2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Closing Functional Coverage With Deep Reinforcement Learning
A Compression Encoder Example

Eric Ohana

Queensland University of Technology

CITEC - The University of Bielefeld

accellera
SYSTEMS INITIATIVE

Agenda

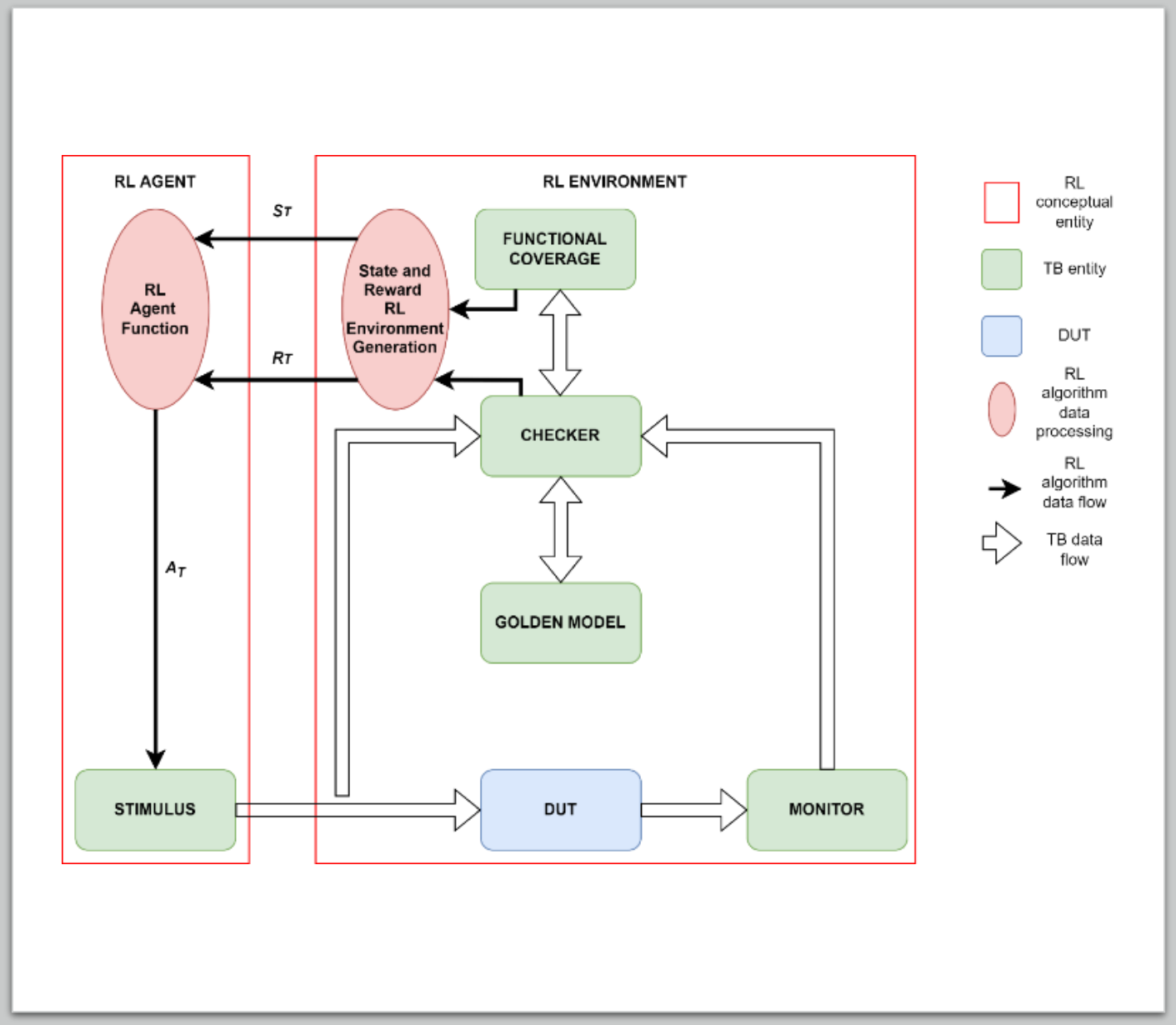
- Introduction
- RTL Verification and Reinforcement Learning
- The LZW Compression Encoder
- Co-Simulating the RTL Design and the Deep Q-Learning (DQN) Agent
- The DQN Agent
- The Simulation Results (Standard and DQN)
- Summary and Conclusion
- Questions

Introduction

- Reaching the last functional coverage (FC) percentages on a design, has traditionally been an obstacle to verification closure
- In this presentation, we tap into reinforcement learning tools and techniques to assist in the simulation based constrained random coverage driven functional verification
- Specifically, we use a DeepMind Technologies inspired Deep Q-Learning (DQN) agent to target a functional coverage closure category reluctant to standard means

RTL Verification and Reinforcement Learning

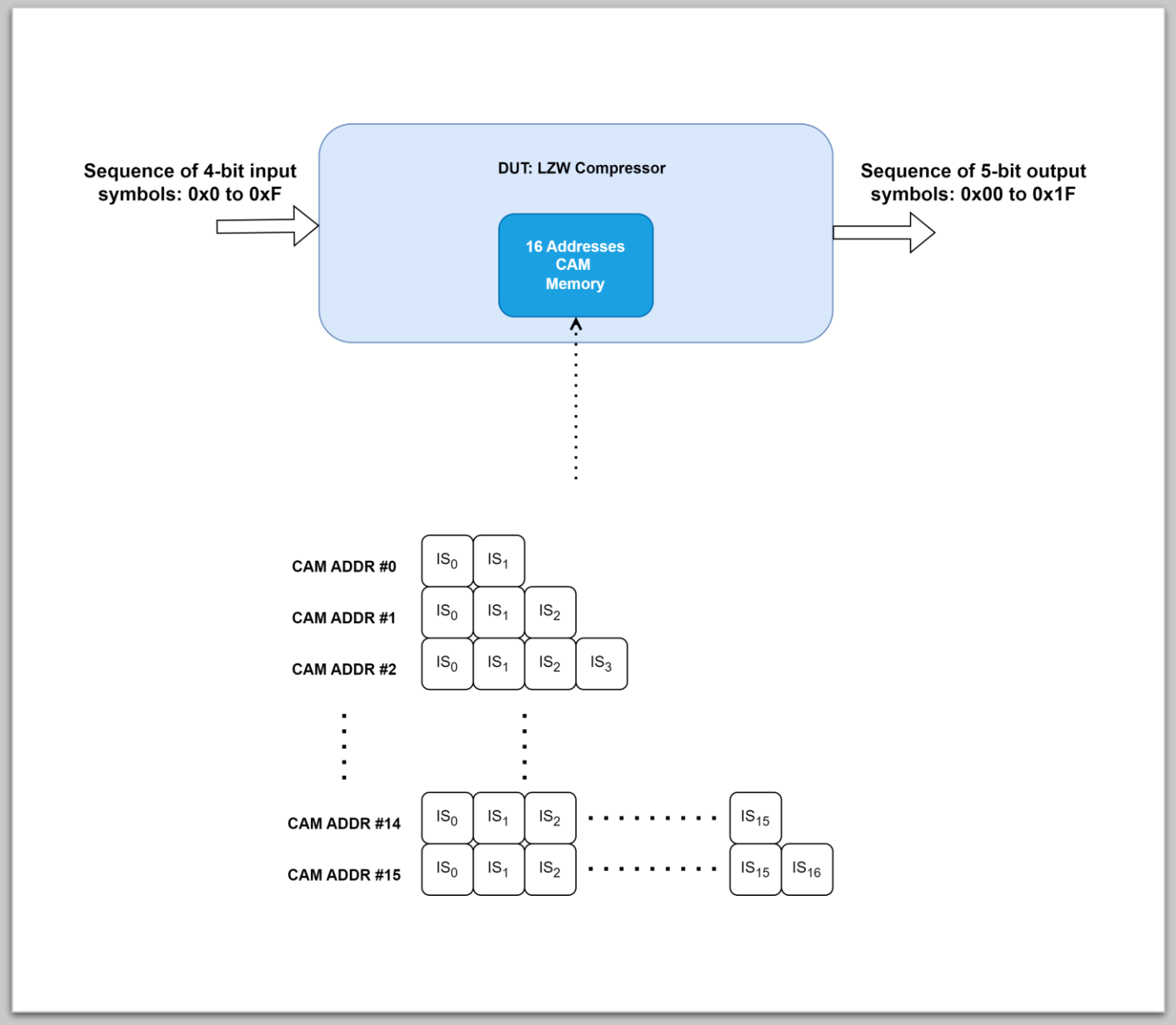
- A Reinforcement Learning (RL) system is a sequential interaction between an agent and an environment
- At every iteration, the agent processes a state and a reward value from the environment to issue an action back to this environment
- Action <> Transaction
- Reward <> FC bins hits/misses, the harder the FC, the higher the reward
- State <> Some representation of how we reached the current FC state (Markov Decision Process)



The LZW Compression Encoder

Timestep	Input Symbol 4-bit HEX	CAM[address]	Output Symbol 5-bit HEX
Start: #1	A	-	-
#2	B	CAM[0] = AB	0A
#3	A	CAM[1] = BA	0B
#4	B	Match on CAM[0]	-
#5	A	CAM[2] = ABA	10
#6	B	Match on CAM[0]	-
End: #7	A	Match on CAM[2]	12

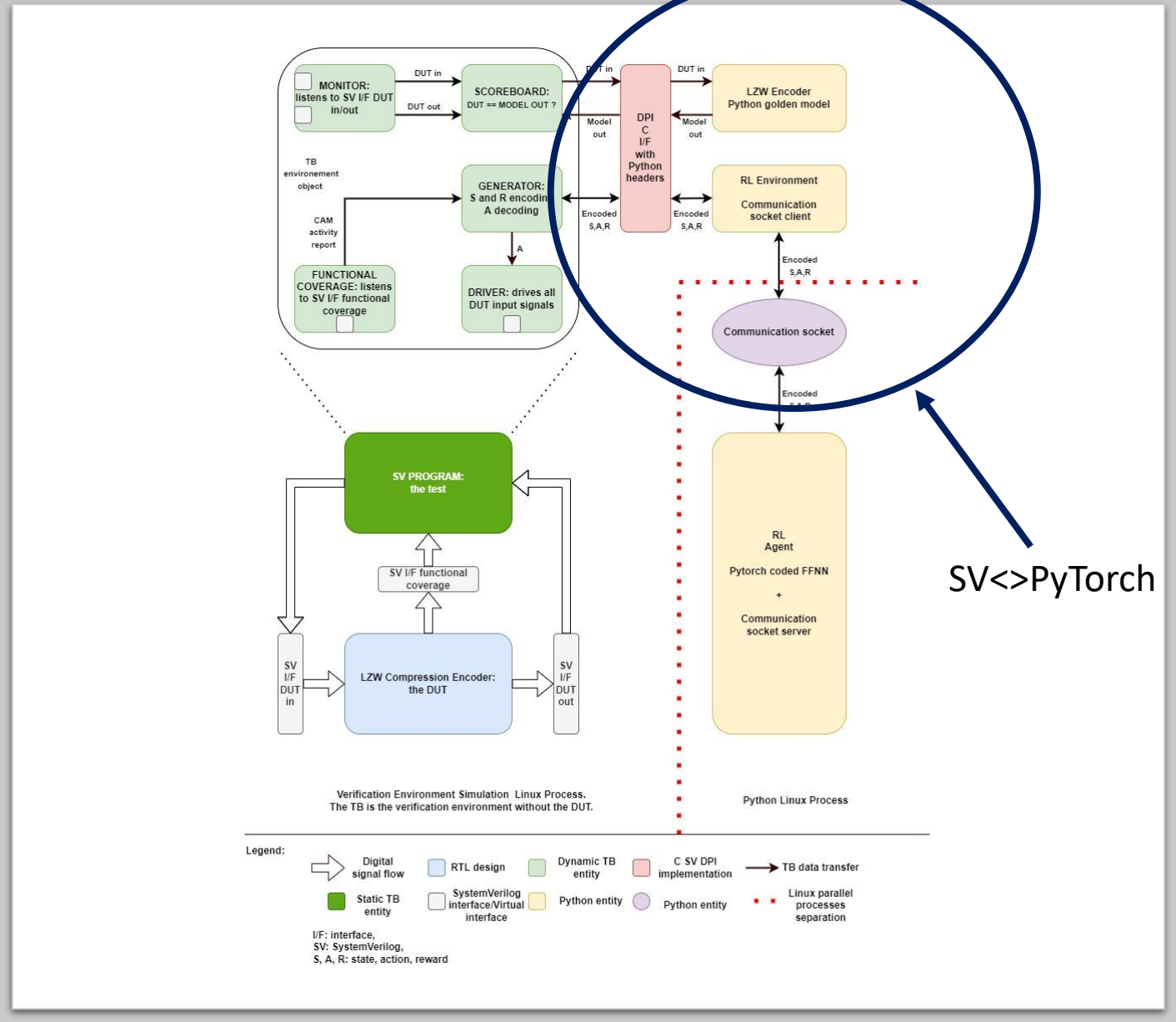
- The shortest sequence is of 2 input symbols. There are 136 CAM write FC bins to cover out of 152 CAM locations
- The CAM write functional coverage category necessitates very specific sequences!
- It is close to impossible to reach them randomly!
- Can our DQN agent help?



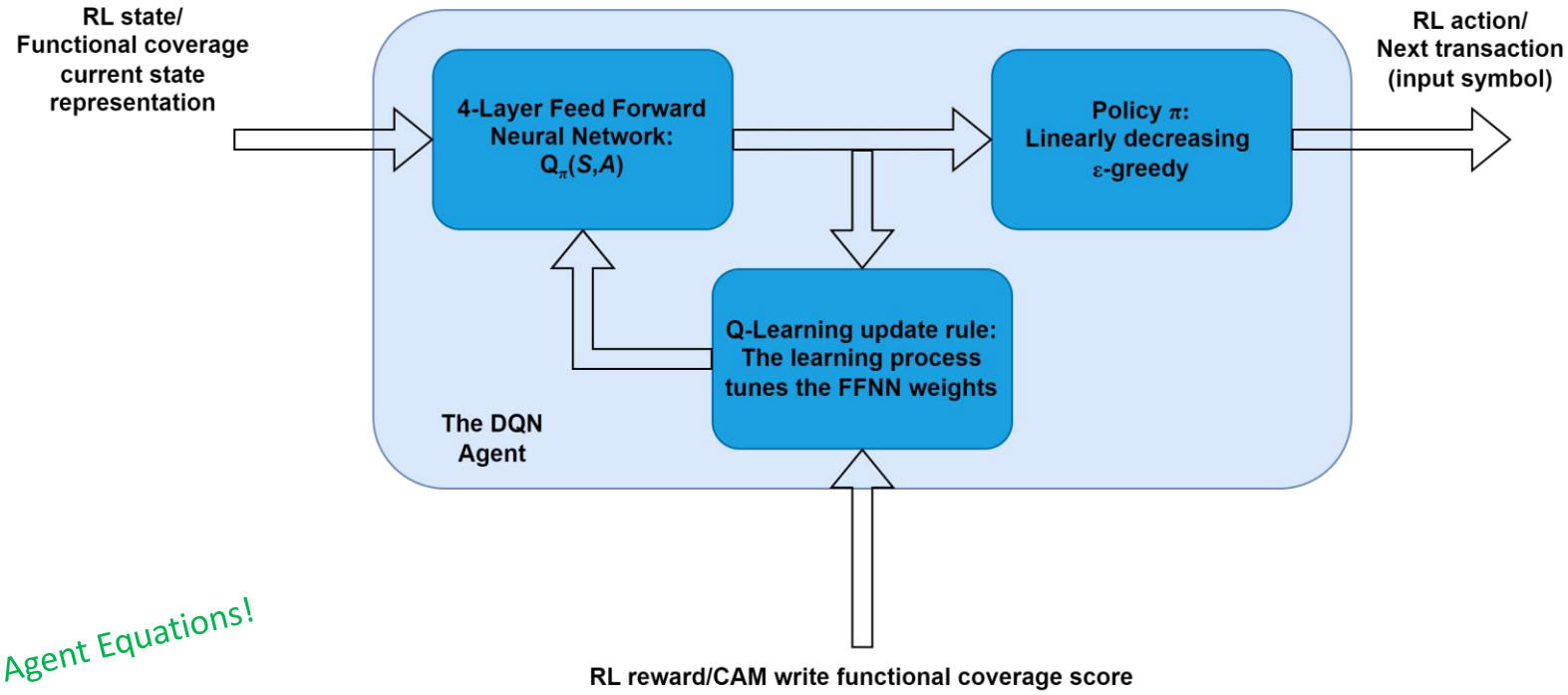
accellera SYSTEMS INITIATIVE

Co-Simulating the RTL Design and the Deep Q-Learning (DQN) Agent

- Our RL agent runs in Python and PyTorch
- Our SV D&V env runs on Aldec Riviera-PRO (research edition)
- Using SV DPI/C/C Embedded Python and a client/server networking protocol, both can communicate efficiently!



The DQN Agent

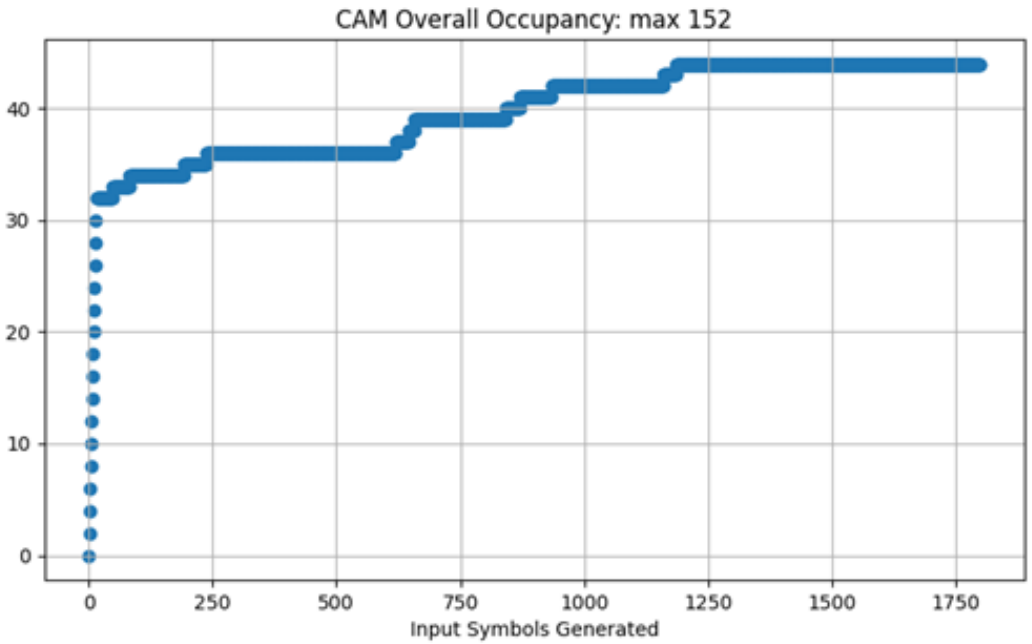


See Appendix for the DQN Agent Equations!

The Standard Simulation Results

- By running a uniform input symbols distribution, over many episodes, where an episode starts with an empty CAM and ends with a full CAM.
- We have managed to hit 28 CAM write bins out of 136 with a CAM overall occupancy of 44 out of 152

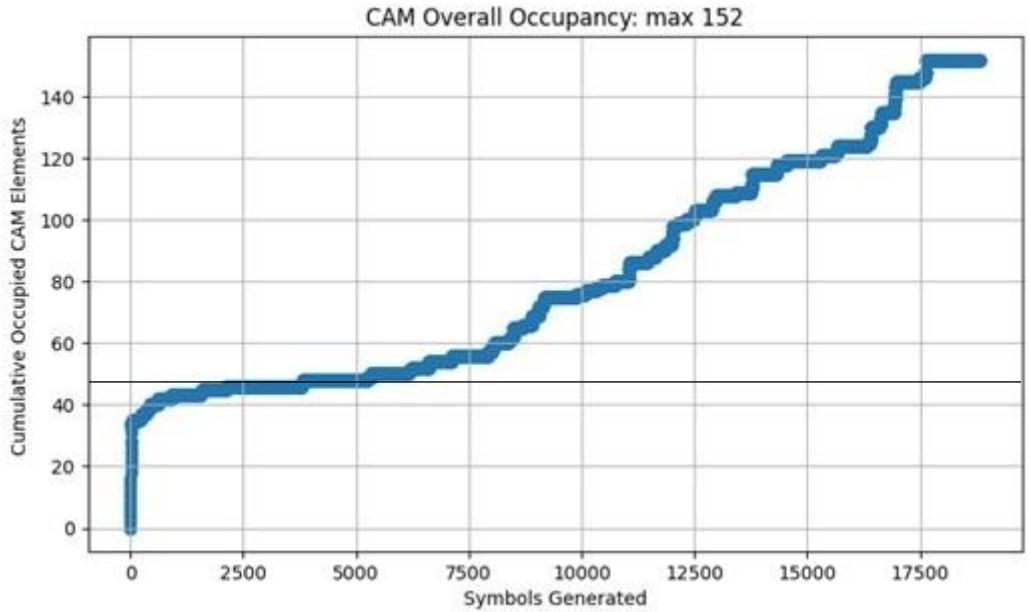
29% CAM write FC



The DQN Simulation Results

- We first run 500 episodes with an ε-greedy linearly decreasing
- We observe a constant incremental increase in the CAM overall occupancy
- We have managed to hit 133 CAM write bins out of 136 with a CAM overall occupancy of 152 out of 152

97.8% CAM write FC



The DQN Simulation Results

- To target the 3 remaining CAM write FC bins, we run 750 episodes, to allow a smoother transition from exploration to exploitation
- By merging both DQN simulations, we reach 100% CAM write functional coverage

Summary and Conclusion

- We identified a functional coverage category which is hard to fully cover using standard means: the CAM write functional coverage for the LZW compression encoder
- We defined an action-value function for a DQN agent, linking between input symbols and the expected future rewards expressed as CAM write functional coverage bins hits
- We used a simple ε–greedy policy allowing a transition from exploration (full randomness) to exploitation (using reinforcement learning lessons) to reach 100% functional coverage

Questions

Appendix: The DQN Agent in Equations

- A Deep Q-Networks (DQN) agent uses a neural network to model an action-value function
- Our action-value function called $\mathbf{Q_\pi(S,A)}$ processes the environment state $\boldsymbol{S}$ and issues an output vector value representing the expected future reward $\boldsymbol{R}$ for every action $\boldsymbol{A}$ called $\mathbf{E(R|A,S,\ \pi)}$
- In the verification realm, it just means that, given the current functional coverage state, every input symbol we can chose for the next transaction, has a particular impact on the CAM write functional coverage overall score
- $\boldsymbol{\pi}$ is called a policy and is just a way of selecting the next transaction from the output vector $\mathbf{E(R|A,S,\ \pi)}$