Validation Test Case Prioritisation Using Deep Learning Hybrid Approach

Loo Willam, Yip Kin Choy

Intel Microelectronics Sdn. Bhd.

willam.loo@intel.com; kin.choy.yip@intel.com

1 Abstract

Exhaustive testing and coverage are not effective methods to perform the validation in manual execution, automation, and continuous integration and development environment. In the current mode of work, we encountered a backlog of execution tasks for up to three weeks instead of the planned one-week duration for completion. Therefore, this research will prioritize test case prioritization which allows the validation engineer and automation framework to execute the validation operatively.

To ensure the validation effectiveness of reducing redundancy of the test cases, a deep learning method and greedy approach to selecting the best combination of test cases that have complete validation coverage will be proposed. The reduction will impact significantly the project cost in terms of resources catered.

These results suggested that using a hybrid approach will expedite the progress by approximately 20%. From the organization's point of view, this proficiency results in cost savings and quality products for the consumer to use.

2 Biography

The primary author is Loo Willam, a validation engineer who works at Intel Penang. His technical expertise is in Validation Methodology, pre-silicon, and post-silicon validation. He did involve validation field for about 6 years from pre-silicon, post-silicon, and software to platform validation. He graduated with a bachelor's degree in Computer Science, majoring in Software Engineering in the year 2014, and currently, he is undertaking a Master of Software Engineering at Universiti Malaysia Pahang (UMP).

The co-author is Yip Kin Choy (KC Yip), a principle engineer working at Intel Penang. <Fill in when available>

3 Introduction

Exhaustive testing and validation on a broad market IoT (Internet of Things) product are not effective from coverage and time to market perspective what-more when this is performed in manual test configuration where development work and continuous integrations are continuously evolved in regular cadence. In the current mode of operation, we encounter a backlog of validation tasks up to as much as three weeks while it was all supposed to be completed in a week. Hence, this paper focuses on research work to develop a smart algorithm to prioritize test cases according to the incoming changes. It allows validation engineers and the automation community to stage the execution more operatively and predictably.

To deliver an effective, lean, and agile approach to validation coverage, a deep learning methodology, recurrent neural network along with the approach named “greedy” approach is being deployed. This is done to reduce the number of test cases and in doing so, selectively pick and choose the best possible combination of test cases that should suffice the incoming requirements. In IoT test qualification, it's critical that the validation encompasses all possible usage scenarios and in that mimicking the customers' end applications for the intended use cases model. In this scenario, running an NVR (network video recorder) application would demand execution of over n possible combinations of different pipelines and workloads to accommodate the utilization of a digitized surveillance system coupled with secured in-band manageability features. The deployment of deep learning helps to predetermine the selection of test cases based on a set of inputs corresponding to the intended validation coverage. This in turn significantly improves the execution duration, reduces cost in resource allocation, and shortens the overall product qualification cycles.

The outcomes of such a hybrid approach have broken the traditional approach of validation and broken the norm of constant and repetitive execution by introducing smarter solutions in determining the right set of test cases according to the defined requirement. This has expedited the completion by approximately 60% by fully utilizing the software-defined "greedy" approach while delivering top-quality focused validation according to the customer's requirement and use case model.

The hybrid approach comprises deep learning on self-learning on the git log from the repository. We will perform data acquisition and pre-processing of the commit id, author name, and commit date. On the training dataset side, we will put a label on the commit messages for numerous sizes of input. When we are working on the live action, after we obtained the predicted output, we have a query out from the database, and we need to filter out the relevant test cases. To optimize the test coverage, a greedy approach will implement to prioritize the test cases for the full test coverage. The greedy approach is the method that seeks the minimum test cases which will cover several requirements. Hence, with the hybrid approach, we can reach the objective of test case prioritization.

4 Literature Review

This literature review will be divided into a few subcategories, which are test case prioritization, test coverage, and deep learning on software validation-related works.

4.1 Test Case Prioritization

Yanshan et al [1] mentioned that the behavior pattern of the test will influence the prioritization of test cases in the group of test cases in the pool. Other than that, they realized that the additional effort by applying the greedy method to eliminate the duplication of the test cases, indirectly will augment the

execution process in the manual execution, continuous integration, and content development (CICD), furthermore with software quality gating process.

Besides that, they are using deep neural network models which include a maximum of ten hidden layers and four different types of adversarial attacks, examples are FGSM, JSMA, decision-based Gaussian Noise, and Uniform Noise to generate different sets of the test case pools. In the set of neurons $N = \{n_1, n_2, \dots, n_p\}$ and the test set $T = \{t_1, t_2, \dots, t_q\}$. Given the number of neurons of $n_i \in N$ and the number of the test set of $t_i \in T$, the activation function will fall into the Boolean state and its function will be $af(n_i, t_i)$. Here is the state for the activation function, where t is the limit to determine the states

$$af(n_i, t_i) = \begin{cases} 1, & if\ af(n_i, t_i) > t \\ 0, & otherwise \end{cases}$$

The neurons' attitude for the test $t_j \in T$ is the array. Following is the expression of the definition, where p is the Boolean state of the pth neuron.

$$B(t_i) = \left[af(n_1, t_i), af(n_2, t_i), \dots, af\left(n_p, t_i\right)\right]$$

From the test set T, the behavior pattern, BP can conclude as a mean array:

$$BP(T) = \frac{\sum_{t_i \in T} B(t_i)}{|T|}$$

To calculate the distance between each test data, for example, T_1 and T_2, it is measured by L1 norm distance:

$$distance = \sum_{i=i}^{p} |bp_i(T_1) - bp_i(T_2)|$$

From the result outcomes in terms of the distance in their experiments, we can observe that the test set with the closest distance will form a cluster but some of it will fall in the intersection area between clusters. The combination of the Greedy approach will further enhance the test case prioritization from the cluster.

Aizaz et al. [2] discussed that previous researchers had coded one test case prioritization using an automatic history-based approach where it was able to detect as many regression faults as possible and fit it into the execution phase for a specific duration.

In the test suite with the given test cases $T_S = \{t_1, t_2, \dots, t_n\}$ and *ExecutionStatus(ES)* of a test case t_i:

$$ES_{(i,j)} = \begin{cases} 1 & if\ t_i\ failed\ cycle\ j \\ 0 & if\ t_i\ passed\ cycle\ j \\ -1 & if\ t_i\ not\ execute\ at\ cycle\ j \end{cases}$$

The historical test case execution result for previous m cycles, p is the calculated test priority for the upcoming $m + 1$ cycle, d will be the average execution time during the m cycle.

$$HistoryTestData = \sum_{j \in 1..m} \frac{d(t_{(i,j)})}{m}$$

In the Deep Order approach, the test prioritization uses history test data execution status and test duration as the baseline. To optimize the priority, Deep Order implements a dual-objective function:

$$g(\max(f), \max(|T|))$$

The main aim is to make the best use of the fault detection ability of the test suite, and the subsequent aim is to fully utilize the number of executed tests in the given timeframe.

To reflect the higher impact of most failures, a particular test case priority value $p(t_i)$ is gauged as follows:

$$p(t_i) = \sum_{j \in 1..m} w_j \times \max(ES_{(i,j)}, 0)$$

The annotation of this equation is

- w_j is allocated weight in the range of Boolean value to every cycle j, for instance, $\sum_j w_j = 1$.
- m is the number of past cycles, $p(t_i) \in (0, 1)$
- $\max(ES_{(i,j)}, 0)$ used to eliminate no-run test where $ES_{(i,j)} = -1$.

In the deep neural network (DNN), the activation function is using Mish's activation function in the hidden layers of the neural network instead of a normal activation function like ReLU, and this new activation function is intended to overcome its flaws of it.

$$f(x) = x \times \tanh\big(softplus(x)\big) = x \times \tanh(\ln(1 + e^x))$$

In Reinforcement Learning for Test Case Prioritisation paper [3], we are clearly understanding that test prioritization depends on the number of cycles, feature records, and the optimal ranking. Aizaz et al. [2] shared the same thought that the elements above have the closest relationship to the test case prioritization. At the optimum ranking, they derive the ranking function of $n!$ distinct sequences are given n test cases,

$$\forall t_1, t_2 \in T,$$
$$idx(s_0, t_1) < idx(s_0, t_2) \leftrightarrow t_{1 \cdot v} > t_{2 \cdot v} \; or$$
$$(t_{1 \cdot v} = t_{2 \cdot v} \; and \; t_{1 \cdot e} \leq t_{2 \cdot e})$$

4.2 Test Coverage

There are a few solutions to solve the test coverage issue, the most efficient way is by maximizing the requirement coverage [4] when we are performing our test plan. In the validation world, individual test cases would be able to cover more requirements and if it does cover the critical requirement, that test case will be shortlisted and marked as the golden test case.

$$fit_{req}(TS) = \sum_{i=1}^{|TS|} |\{ Req(tc_i) \mid tc_i \in TS \}|$$

Pairwise testing or t-way testing [5] is the method where we specified at least a pair of test requirements as our exhaustive pair and other associated test requirements are free to match with it and duplicate pairs will be eliminated from the table. With the implementation of the greedy approach and t-way testing, we are ensuring that combinational testing will have complete coverages with the most minimal test cases.

4.3 Related Works

In the year 2021 at IEEE International Conference on Software Testing, Verification and Validation Workshop (ICSTW), Eran et. al. [6] proposed the QUADRANT approach. With their designed system architecture as shown below, this approach helps for automated test case generation erstwhile the whole kit and caboodle perfectly at software fault prediction.

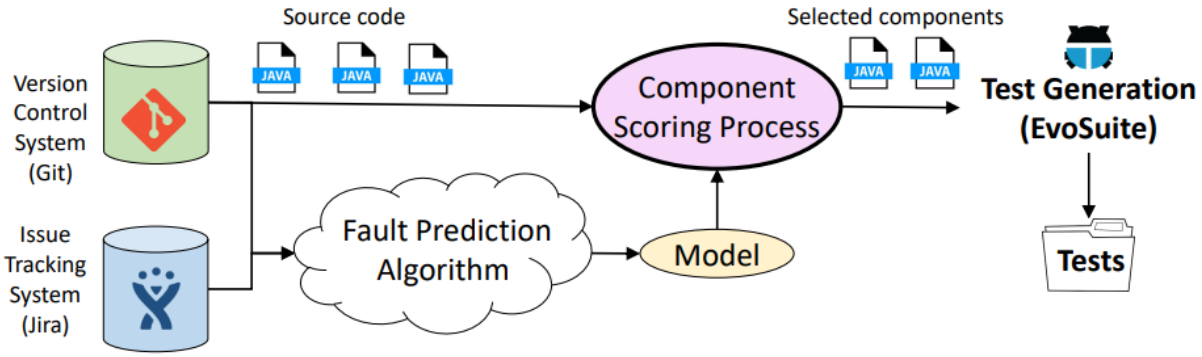


Figure 1: System Architecture with QUADRANT approach

Software test validation does involve manual execution and continuous integration (CI). Test cases will be residing in various repositories including Git, and Tortoise SVN. With incorporation into the bug sighting database, deep learning will take these as input so that they will be fed into the component scoring process. The outcome will be in the form of the test suite.

There are few heuristic ways to contrivance it. The component scoring process will list as follows:

a. Scoring with Fault Prediction (FP)

b. Scoring with Lines of Codes (LOC)

c. Scoring combination in FP and LOC. The formula for this combination is below.

$$FP_{\alpha}(C) = \alpha \cdot FP(C) + (1 - \alpha) \cdot LOC(C)$$

Indicates that α is the parameter in [0, 1], component scoring function, C of score $FP_{\alpha}(C)$.

In the research carried out by Kai et. al., [7] they are using a deep neural network (DNN) as the tool for performing the test case prioritization. Generally, in the concept used by YanShen et. al. [1], the key difference between them is the percentage of fault detected.

$$APFD = 1 - \frac{\sum_{i=1}^{m} w_i}{n \times m} + \frac{1}{2n}$$

Given that the test suite denoted by T with n number of test cases, m errors can be detected in the error set of F. Assumed that w_i be the number of test cases required to execute for i-th of errors found. The range of metrics will be in the range of 0 to 1.

4.4 Deep Learning Related Works in Software Test Validation

In the past section, we performed the analysis of fellow researchers with their published papers. Some benefits which can be adopted from it. On the other hand, they are a few points that have room for improvement, just that we are not planning for implementation in this research. There will be the pros and cons discussion as follows: -

With the combination of recurrent neural network (RNN) and deep neural network (DNN) methods, RNN will aid in text classification obtained from the git repository and provides the category that the git commit belongs to. Through this method, DNN plays an important role in test case selection and prioritization.

The greedy approach implemented in DeepOrder [2] helps in test case redundancy. With the redundancy and yet maximize the requirement coverage takes place, for example, 1000 test cases in the pool, with this approach we can reduce and consolidate up to 70% to 85% reduction. Test case prioritization incorporated with the greedy approach will rank the closely related and the highest coverage test cases to be recommended for the automation framework and validation engineers in software quality gating and test case execution.

In combinational t-way testing, we know that we are capable to perform exhaustive testing with lesser pair of test cases and selective pair of test cases in the test case pool. Besides that, by combining a few test redundancies methods, we can suggest to the validation engineer or the automation framework to execute their test suite and the time taken will be greatly reduced.

5 Proposed Test Case Prioritization Model

This section will explain the structure of the overview of the proposed test case prioritization model.

Figure 2 demonstrates the proposed design and the methodology of this research to solve our issues in our software test validation at Intel Malaysia. Fundamentally, this methodology will divide into three portions, proposed recurrent neural network (RNN) on extracting features from git log, data acquisition and pre-processing, and test case prioritization based on the type of test case.

In addition, this model design is divided into two phases, the training phase, and the testing phase. In the training phase, we will ensure that our design model can suggest a list of the test cases with high similarities to our expected list of test cases by analyzing manually. After several iterations of model training, we will have a certain level of confidentiality to proceed to the testing phase. In the testing phase, we will obtain a live log from the git repository and an updated test case from the database. The outcomes will pass to validation engineers and automation framework via email and triggers execution respectively. Figure 3 will provide the context of the flow for the proposed approach in Software Validation in Intel Malaysia.

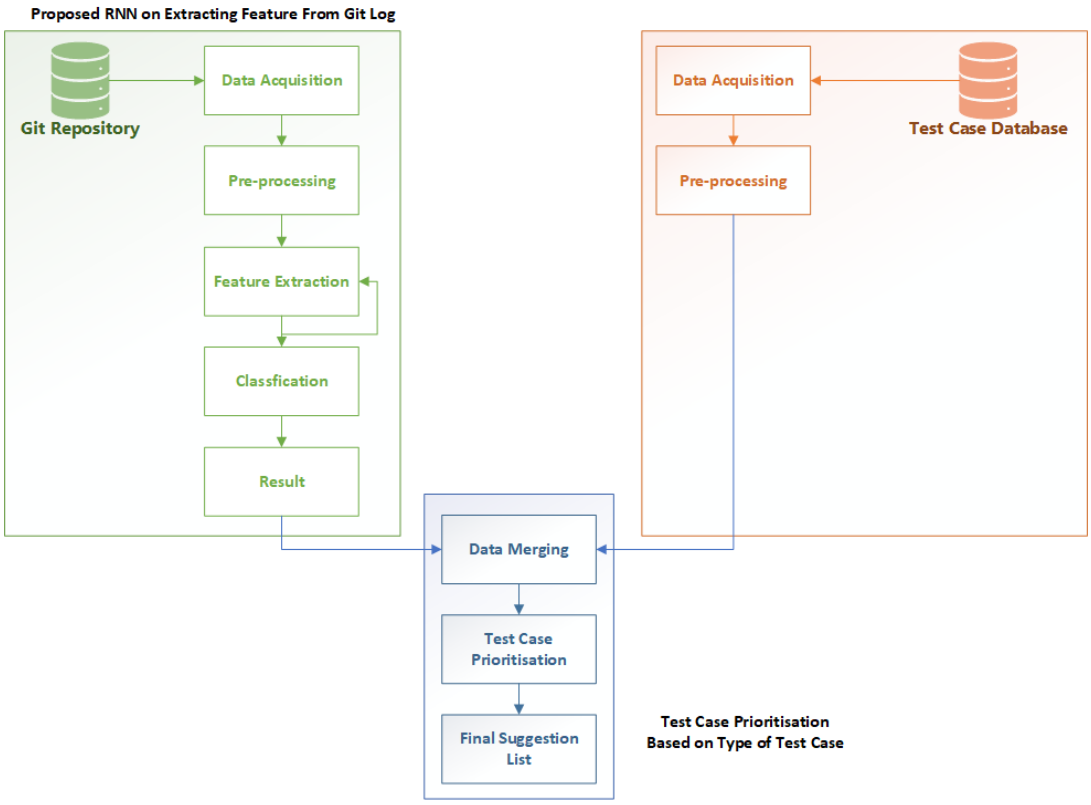


Figure 2: Proposed Approach for Software Test Validation in Intel Malaysia

```
BEGIN:

    // Predict git commit category
    Input : Obtain git log
    FOR i=0 in range total input:
        Get commit message
        Remove stopwords
        Feed into RNN model
        Get predicted output
    ENDFOR

    // Obtain test cases from test case database
    Query from database
    Save into variable A

    Select relevent test cases based on predicted output

    FOR j=0 in relevent test case size
        Search essential requirement
        Map to test case
        Search requirement covered in particular test case
        Save info in variable suggested_test_case
    ENDFOR

END
```

Figure 3: Pseudocode for Proposed Approach

5.1 Proposed RNN on Extracting Feature from Git Log

In this sub-section, we will discuss in deep about the processes involved with the recurrent neural network as shown in Figure 4, the main reason that we planned to implement this neural network due to our input data in text format.

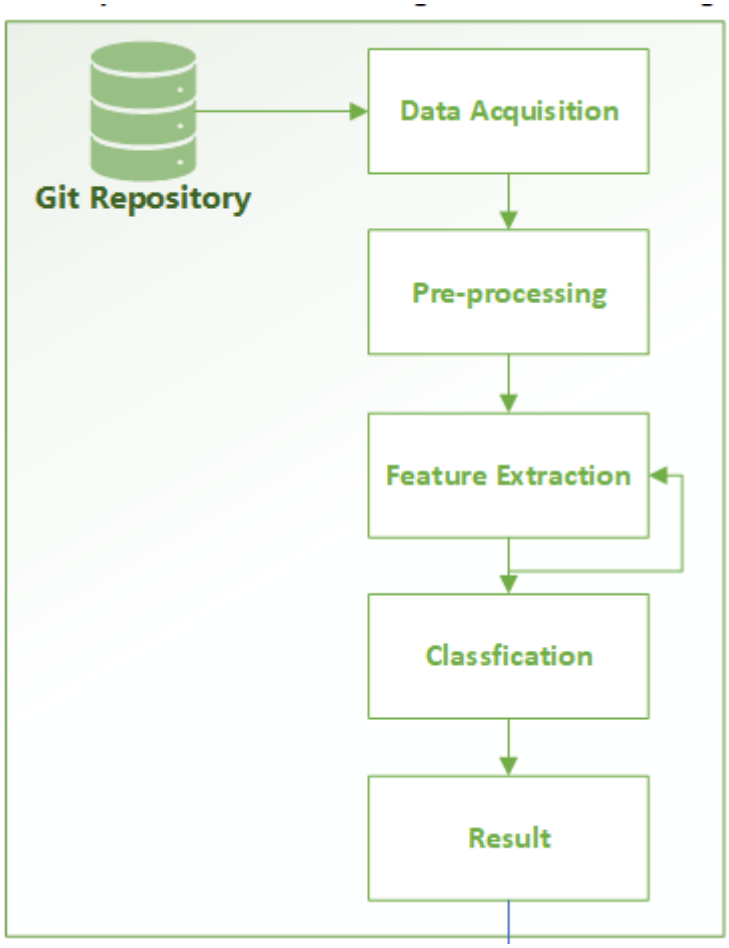


Figure 4: Proposed RNN Framework

5.1.1 Data Acquisition Phase

Our data is coming from the git repository. The git log consists of the commit hash of the commit, branch header, branch tag information, author of the commit, date, and the commit messages.

The prerequisite is we need to clone a branch from a repository. Inside the git folder, we collected the log and exported it into the text file. An example of the log will present in Figure 5. In a real-case scenario, every commit will trigger the process. Next, we will proceed to the data pre-processing phase.

```
commit fb488a8b8523fa544a8b71c6ab700fe72c0087b7 (HEAD, tag: <branch tag>, <branch_header>)
Author: Authour_name <author_email@intel.com>
Date:   Sat Feb 12 11:21:23 2022 -0700

    fixing GPIO in value

commit fb488a8b8523fa544a8b71c6ab700fe72c0087b7
Author: Authour_name <author_email@intel.com>
Date:   Sat Feb 12 11:21:23 2022 -0700

    UART Tx internal loopback content enabling
```

Figure 5: Sample Log File from Git

5.1.2 Data Pre-processing Phase

In this section, after we obtained the log file, it will parse over to this phase. At this phase, we will perform the metadata removal, like commit hash, author name and email, header tag and branch name as well as the date of the commit.

Once this process had been completed, we will acquire the string of the commit message, we need to consume the string into the sentence filtering process where in this case, we just filtered out the punctuations if any. Thus, after filtration is completed, we must serialize every word into an array as an input neuron for the input layer.

5.1.3 Feature Extraction Phase and Classification Phase

In this section, we will apply the recurrent neural network (RNN) to extract the features of the sentence array.

The RNN formula is formulated as follows where h_t represents the current state at period t, fw represents the function with the parameter w, h_{t-1} shows the previous state of the RNN, and x_t represents the input vector at t timestamp.

$$h_t = fw(h_{t-1}, x_t)$$

At the end of this phase, we will proceed to the next phase, the classification phase. In this phase, we need to undergo a dense approach to produce the correct classification.

In a conclusion, Figure 6 illustrates the overall structure of the RNN from the input phase to the classification phase with examples.

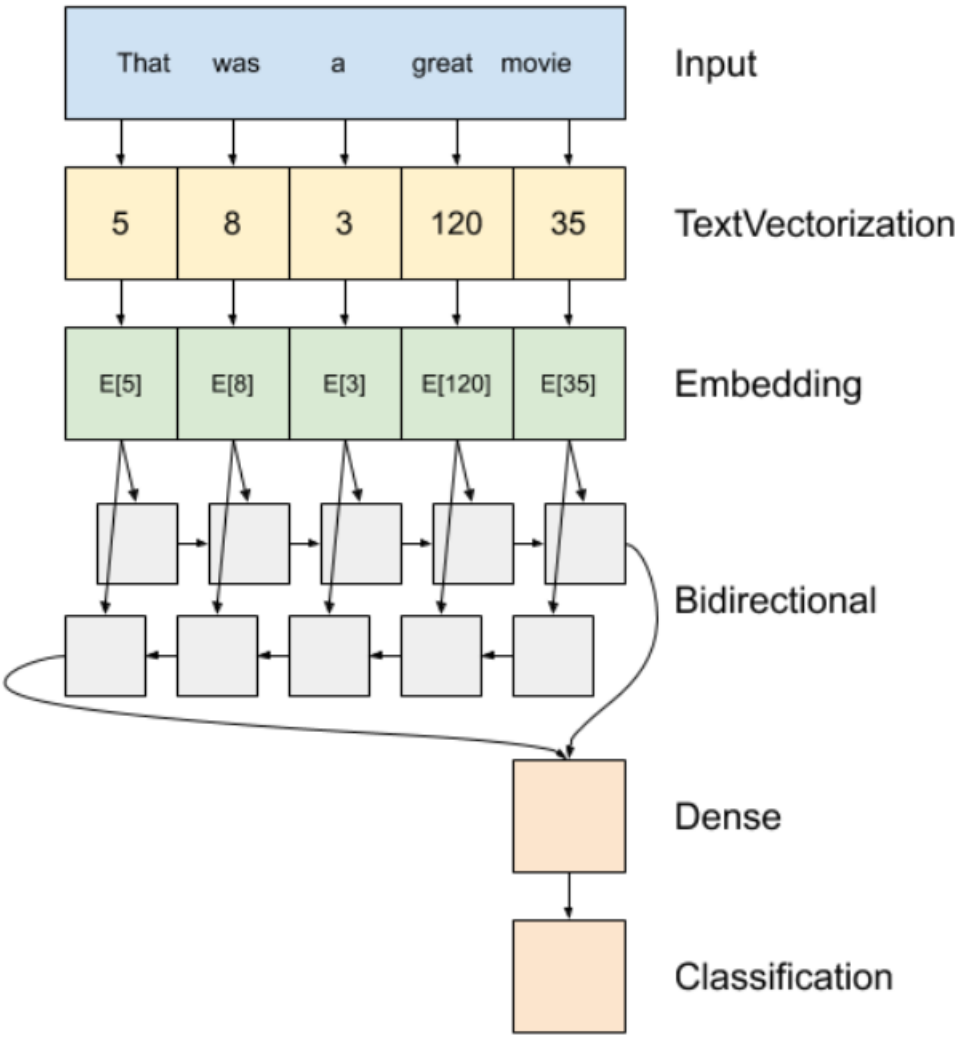


Figure 6: Overall Example Structure of the RNN in detail

5.2 Data Merging Phase

After querying all test cases from the database, we performed the data merging. In this phase, we will filter out the test case pool based on the necessary category. Once this process is completed, we recommended proceeding to the next phase. The equation below shows the notation on the method that we used to filter out the test cases pool.

$$FTC_i = TCP_i \cap C_i$$

In this equation, FTC represents filtered test cases and TCP represents the test case pool, C represents the category label we obtained and i represents the iteration in the *for* a loop.

5.3 Test Case Prioritization

In this section, we will reduce and prioritize the test cases based on the required coverage. At the end of this section, suggested test cases based on the required coverage will be presented and forwarded to the automation framework and validation engineers kickstart their execution.

For the test case prioritization, we will implement the test case reduction method, which is a Greedy algorithm to remove the redundancy of the test cases and prioritize the test cases by maximizing the coverage of the requirement. From the pseudocode in Figure 3, we are going to drill down the process that we will implement. Equation following is the method that is going to execute.

$$TC_i = \sum_{i=0}^{n} \left(\sum_{j=0}^{p} REQ_{(i,j)} \geq 1 \right) \leq \frac{1}{2} n$$

From this equation, TC shows that the test cases, given that $TC = \{TC_0, TC_1, \dots, TC_{n-2}, TC_{n-1}\}$, REQ represent the requirement row, given that $REQ = \{REQ_0, REQ_1, \dots, REQ_{p-2}, REQ_{p-1}\}$, i and j are the coordinates of the test cases versus to the requirements and n and p are the total row and column numbers in the test requirement matrix.

6 Result and Discussion

This section will focus on evaluating the performance of the developed algorithm in the software validation process. In the dataset preparation due to the company's private and confidential data in the git log, the test case requirement and the respective test cases definition in the portal, an open-source dataset from Kaggle, which is the customer complaints spreadsheet, and the sample draft of the test requirements versus to the test cases in the CSV format.

6.1 Text Analysis Experiment

The customer complaints spreadsheet illustrated in Figure 7, contains 1,324,194 complaints, and here is the sample data snapshot captured from the CSV file. From the screenshot, the product column is the final decision of the complaints that the customer had made.

Date recei	Product	Sub-produ	Issue	Sub-issue	Consumer	Company	Company	State	ZIP code	Tags	Consumer	Submitted	Date sent t	Company r	Timely res	Consumer	Complaint ID
########	Mortgage	Other mor	Loan modification,collection,foreclosure				M&T BANI	MI	48382		N/A	Referral	03/17/201	Closed wit	Yes	No	759217
########	Credit reporting		Incorrect i	Account st	I have out	Company	TRANSUNI	AL	352XX		Consent pr	Web	########	Closed wit	Yes	No	2141773
10/17/201	Consumer	Vehicle loa	Managing the loan or	I purchased a new car			CITIZENS F	PA	177XX	Older Ame	Consent pr	Web	10/20/201	Closed wit	Yes	No	2163100
6/8/2014	Credit card		Bankruptcy				AMERICAN	ID	83854	Older Ame	N/A	Web	########	Closed wit	Yes	Yes	885638

Figure 7: Sample Snapshot from Customer Complaint Spreadsheet

In the list of customer complaints, there are eighteen categories, and the test program will query out the statements of the complaint and remove the stop words and punctuation. After that, it will get padding and populated into the matrix for the LSTM algorithm to consume the matrix. The LSTM algorithms will analyze the customer complaints and it will move to the dense layer which will decide the predicted output.

Once the predicted output is obtained, the result will be parsed to the database for querying out the necessary test cases and tabulated in the form of the list.

6.2 Test Case Experiment

When section 6.1 is completed and the test case list had been exported in the text form, it concluded as a requirement test matrix.

In the requirement test matrix (RTM) as shown in Figure 8, a few sets of matrices had been prepared with numerous requirements and test cases listed in there. It does represent the actual test cases that going to execute by the validation engineers in Intel Malaysia. Due to the company's private and confidential information, this research is unable to use the real data for validation, and hence, draft copies of the RTM have been used for demonstration purposes.

```
t0=1:1:0:0:0:0:0:0:0:0:0:0:0:0:0:1
t1=0:0:0:0:0:0:0:0:0:1:0:0:0:0:0:0
t2=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t3=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t4=0:0:0:0:0:0:1:0:0:0:0:0:0:0:0:0
t5=0:0:0:0:0:0:0:0:0:1:0:0:0:0:1:0
t6=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t7=0:0:0:0:0:0:1:0:0:0:0:0:0:0:0:0
t8=0:0:0:0:0:0:0:0:0:1:0:0:0:0:0:0
t9=0:0:0:1:0:0:0:0:0:0:0:0:0:0:0:0
t10=0:0:0:0:0:1:1:0:0:0:0:0:0:1:0:0:0
t11=0:0:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t12=0:0:0:1:0:0:0:1:0:0:1:0:0:0:0:0:0
t13=0:1:0:0:0:0:0:0:0:0:0:0:0:0:1:0
t14=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t15=0:1:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
t16=1:0:0:0:0:0:0:0:1:1:0:0:0:0:0:0
t17=0:0:0:0:0:0:0:1:0:0:0:0:1:0:1:0:0
t18=0:0:0:0:1:0:0:0:0:0:0:0:0:0:0:0:0
t19=0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
```

Figure 8: Test Case in The Form of List

6.3 Result Presentation

This section will present the result obtained from the text analysis from section 6.1 and the accuracy with the respective time taken from section 6.2. Therefore, these results will separate into two sub-categories, text analysis, and test case prioritization respectively.

6.3.1 Text Analysis Portion

From the outcome in section 6.1, the ideal accuracy for the test data is approximately 88% as the overfitting scale had set to 0.2 meanwhile the ideal loss accuracy for the test data falls at about 32%. Time taken for the training process is about 40 minutes as the epochs are set to 8 and the batch_size of the LSTM algorithms is set to 128 as the default recommended value needs 7 minutes and 30 seconds

for each epoch. The overall train and test data had been plotted in the form of the graph as shown in Figure 9 and Figure 10.

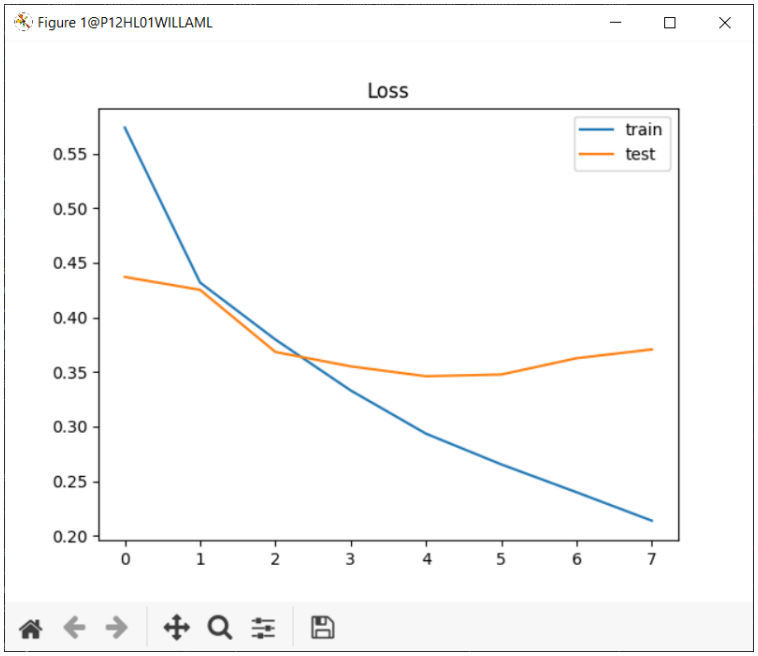
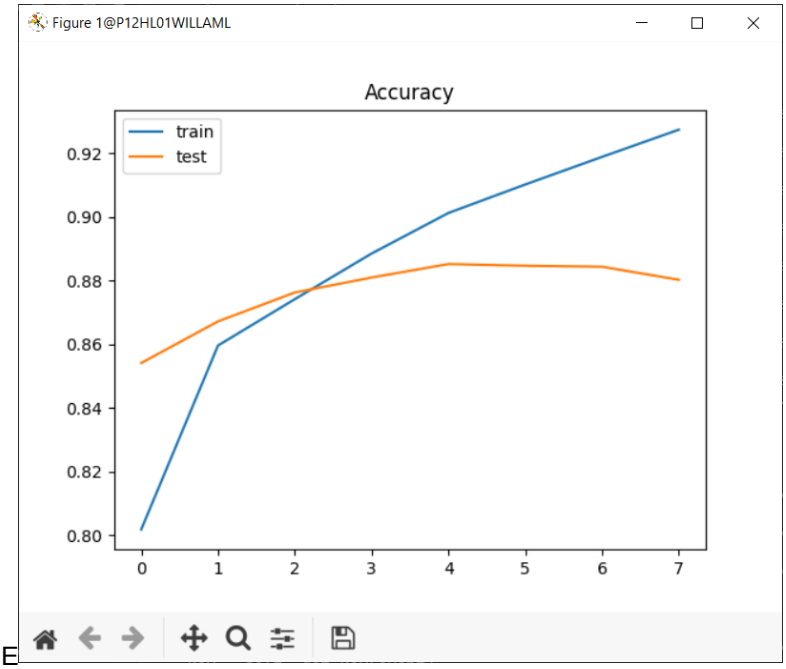


Figure 9: Loss Between Train and Test Data Training



E

Figure 10: Accuracy Between Train and Test Data Training

6.3.2 Test Case Prioritisation Portion

From the output in section 4.3, two sets of requirement test matrix (RTM) which comprises the 30 test cases and 51 test cases respectively with the fixed number of the requirements, from $REQ_0, REQ_1, REQ_2, \ldots, REQ_{n-1}$ where $n = 16$. Detailed information will tabulate in Table 1.

Table 1: Detailed Information of the Test Case Reduction

Requirement Test Matrix (RTM) Number	RTM #1	RTM #2
Original Test Case Number	30	51
Reduced Test Case Number	12	9
The total reduction of test cases in percentage (%)	60.000	82.353
Time Taken (seconds)	0.00081	0.00087

6.4 Discussion

This section will perform the discussion and findings that the result collected from section 6.3. The discussion will separate into two sub-sections which encompass the text analysis and the test case prioritization.

6.4.1 Text Analysis

In this sub-section, a detailed discussion will be held on the computing efficiency, algorithm, and the result comparison with the previous researchers that had completed the Literature Review section.

In this research, python is used as the programming language compared to C programming which can execute the code by fully utilizing the threads in the compute processing unit (CPU). Besides that, the LSTM algorithm in the recurrent neural network (RNN) will consume more computing power from the multithreading mechanism. Due to the limitation of python programming, each epoch is taking about 5 minutes and 20 seconds with the batch size set to 128. Meanwhile, the default value for the batch size is 64 where it will take about 7 minutes to run per epoch.

From the algorithm perspective, this research uses the Long Short-Term Memory (LSTM), an advanced version of the prevailing Recurrent Neural Network (RNN) with the formula mentioned

$$h_t = fw(h_{t-1}, x_t)$$

In LSTM, it is easier to memorize the past input in the memory. Overall, LSTM is well-suited for cataloging, dealing out, and envisaging time series in the provided time lags with the unforeseen timeframe. In the context of results obtained in Figure 9 and Figure 10, it clearly shows that when the

relationship between the loss and the accuracy is reverse proportional to each other. Due to the limitation mentioned in section 6.1, the graph presented was not as ideal as expected.

From the result comparison perspective, the loss and accuracy are highly dependent on the dropout and the recurrent dropout value, which falls between 0 to 1 in the LSTM. The purpose of the dropout and the recurrent dropout is to prevent the overfitting situation from happening. In the current research prototype, the value used is 0.2 respectively for both variables. This value is the most suitable value to overcome this situation. For the training data, we are using the categorical cross-entropy loss function and the activation function wise, SoftMax activation energy was chosen to have a clear graph that consists of the value from value 0 to 1.

6.4.2 Text Analysis

This subsection will converse about the result obtained and compare it with the method used by the researchers deliberated in Literature Review section.

In this research for test case prioritization, by using the formula stated as follows,

$$TC_i = \sum_{i=0}^{n} \left(\sum_{j=0}^{p} REQ_{(i,j)} \geq 1 \right) \leq \frac{1}{2} n$$

and the result obtained in sub-section 6.3.1, the test case prioritized is based on the coverage of the requirement for the specific test case. In the greedy approach, the last occurrence of the requirement being covered will be prioritized. Thus, the redundant test cases for the overlapped requirement will be consolidated by the test case which covers the greatest number of requirements. As JPCT performed better than the SPCT [6] carried out by the researchers from Japan and compared the result with the JPCT and the proposed enhanced Greedy algorithm proposed in this research, JPCT can cover 73% of the coverage while the proposed enhanced version of Greedy algorithm covers 100% of the requirements listed with the minimum test cases needed.

7 Conclusion

The key objective of this chapter is to epitomize the verdicts of the recurrent neural network (RNN) on the text analysis of the log file in the form of the CSV format and the enriched Greedy algorithm whereby this approach does help the performance of the execution at the end of the day to accomplish the goals and get to the bottom of problems avowed in the problem statements.

Moreover, this research work also provides a detailed analysis of the efficiency of the algorithm that impacted the validation execution progress. The advanced stratagem is aided to ensure the robustness of the set of rules implemented towards the validation coverage and LSTM model training. The objectives and problem statements of this research can be achieved and solved. This project had contented the purposes as follows:

1. To evaluate and determine a deep learning algorithm that is suitable for the software validation process in Intel Malaysia.

2. To develop the determined deep learning algorithm for the software validation process in Intel Malaysia.
3. To evaluate the performance of the developed algorithm in the software validation process.

7.1 Research Contribution

7.1.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is part of the recurrent neural network (RNN) for the text and character analysis for the result prediction. This technique can learn the knowledge provided from the input and if the previous input is not relevant to it, the algorithm will not memorize it and vice versa.

By denoting to Deep Learning Model article [8], they will be pre-processing their plain input and removing the stop words plus punctuation, data vectorization, building models from the training sets, predicting the test data then populating the accuracy. Aside from that, researchers presented the data or results obtained using long and short sentences as input are very close to the results obtained in this research with the limited number of epochs.

7.1.2 Enhanced Greedy Algorithm

Typical Greedy algorithm help in optimizing the coverage but the efficiency wise is not optimum. With the enhanced Greedy algorithm implemented in the previous chapter, it does speed up the execution speed yet had full coverage of the test cases concerning the requirements.

7.2 Limitations

This research has some limitations which will be jot down in the following points.

1. LSTM has will drag the performance of the CPU and GPU and it will affect the epoch cycle.
2. Greedy algorithm unable to perform the mix-and-match test cases prioritization while maximizing the requirement coverage.
3. Company's private and confidential git log and test requirement for the internal project is prohibited to use in this research. Hence, an open-source database, named customer complaints had been used in this research.\

7.3 Future Works

Several perfections can be suggested for the Validation Test Case Prioritisation Using Hybrid Approach for the future to boost the functionalities. The suggested enhancement is shown below:

1. LSTM can train with multiple epochs with better hardware and software requirements.
2. Advanced Greedy algorithm able to incorporate with the pairwise testing which will help in removing the test case redundancy that happened either from Greedy algorithm or pairwise testing.

8 References

[1] Yanshan Chen, Ziyuan Wang, Dong Wang, Yongming Yao, and Zhenyu Chen, Behaviour Pattern-Driven Test Case Selection for Deep Neural Networks, 2019, IEEE International Conference on Artificial Intelligence Testing (AITest) page 89, 90.

[2] Aizaz Sharif, Dusica Marijan, and Marius Liaaen, DeepOrder: Deep Learning for Test Case Prioritisation in Continuous Integration Testing, 2021, IEEE International Conference on Software Maintenance and Evolution (ICSME) page 525-534.

[3] Mojtaba Bagherzadeh, Nafiseh Kahani, and Lionel Briand, Reinforcement Learning for Test Case Prioritization, 2011, arXiv:2011.01834v [Online Journal]

[4] Remo Lachmann, Michael Felderer, Manuel Nieke, Sandro Schulze, Christoph Seidl, and Ina Schaefer, Multi-Objective Black-Box Test Case Selection for System Testing, 2017, GECCO 2017, pg1311-1318.

[5] Eran Hershkovich, Roni Stern, Rui Abreu, and Amir Elmishali, Prioritized Test Generation Guided by Software Fault Prediction, 2021, IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), page 218-225

[6] Yuper Lay Myint, Hironori Washizaki, Yoshiaki Fukazawa, Hideyuki Kanuka, and Hiroki Ohbayashi, Test case reduction based on the join condition in pairwise coverage-based database testing, 2018, IEEE International Conference on Software Testing, Verification and Validation Workshops, page 239-243

[7] Kai Zhang, Yongtai Zhang, Liwei Zhang, Hongyu Gao, Rongjie Yan, and Jun Yan, Neuron Activation Frequency Based Test Case Prioritization, 2020, International Symposium on Theoretical Aspects of Software Engineering (TASE), page 81-88.

[8] JingJing Cai, Jianping Li, Wei Li, Ji Wang, Deep Learning Model Used in Text Classification, 2018, 978-1-7281-1536-8/18/$31.00 ©2018 IEEE, page 123 – 126.