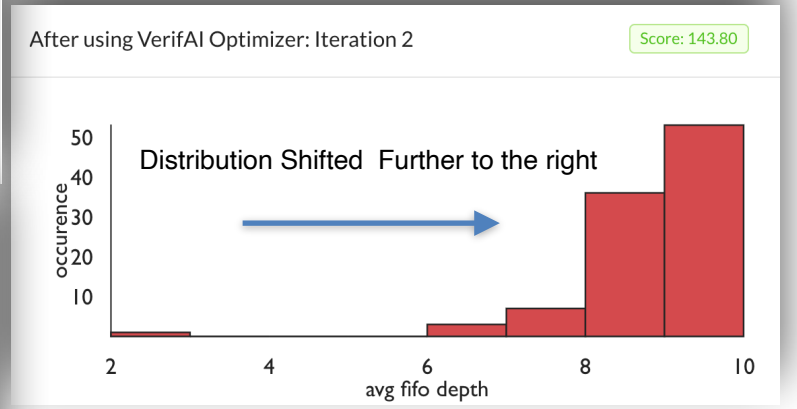


Histogram of Average FIFO Depths





Average FIFO Depths after 2 iterations



Test Case#2 - RISCV CVA6 (Ariane)

- **RISC-V OpenSource design**
  - 64-bit Ariane processor core
  - 6-stage pipeline
  - In-order issue, out-of-order write-back

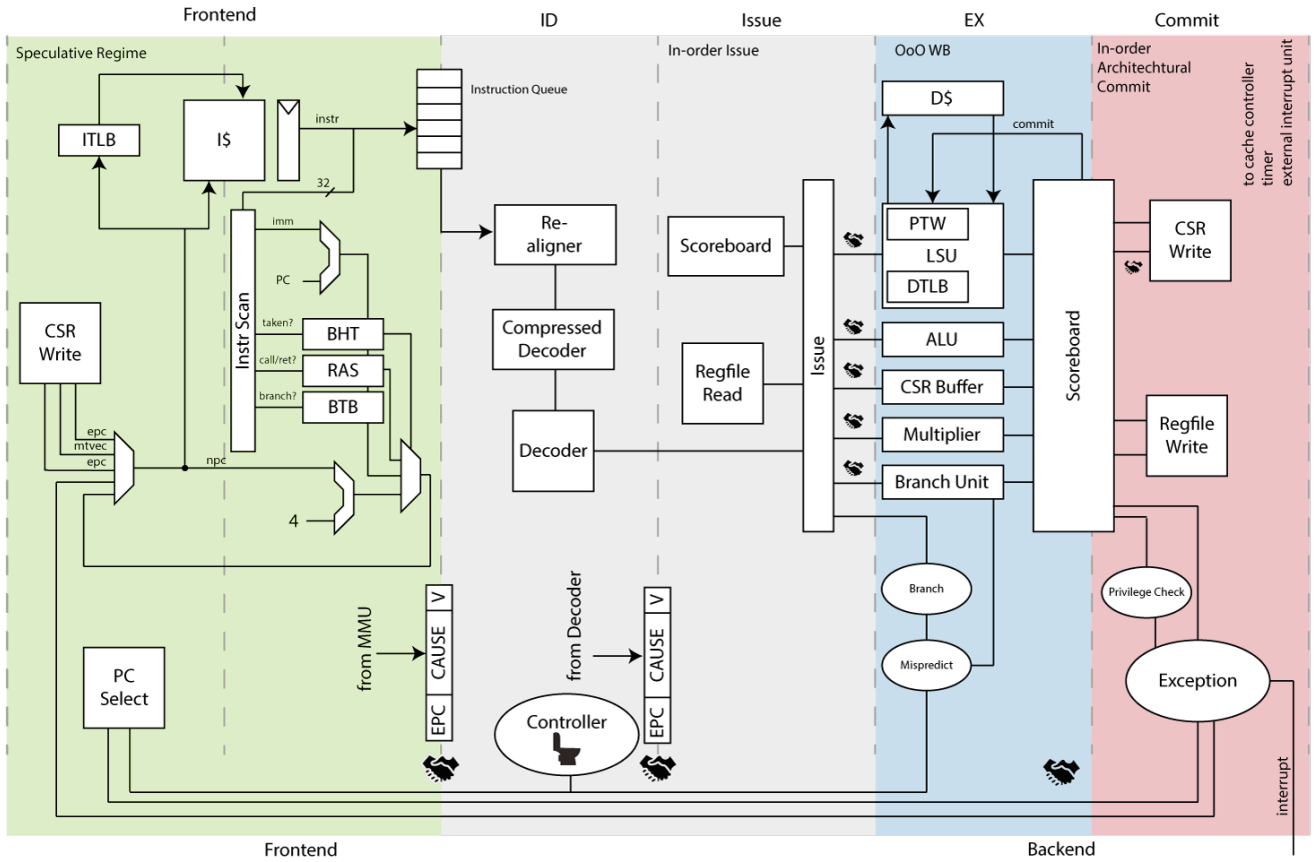
- **Data cache (D$) victim buffer (VB)**
  - Data cache stores recently accessed data from memory
  - New cache allocations cause existing lines to be evicted
  - Modified evicted lines are held in Victim buffer
  - Cache Size – 2 MB, Tag – 44 bit, Index – 12 bit

- **Design complexity of VB**
  - Cache line eviction and memory write-back is complex
  - Subsequent CPU core accesses to VB
  - Snoop accesses to VB
  - VB full requires back-pressure for cache refills

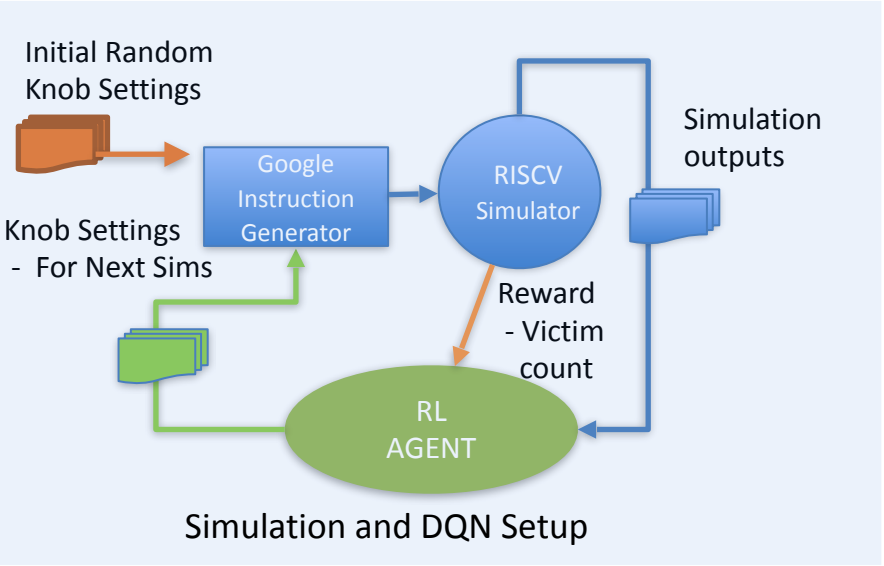
- **Output captured from testbench**
  - average number of victims and percent of hitting of maximum of VB

- **Goal**

-Increase Victim-Buffer Counts



Test Case #2- Simulation Setup - List of Knobs



Simulation and DQN Setup

RL Agent

- Learns each iteration: which knobs impact VB occupancy
- Adjust knob settings for **IG** for next iteration
- Learns to generate settings which increase victim counts

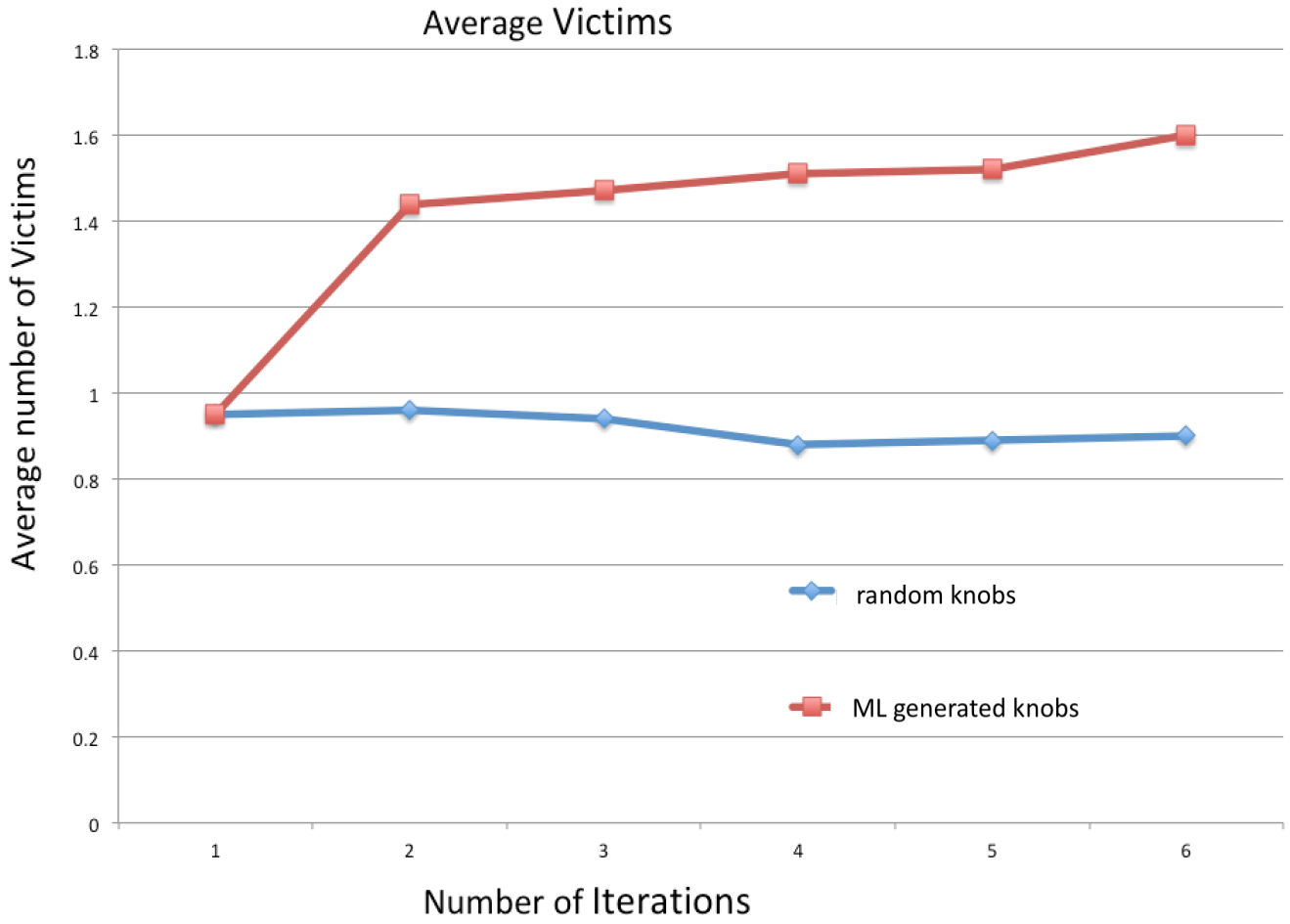
Input Signal	Values	Explanation of type of control
wt_load_store_jump_branch	Valid values – 0 to 100	**Knob**: Distribution of commands *Weight* on commands - (LD/ ST/ JMP/ BRANCH vs other non-complex arithmetic instructions)
wt_other_ins	Valid values – 0 to 100	
wt_loads	Valid values – 0 to 100	**Knob**: Distribution of commands *Individual Weight* on commands between the knob - wt_load_store_jump_branch
wt_stores	Valid values – 0 to 100	
wt_jumps	Valid values – 0 to 100	
wt_branches	Valid values – 0 to 100	
tag range	Valid range – 0 to 7	**Knob**: 0 indicates minimum freedom of tag, 7 indicates maximum freedom of tag
index range	Valid range – 0 to 7	**Knob**: 0 indicates minimum freedom of index, 7 indicates maximum freedom of index

Tag Range	Value
0	44'h 0 – 44'h F
1	44'h 0 – 44'h FF
2	44'h 0 – 44'h FFF
3	44'h 0 – 44'h FFFF
4	44'h 0 – 44'h FFFFF
5	44'h 0 – 44'h FFFFFFF
6	44'h 0 – 44'h FFFFFFFFF
7	44'h 0 – 44'h FFFFFFFFFFF

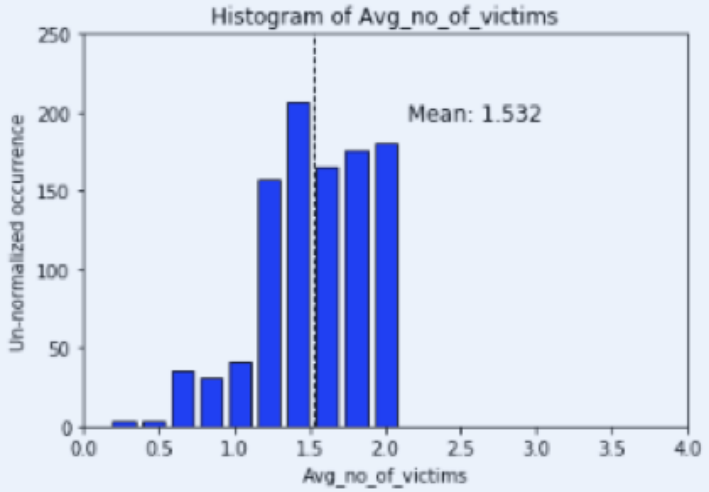
Index Range	Value
0	12'h 0 – 12'h 2
1	12'h 0 – 12'h 4
2	12'h 0 – 12'h 8
3	12'h 0 – 12'h A
4	12'h 0 – 12'h C
5	12'h 0 – 12'h F
6	12'h 0 – 12'h FF
7	12'h 0 – 12'h FFF

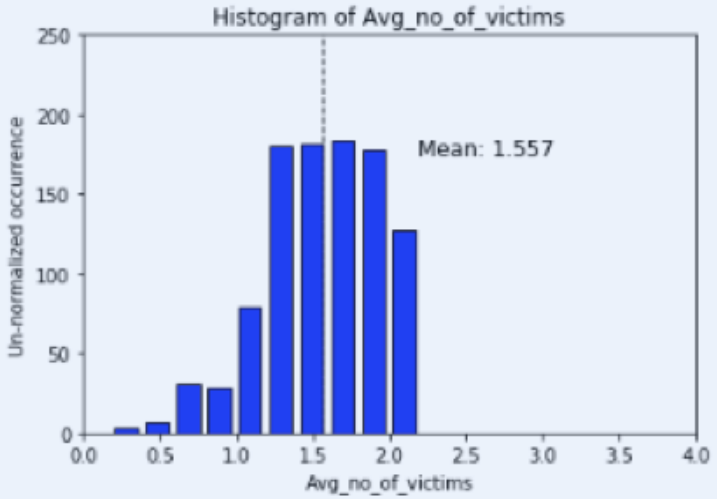
Test Case#2 Results

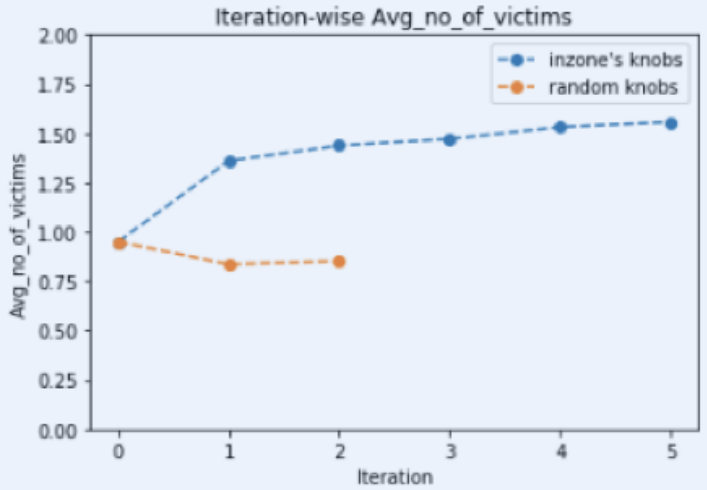
- Average VB occupancy increases from 0.95 to 1.6 after just three iterations, gain of approximately 55%
- RL Agent identifies salient knobs (inputs to Google IG) and exploits them

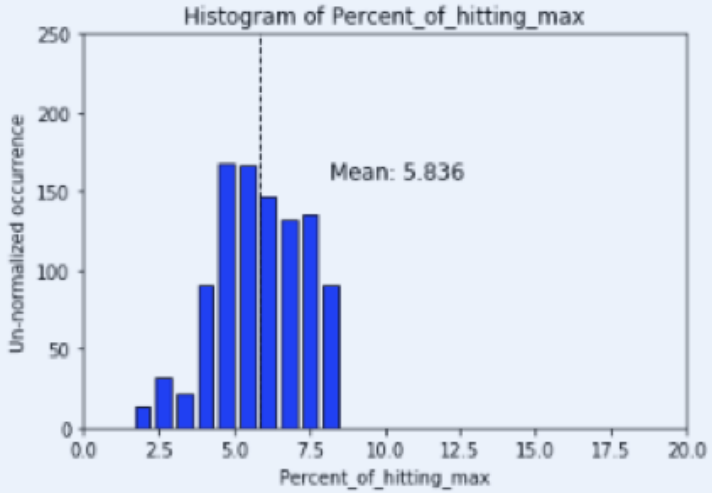


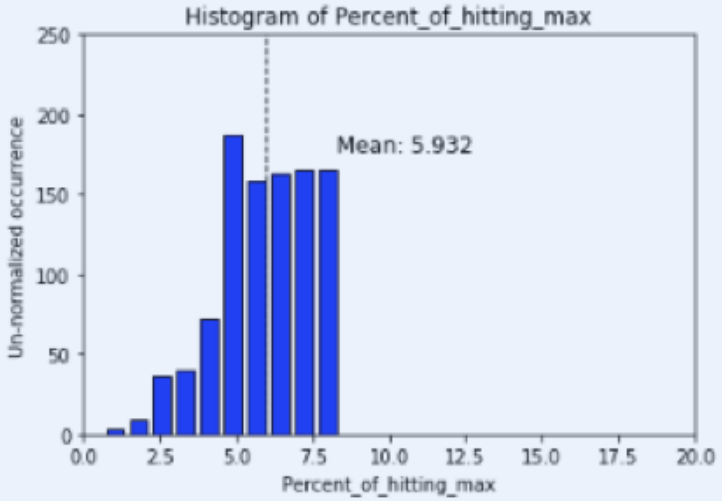
Test Case#2 - Iteration wise results

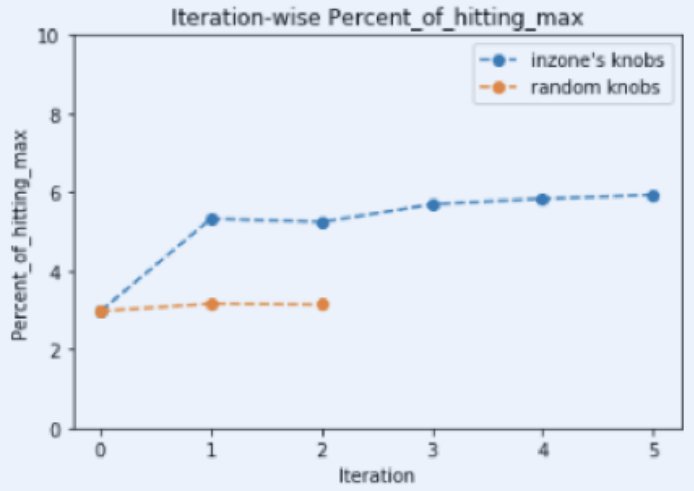












Future Direction - ML for DV

- Reinforcement Learning Algorithms to handle large action spaces (large number of knobs) is an active area of research
    - DDPG (Deep Deterministic Policy Gradient)
    - Soft Actor Critic (A3C)
- Explore Statistical Algorithms
    - Contextual Bandit
- In simulation RL (online-RL)

Conclusion

- We have demonstrated the value of using Machine Learning techniques on industrial designs to do significantly better than constrained random methods (UVM etc..)
- Using RL for DV can save months of verification resource for Coverage Improvement and Early Bug Finding
- DNN's provide a mechanism to fit non-linear functions that can mimic complex behaviors - even of a simulator