2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE OCTOBER 26-27, 2021

Machine Learning for Coverage Analysis in Design Verification

V Jayasree, Engineer, Qualcomm India Private Limited (*vjayasre@qti.qualcomm.com*)

***Abstract*— This paper aims at formulation of an algorithm for selection of test cases for code coverage, based on the analysis of initial coverage database comprising basic test cases, using Machine Learning (ML) models. From the predicted outputs of the ML model, the relative impact of features on Coverage Percentage and Simulation Time is analyzed. The optimal set of test cases to achieve higher coverage numbers in shorter duration is determined from the inferences of the analysis.**

Keywords— verification; coverage; machine learning; automation

I. INTRODUCTION

Coverage is an integral aspect of Register Transfer Level (RTL) Verification, for maximizing the accuracy of testing of the design. At present, manual analysis of coverage database is performed by Design and Verification engineers, towards the closing phase of projects. The implementation of this algorithm would help in saving considerable amount of manual effort and time spent for coverage analysis, to develop test plans and meet stringent deadlines for hectic tape-out schedules.

II. RELATED WORKS

Automation of coverage analysis has been on the radar in recent years, with the evolution of ML. Research on applications of various ML algorithms to coverage have been carried out. References [2] and [3] focus mainly on stimulus generation for functional coverage. However, modelling a specific type of coverage may not be viable in situations where all kinds of coverage need to be optimized. The algorithm in this paper provides a comprehensive solution, where the impact of tests on different types of coverage can be integrated and a consolidated set of tests/features can be chosen by the verification engineer. Further, in the computation of "Score" mentioned in the "Implementation Methodology" section, the engineer can tune the weights of scalars as per requirement in different projects. Thereby flexibility and reusability are some of the key aspects of novelty of this paper.

III. BACKGROUND

A. *Existing flow of coverage analysis*

In RTL Verification, the existing flow of coverage closure involves the following steps:

- Regressions are run for a large set of tests by the verification engineer
- The coverage database is manually analyzed by the design and verification engineers
- Uncovered code is covered by re-running a directed set of tests, post manual analysis

B. *Proposed flow of coverage analysis*

Understanding the impact of various features and reduction of manual effort and time spent on code coverage are the primary goals of the paper. The proposed solution involves the following stages:

- Selection of important arguments/parameters used to configure verification tests, as inputs to ML model.
- Procuring the individual coverage database for a set of basic test cases. The percentage of coverage can be obtained from Unified Report Generator (URG) report, generated by the simulation tool. The run-time of each test can also be obtained by parsing log files.
- The input parameters (configurations) used in tests are considered as the features for the ML model. The coverage percentage and simulation time are considered as the labels (outputs) for the model.

- The training and test datasets can be created by splitting the initial database in the ratio of 75:25. Using the training data set, we generate the models for supervised learning algorithms in any software such as Python/Matlab.
- The efficiency of a particular model can be tested by obtaining the cross-valuation score of test data set. The model that maximizes the score can be chosen.

Once the data is modelled, the verification engineer can predict the coverage percentage of various combination of features. The impact of each feature on the overall coverage percentage, for every coverage type, can be analyzed and understood. If a particular feature does not improve the coverage much, running too many seeds of that test can be prevented. This will help in improving efficiency of coverage analysis. Figure 1 depicts the difference in the current flow and the proposed flow for speeding up Coverage Closure through Machine Learning.

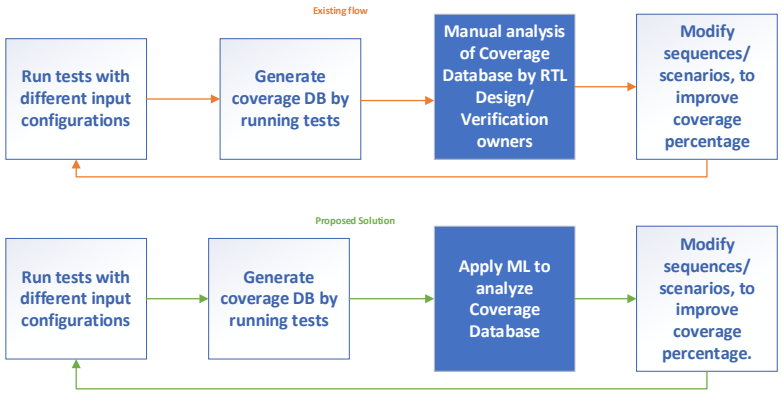


Figure 1. Existing vs. Proposed flow of Coverage Analysis

IV. IMPLEMENTATION METHODOLOGY

A prototype of the aforementioned solution has been implemented with regression results of RTL code of an IP module, using scikit-learn package in Python. Analysis has been performed on a dataset containing coverage numbers and run-time of tests, procured from simulations. Three different regression models were experimented to be used in the algorithm, namely:

- Linear regression
- Decision Tree regression
- Random Forest regression

A. Inputs and Outputs of the ML Model

- **Features:**
  - Parameters with Bernoulli distribution, defined in the inputs of test cases, based on which the functionality and corresponding RTL code coverage will differ. For example, presence (1) or absence (0) of encryption in the use cases of a product.
- **Labels:**
  - Per-test Coverage Percentage
  - Per-test Run-time of simulation
- **Training Dataset:**
  - Features and labels of basic testcases
- **Test Dataset:**
  - Different combinations of features chosen at random
- **Predicted Outputs:**
  - Predicted values of coverage percentage and simulation time for test dataset

Figure 2 depicts the inputs and outputs of the ML Model.

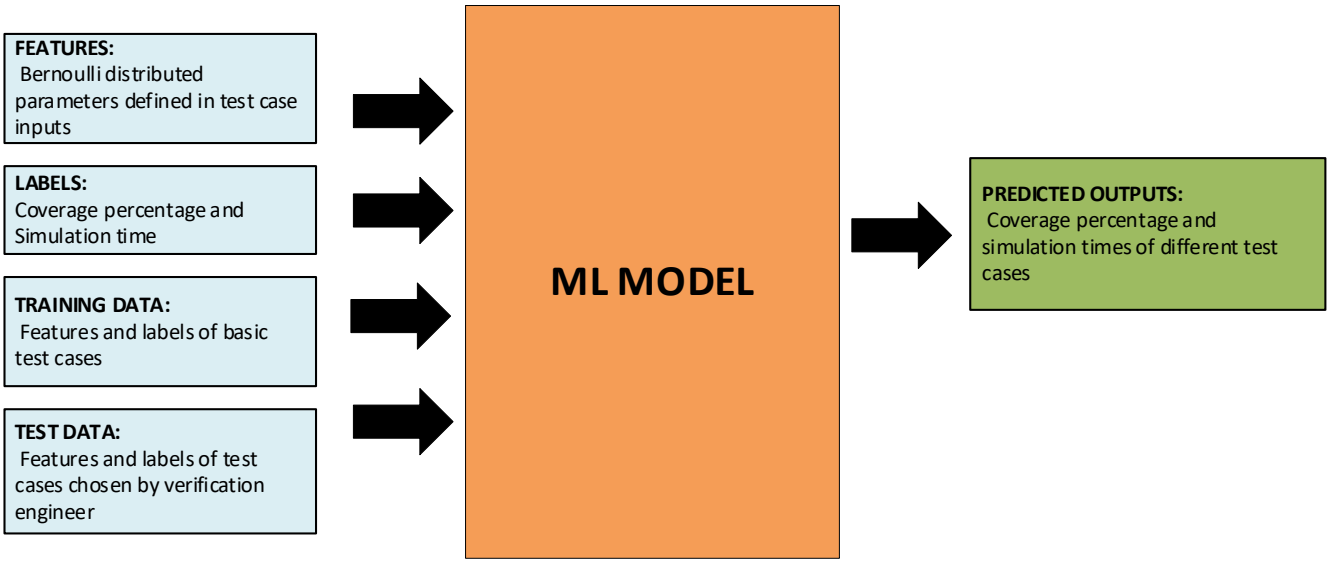


Figure 2. Inputs and Outputs of ML Model

B. Selection of features, tests

For a given set of inputs (combination of features defined by the verification engineer), the Coverage Percentages and Simulation Times can be predicted from the trained ML model. From these values, the algorithm involves computation of a final parameter called **"Score",** based on which the subset of tests for achieving a higher coverage in shorter time can be chosen. The parameter "Score" is computed as a linear combination of the outputs of the ML model, i.e. the Coverage Percentages for various types of coverage (such as line, conditional, branch, FSM) and the Simulation time. The scalars involved in the computation of "Score" can be tuned based on the priority assigned to each output and their impact on speeding up the coverage closure process. The value of "Score" for each combination of features is used to select the optimal set of tests and features that would maximize the coverage in minimal run-time.

V. RESULTS OF PROTOTYPE IMPLEMENTATION

The results obtained from the prototype implementation of the algorithm are summarized in this section.

A. Accuracy of ML models

The **"Accuracy Scores"** of three different ML models are captured in Table 1. From the Accuracy Scores, it is seen that **Decision Tree (DT)** or **Random Forest (RF)** models would be better choices for the given data set. The Actual values of overall Coverage Percentage and Simulation Time for various testcases are presented in Table 2. The Actual and Predicted values (from Random Forest and Decision Tree models) of Coverage Percentages for Line, Condition, Branch and FSM coverage, and the Simulation Time are presented in Table 3 and Table 4. Graphical comparisons of the Predicted and Actual outputs are presented in Figure 3 and Figure 4.

Table 1. Accuracy Scores of ML Models

ML Model	Accuracy Score
Linear Regression	0.55
Decision Tree	0.99
Random Forest	0.955

Table 2. Coverage percentage and Simulation time (Actual values)

Test Inputs (Features)								Actual Test Outputs (Labels)	
Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Coverage Percentage Actual (%)	Simulation Time Actual (s)
1	1	1	1	1	1	0	0	64.2	800
1	1	1	1	1	0	1	0	64.2	800
0	1	1	0	1	0	1	0	53.8	210
0	1	1	0	0	0	1	0	49.74	209
1	0	1	0	0	0	0	1	54.34	556

Table 3. Coverage percentage and Simulation time (Actual and Predicted values)

Coverage Percentage Actual (%)	Coverage Percentage: RF (%)	Coverage Percentage: DT (%)	Simulation Time Actual (s)	Simulation Time: RF (s)	Simulation Time: DT (s)
64.2	63.3264	64.2	800	774.33	800
64.2	63.024	64.2	800	772.04	800
53.8	53.6281	54.6	210	270.86	228
49.74	51.5003	47.14	209	275.19	215
54.34	52.6044	53.07	556	507.09	585

Table 4. Coverage percentages of different coverage types (Actual and Predicted values)

Line		Conditional		Branch		FSM	
Actual values (%)	Predicted values (%)	Actual values (%)	Predicted values (%)	Actual values (%)	Predicted values (%)	Actual values (%)	Predicted values (%)
56.57	56.8628	49.49	49.73	50.14	50.43	22.55	22.55
59.38	59.1403	54.41	53.87	53.86	53.98	31.37	30.15
59.48	60.5786	56.75	57.6	54.61	55.11	27.94	29.17
69.59	68.3014	70.67	70.67	65.73	65.73	50.25	50.25
69.59	69.0457	70.67	70.67	65.73	65.73	50.25	50.25

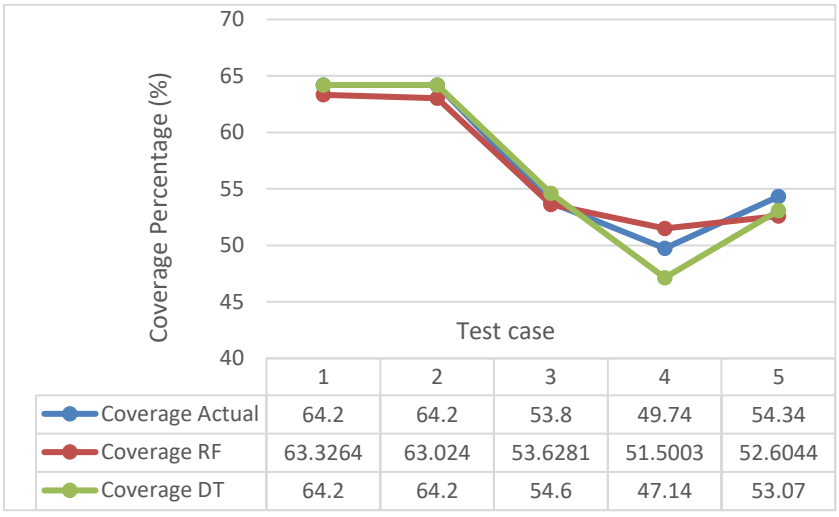
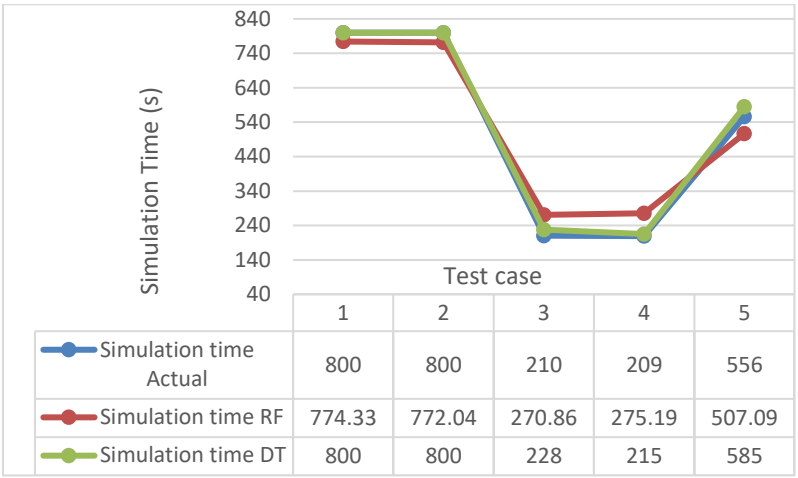


Figure 3. Coverage percentage: Actual vs. Predicted Outputs



	1	2	3	4	5
Simulation time Actual	800	800	210	209	556
Simulation time RF	774.33	772.04	270.86	275.19	507.09
Simulation time DT	800	800	228	215	585

Figure 4. Simulation Time: Actual vs. Predicted Outputs

B. Computation of Score

Score as a function of Coverage Percentage and Simulation Time

In this model, the 'Score' for each test is expressed as a linear combination of Coverage Percentage and Simulation Time, using the formula given below:

Score = α * (Coverage percentage) – β * (Simulation Time)

The formula is based on the idea that Score (speeding up of coverage) is directly proportional to coverage percentage and inversely proportional to simulation time. The test with a higher score indicates a higher coverage percentage, with a lower simulation time. The Computed Score as a function of Overall Coverage Percentage and Simulation Time for various test inputs with $\alpha = 10$ and $\beta = 0.1$ are presented in Table 5.

Score as a function of the coverage percentages of different sub-types of coverage

In this model, the 'Score' is expressed as a linear combination of various sub-types of code coverage, using the formula given below:

Score = α * (Branch coverage %) + β* (Conditional coverage %) + γ * (FSM coverage %) + δ* (Line coverage %)

Here, all four scalar multipliers are positive, since all four coverage sub-types impact the Score positively. The Computed Score as a function of Coverage Percentage of different coverage types for various test inputs with $\alpha = 0.9$, $\beta = 0.9$, $\gamma = 0.6$, $\delta = 1.0$ are presented in Table 6.

C. Relative Impact of features on coverage

The relative impact of each input feature on the overall Coverage percentage and Simulation time is computed based on the below mentioned algorithm:

- From the input data, the mean values of Coverage percentages and Simulation Times are computed.
- For each feature, the difference in Coverage percentages and Simulation Times when the feature is present or absent is computed as follows:
  - **Coverage Percentage:**
    - Coverage increase with feature = Coverage percent with feature – Mean coverage percent

- Coverage increase without feature = Coverage percent without feature – Mean coverage percent
- Coverage impact with feature = Coverage increase with feature – Coverage increase without feature

- **Simulation (Run) Time:**
  - Run time increase with feature = Run time with feature – Mean run time
  - Run time increase without feature =Run time without feature – Mean run time
  - Run time impact with feature = Run time increase with feature – Run time increase without feature

- The relative impact of each feature is computed as follows

Feature Impact = α * (Coverage increase with feature) – β * (Run time impact with feature)

Figure 5 depicts the relative impact of each input feature on the Coverage Percentage and Simulation Time, with $\alpha = 10$ and $\beta = 0.1$. From the results of experimentation, it can be inferred that Features 1 to 5 have the maximal impact on improving the coverage numbers in shorter run-times, and Feature 6 to 8 have relatively lesser impact.

D. Productivity improvement

Table 7 provides the average increase in Simulation Time due to each feature. Since feature 6,7 and 8 have only a minimal impact on improving the coverage numbers, it is sufficient to run fewer reseeds of regression, with those features. CPU time saved by running lesser seeds of test cases is calculated as follows:
Number of reseeds of test cases run with features 6,7,8 enabled, before implementation of algorithm = 20
Number of reseeds of test cases run with features 6,7,8 enabled, after implementation of algorithm = 5

CPU Time Saved = (Number of reseeds of test cases not run for features 6,7,8) *(Average Simulation time increase due to features 6,7,8)
= (20 - 5) *(39.2 + 66.56 + 54.85) ≈ 2410 seconds

Percentage savings = (CPU Time Saved / (Average run time per seed * Number of reseed runs)) *100
= (2410/ (468.2*20)) *100 = 25.7%

Number of NAND2 gates in module (14 nm technology) = 300000 instances

Thereby, around **25% of CPU time and machine resources** are saved in the prototype implementation with a design complexity of 300000 NAND2 gates in 14 nm process. Figure 6 depicts the reduction in the aggregate run time of tests for achieving maximal coverage.

Table 5. Computed Scores as a function of Overall Coverage Percentage and Simulation Time (Random Forest and Decision Tree models)

Test Inputs (Features)								Computed Scores	
Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Total Score: RF	Total Score: DT
1	1	1	1	1	1	0	0	555.831	562
1	1	1	1	1	0	1	0	553.036	562
0	1	1	0	1	0	1	0	509.195	523.2
0	1	1	0	0	0	1	0	487.484	449.9
1	0	1	0	0	0	0	1	475.335	472.2

Table 6. Computed Scores as a function of Coverage percentage of different coverage types (Random Forest and Decision Tree models)

Test Inputs (Features)								Computed Scores	
Feature 1	**Feature 2**	**Feature 3**	**Feature 4**	**Feature 5**	**Feature 6**	**Feature 7**	**Feature 8**	**Total Score: RF**	**Total Score: DT**
1	1	1	1	1	1	0	0	210.1118	212.787
1	1	1	1	1	0	1	0	207.6865	211.4204
0	1	1	0	1	0	1	0	166.6064	168.4108
0	1	1	0	0	0	1	0	160.6769	162.6761
1	0	1	0	0	0	0	1	150.4944	149.723

Table 7. Average simulation time increase for each feature

Feature	Simulation time increase (s)
1	102.884
2	17.62
3	117.2
4	260.9
5	54.35
6	39.2
7	66.56
8	54.85

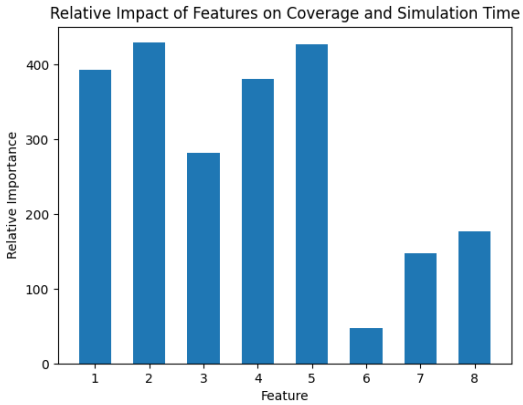


Figure 5. Relative Impact of features on coverage

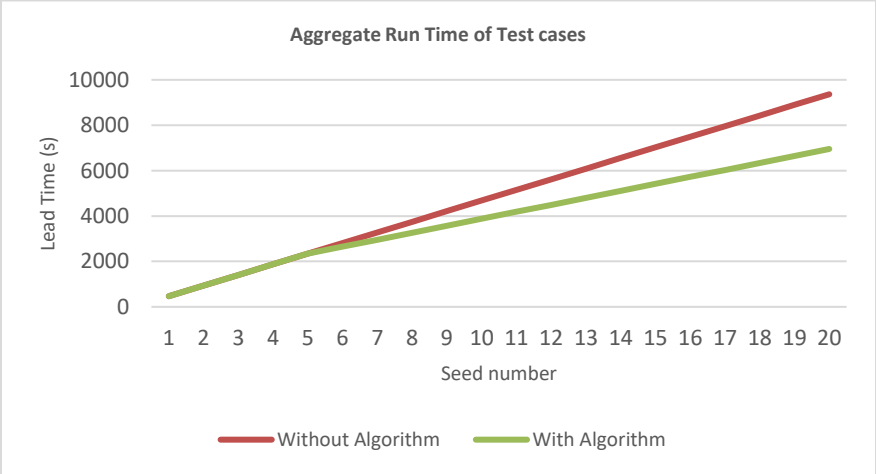


Figure 6. CPU time reduction with algorithm

VI. APPLICATIONS

A. *Advantages*

The key advantages of implementing this algorithm are as follows:

- Reduction in time and machine resources for sanity testing on large modules, due to intelligent selection and elimination of test cases
- Identification of redundant tests from code coverage perspective and prevention of re-seed runs
- Deeper knowledge of functionality of input parameters and their correlation with RTL code
- Selection of constraints for modern Metric-Driven Verification techniques

B. *Future Scope*

The algorithm is a prototype to demonstrate the usage of ML in code coverage analysis. Using the algorithm, the impact of any new feature addition and its combination with existing features can be analyzed. For further improvement, a model can be developed to merge the database of both code and functional coverage and provide an optimal combination of tests and features for maximizing the same. Thereby, it can be concluded there is a huge scope for Machine Learning in improving the efficiency of coverage analysis in verification.

REFERENCES

[1] Kerstin Eder, "Coverage-directed test generation automated by Machine Learning - A review," (https://www.researchgate.net/publication/220306081_Coverage-Directed_Test_Generation_Automated_by_Machine_Learning_-_A_Review)

[2] William Hughes, Sandeep Srinivasan, and Rohit Suvarna, "Optimizing design verification using Machine Learning: Doing better than random," (https://arxiv.org/ftp/arxiv/papers/1909/1909.13168.pdf)

[3] S. Gogri, J. Hu, A. Tyagi, M. Quinn S. Ramachandran, F. Batool, and A. Jagadeesh, "Machine Learning-guided stimulus generation for functional verification," (DVCON 2020: http://confcats-event-sessions.s3.amazonaws.com/dvcon20-us/papers/03_1.pdf)