Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

A Framework for Continuous Regression and Integration Testing in IoT Systems based on Deep Learning and Search-based Techniques

Noha Medhat[1], Sherin M. Moussa[1], Nagwa L. Badr[1], and Mohamed F. Tolba[2], Senior Member, IEEE

[1]Department of Information Systems, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, 11566, Egypt
[2]Department of Scientific Computing, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, 11566, Egypt

Corresponding author: Sherin M. Moussa (sherinmoussa@cis.asu.edu.eg), Noha Medhat (e-mail: noha_medhat@cis.asu.edu.eg).

ABSTRACT Tremendous systems are rapidly evolving based on the trendy Internet of Things (IoT) in various domains. Different technologies are used for communication between the massive connected devices through all layers of the IoT system, causing many security and performance issues. Regression and integration testing are considered repeatedly, in which the vast costs and efforts associated with the frequent execution of these inflated test suites hinder the adequate testing of such systems. This necessitates the focus on exploring innovative scalable testing approaches for large test suites in IoT-based systems. In this paper, a scalable framework for continuous integration and regression testing in IoT-based systems (IoT-CIRTF) is proposed, based on IoT-related criteria for test case prioritization and selection. The framework utilizes search-based techniques to provide an optimized prioritized set of test cases to select from. The selection is based on a trained prediction model for IoT standard components using supervised deep learning algorithms to continuously ensure the overall reliability of IoT-based systems. The experiments are held on two GSM datasets. The experimental results achieved prioritization accuracy up to 90% and 92% for regression testing and integration testing respectively. This provides an enhanced and efficient framework for continuous testing of IoT-based systems, as per IoT-related criteria for the prioritization and selection purposes.

INDEX TERMS Deep learning, Integration testing, IoT, Regression testing, Test case prioritization, Test case selection, Search-based techniques

I. INTRODUCTION

The Internet of Things (IoT)-based systems are increasingly penetrating all business industries, in which the main characteristic of these systems is the heterogeneity of their components and technologies. Huge numbers of diverse independent devices, such as embedded objects, actuators and sensors, are continuously connected, leading to an enormous scale of components [1]. Such systems usually include a data generator layer, which aggregates data from all connected devices. Then through the network layer, different protocols and gateways are used for data transition to apply analytical processes, providing appropriate services to the targeted user's applications [2]. Testing is required at all stages, when everything is connected, and data are transmitted through networks [3]. Issues and threats are increasingly faced, especially regarding the privacy of critical personal data in order to maintain security and performance processes [4].

Thus, research studies on testing IoT-based systems are dramatically increasing, in which the changes on such large systems are endless. Continuous runtime regression and integration testing are repeatedly required to face the frequent dynamic integration of IoT components [5], as well as system change requests. A huge number of test cases (TCs) are usually generated over time that could be relevant or irrelevant to a specific new integration or change request. Hence, the need for minimizing the number of executed TCs is exponentially increasing to reduce testing costs and maximize efficiency.

Many limitations have been explored for testing IoT systems, such as the inability to handle the dynamicity of connected components [6], diversity of network protocols and technologies [7], real time complexity and scalability of systems that manage their connectivity [8]. In addition, the systems' interoperability should be frequently verified by checking the consistency of services provided by different sensors [1], as well as the transmitted data between receivers and suppliers should be verified against the confidentiality of networks and the safety of users' data [9].

The applicability of automated test suites generation through Model Based Testing (MBT) Techniques have been investigated, in which TCs can be constructed from a graph model, i.e. finite state machines, UML diagrams, etc. [10] to continuously generate test suites and execute automatically in order to check the coverage of newly added components [5]. However, MBT techniques lacked to effectively execute the generated test suites as per any new changes [11]. Thus, they do not support integration and regression testing, in which all generated test suites should be executed in such rapidly evolving systems [12]. Such critical limitation causes great time and cost consumption. Consequently, the selection and prioritization of test suites are vitally needed to execute the related TCs only during regression testing, avoiding redundancy and irrelevant TCs. On the other hand, Search Based Testing (SBT) techniques have been considered for TCs prioritization and reduction, such as Genetic Algorithms (GAs), Hill climbing, Ant Colony and Simulated Annealing (SA) techniques [8]. They proved lower performance and high time consumption, compared to MBT, when used for TCs generation [13]. Yet, SBT techniques have never been investigated for testing IoT-based systems.

In this paper, a scalable framework for prioritizing and selecting TCs in IoT-based systems (IoT-CIRTF) is introduced based on deep learning and SBT techniques. This framework is intended to support continuous integration testing and regression testing for IoT-based systems by providing an optimized self-adaptive prioritized set of TCs to select from to continuously ensure the overall reliability of these systems upon the addition or removal of their independent components. The SBT techniques are utilized to select the relevant TCs after the training phase of the proposed framework on the IoT system specifications using the Long Short-Term Memory (LSTM) classifier, which is a deep learning supervised prediction algorithm [26][27]. Deep learning algorithms have not been considered for testing IoT-based systems yet, in which IoT-CIRTF uses them to classify the given TCs to the targeted IoT components. The matched TCs are then executed with respect to the change requests in the requirements -for regression testing - or to the newly added modules of the specific IoT system -for integration testing.

II. RELATED WORK

Several testing approaches have been directed to IoT-based systems, where MBT techniques were dominant [29][30]. Some MBT limitations were addressed, such as the time consumed for TCs generation and TCs redundancy when reconfigurations or changes occur, and the lack of TCs prioritization or selection approaches to handle continuous integration and regression testing in IoT-based systems [31]. Some studies lacked the presence of a real-world case study, or the scalability of testing approaches to fit IoT-based systems. Another challenge was the process automation, in which the tracking and testing of runtime systems were very exhaustive in terms of time and cost [10]. The following subsections present the main approaches introduced in the literature that consider testing in IoT systems.

A. TESTING TECHNIQUES IN IOT-BASED SYSTEMS

Most of the MBT techniques used for TCs generation in IoT systems from modeled systems were based on reformatting the model to an XML code, then constructing TCs from that generated code. In [32], a runtime verification of IoT systems was achieved using an MBT tool, which generated a sequence diagram and TCs statements during runtime with an event processing language for complex diagrams. This approach was developed and tested using Constrained Application Protocol (COAP) only for data transmission between devices, which made it unavailable for any other protocols. [33] introduced an approach for runtime testing in component-based systems. By applying TCs selection and MBT techniques, it selected TCs from previously created and new ones as per changes using UPPAAL tool. Yet, it was not proven experimentally how efficient it worked with IoT systems.

In [34], a study for some testing tools for IoT systems was presented, like MBTAAS, IOTSim and MAMMotH for TCs generation from models. A comparison was made to show what to use when considering different IoT systems' layers and the testing types to cover according to the systems' complexity. The absence of reliable evaluation, in terms of accuracy and efficiency, was the main limitation.

In [35], the VDM++ formal specification tool was investigated for integration testing in distributed systems. It used sequence diagrams to generate relevant TCs and overcome their explosive number by observing the system to decide whether a distribution of the integrated components would be required to test locally by manual testers.

Authors in [10], emphasized that there are still influential and unexplored areas for TCs effective selection, in which many tools lacked TCs prioritization, wasting huge time, cost and efforts. They recommended that there should be selection-based criteria to prioritize TCs of IoT-based systems, including all functional and non-functional requirements. The survey of [36], discussed that all current testing techniques for IoT systems lack integration testing handling.

B. TEST CASES PRIORITIZATION IN IOT SYSTEMS

Some metrics were proposed in [30] to measure the importance of TCs and decide the most valuable ones to prioritize and rerun for regression testing. The suggested metrics included measuring the faulted devices' locations in the area around using Newton's law to select TCs according to the nearest devices, in which TCs were generated as per the location of devices. This was dependent on the system's behavior with experienced testers, indicating that there was no standard generic approach to apply for IoT-based systems, as the metrics would differ from one system to another.

For the selection of relevant TCs at retesting phases, the Ant Colony Optimization algorithm was proposed in [37] integrated with Bayesian approaches to calculate the probability of each TC. TCs appearing the most in detected failures had a higher priority. No experiments were conducted to evaluate the proposed approach. Furthermore, it was not mandatory to select the same set of TCs due to any changes, as it should have considered another metrics, such as the propagation of the changed features or components.

C. RESEARCH GAP AND LIMITATIONS IN IOT TESTING

Most of the testing-related studies in IoT-based systems have shown a real focus on TCs generation using MBT techniques, supporting TCs generation for newly added components. However, this raises a crucial concern regarding their applicability for continuous integration and regression testing in such evolving IoT-based systems, provided the increasing number of associated TCs. Thus, further investigations are needed to prioritize and select TCs in order to minimize the time and efforts required for the repeated integration and regression testing expected for IoT-based systems, allowing fast faults detection and localization with respect to new changes or components. The current literature lacks proven experimentations for efficient and reliable approaches to face the detected challenges.

III. THE IOT CONTINUOUS INTEGRATION AND REGRESSION TESTING FRAMEWORK (IOT-CIRTF)

In this study, an IoT-related framework for continuous integration and regression testing (IoT-CIRTF) is proposed based on a hybrid combination of deep learning and search-based techniques. Fig.1 presents the system architecture of the proposed framework. It consists of three main layers as described in the following subsections.

A. THE IOT COMPONENTS TRAINING LAYER

As the number of requirements in IoT systems and their associated test cases is increasing continuously, the cost and time needed to test such systems are directly proportional. This layer is responsible for generating a trained model for IoT components in order to learn the specifications of the IoT system and classify them according to standard IoT system components. The created model would provide the backbone to automate the classification, selection and prioritization of test cases throughout the framework, decreasing the overall time, efforts and cost of testing IoT systems. The input of this layer is the specifications of the IoT system to extract the standard components. These specifications describe the different and huge number of components and technologies connected in the IoT system, such as data gathering (i.e. RFID, Sensors, actuators, GPS), communication protocols (i.e. MQTT, HTTP, Wifi, Zigbee, WAN), cloud processing and user devices/applications. The layer consists of some modules as follows:

- *IoT specifications pre-processing*, in which the specifications are cleaned and reduced by omitting stop words and redundancy using NLP techniques.
- *IoT components features extraction*, which uses the LSTM deep learning classifier to analyze the cleaned reduced specifications, resulted from the previous process to extract the main features of the IoT system's specifications. The extracted features represent the selected words having high weights from the whole specifications, which are needed to classify the IoT TCs in the next layer effectively [14][15]. These features describe certain IoT standard components, which are the defined classes (i.e. user device, protocols and gateways, sensors and actuators, data processing). The LSTM classifier was proven to work effectively with long sequences [16]. It calculates the probability of each word to follow another, given the sequences of words with their weights.
- *IoT components training by classifier*, in which it works on the extracted features by the classifier based on LSTM algorithm [26][27]. Algorithm 1 presents the LSTM classifier, consisting of four layers/nodes.

The extracted features are used by the classifier in order to recognize the classification pattern to follow, so that the class labels that the IoT components belong to are determined [28]. The output is the IoT components trained model that is used in the next layers of the framework to classify the IoT TCs as per the IoT components.

In our IoT-CIRTF, LSTM classifier is modified to fit long sequences of the IoT requirements and is configured to work on four layers/nodes, in which LSTM functions run four times to increase the accuracy of learning, avoiding low accuracies when applying two LSTM layers/nodes [20]. The first layer uses the Sigmoid function to calculate the scalar value of the three main LSTM gates (i.e., forget, input and output gates), maintaining their values in [0, 1] range as shown in (1), (2), (3) [19]:

$$f_{(t)} = \sigma(w_{f,x}x_{(t)} + u_{f,h}h_{(t-1)}) + b_f \quad (1)$$

$$i_{(t)} = \sigma(w_{i,x}x_{(t)} + u_{i,h}h_{(t-1)}) + b_i \quad (2)$$

$$o_{(t)} = \sigma(w_{o,x}x_{(t)} + u_{o,h}h_{(t-1)}) + b_o \quad (3)$$

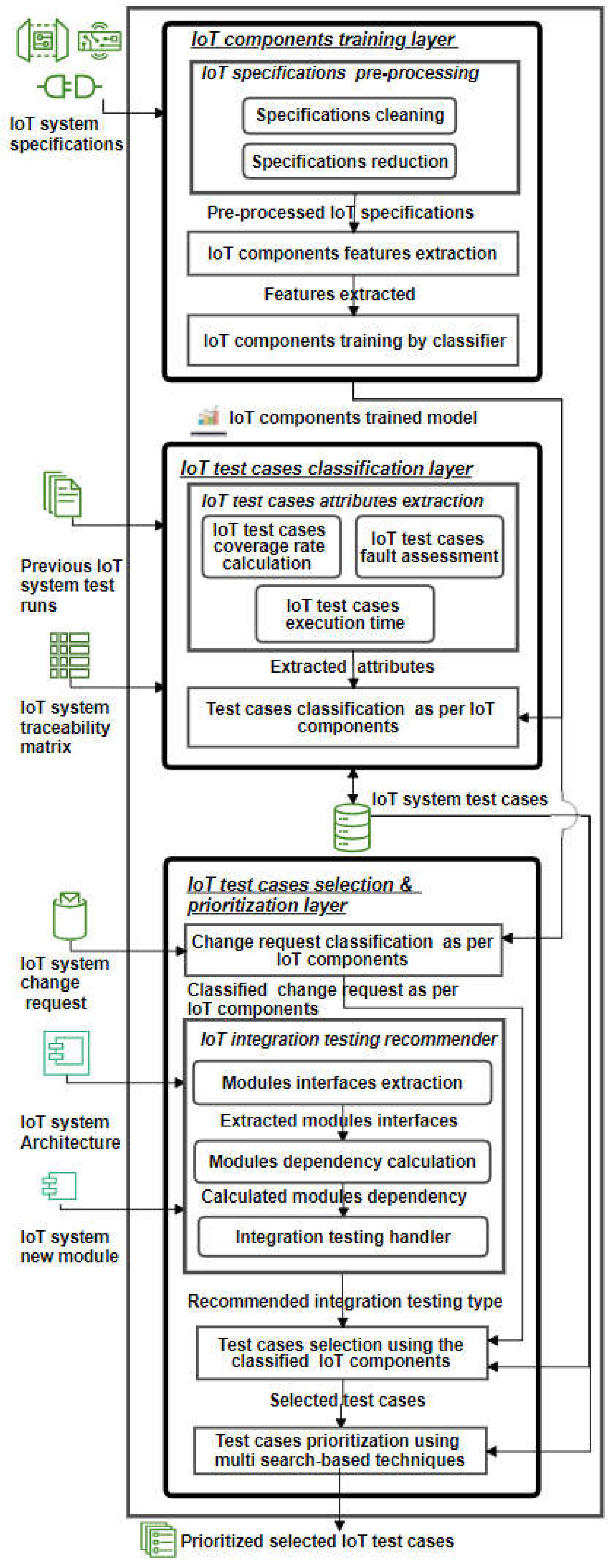

IoT system specifications
IoT components training layer
IoT specifications pre-processing
Specifications cleaning
Specifications reduction
Pre-processed IoT specifications
IoT components features extraction
Features extracted
IoT components training by classifier
IoT components trained model
IoT test cases classification layer
IoT test cases attributes extraction
IoT test cases coverage rate calculation
IoT test cases fault assessment
IoT test cases execution time
Previous IoT system test runs
IoT system traceability matrix
Extracted attributes
Test cases classification as per IoT components
IoT system test cases
IoT test cases selection & prioritization layer
IoT system change request
Change request classification as per IoT components
Classified change request as per IoT components
IoT integration testing recommender
IoT system Architecture
Modules interfaces extraction
Extracted modules interfaces
Modules dependency calculation
Calculated modules dependency
IoT system new module
Integration testing handler
Recommended integration testing type
Test cases selection using the classified IoT components
Selected test cases
Test cases prioritization using multi search-based techniques
Prioritized selected IoT test cases

FIGURE. 1: **The proposed IoT Continuous Integration and Regression Testing Framework (IoT-CIRTF).**

Algorithm 1 LSTM deep learning algorithm in IoT-CIRTF.

```
    Output: Learned classifier
1   Begin
2       Initialize sequence length (SL) to a large value
3       For i in range (0, SL)
4         If at the first node
5             C_R= Random() //Initialize cell vector (C)
              to a random value
6             H_R= Random() //Initialize hidden value
              (h) to a random value
7         Else
8             C_R=C_N and H_R=H_N
9         End if
10        Calculate the gates' activation functions using
          Sigmoid function.
11        Calculate the candidate values (CV) using
          Sigmoid function.
12        Calculate new cell (C_N) using CV,
          previous cell (C_R), forget gate (f) and input
          gate (i).
13        Calculate new hidden value (H_N) using
          output gate (o) and new cell (C_N) by CV.
14        Generate vector of possible outcome classes
          probabilities for output sequence using
          Softmax function.
15        Calculate the cross-entropy loss function.
16      End for //iterating sequences in training set
17  End // iterated for 4 layers/nodes of LSTM
```

Where $f_{(t)}$ is the forget gate that defines how many features with very low weights are going to be ignored, $x_{(t)}$ is the current feature/word in the IoT system's specifications sequences at time t. $i_{(t)}$ is the input gate that represents the input sequence of IoT system's features or words, $o_{(t)}$ is the output gate that defines the words/features to be added to the hidden gate $h_{(t)}$ for later use, which may be added to the sequence of extracted features as in (6). Hence, the first iteration of $h_{(t)}$ is initialized with a random value. $w_{f,x}$,$w_{i,x}$,$w_{o,x}$ and $u_{f,h}$,$u_{i,h}$, $u_{o,h}$ are the weights of$x_{(t)}$, b_f,b_i , b_o are the biases (initialized by 1), to be learned during the training of the classifier. The cell state represents the current input of features/words, as well as how many features to forget from the previous cell state.

Next, the new hidden value and the new cell state are calculated as shown in (4), (5) and (6) [19]:

$$cv_{(t)} = \sigma(w_{c,x}.x_{(t)} + u_{c,h}.h_{(t-1)} + b_c) \tag{4}$$

$$c_{(t)} = f_{(t)}.c_{(t-1)} + i_{(t)}.cv_{(t)} \tag{5}$$

$$h_{(t)} = o_{(t)}.\sigma(c_{(t)}) \tag{6}$$

Where $cv_{(t)}$ is the candidate value to use as an input for the next cell $c_{(t)}$ through time t, $w_{c,x}$ and $u_{c,h}$ are the weights for the current word x and the previous hidden state $h_{(t-1)}$,

b_c is the bias value initialized by 1, $c_{(t)}$ is the new cell state value, and $c_{(t-1)}$ is the previous cell state. The new hidden state $h_{(t)}$ is calculated by multiplying the data to output $o_{(t)}$ by the Sigmoid of the current cell state value $\sigma(c_{(t)})$.

The results are then passed as an input to the Softmax function to output a normalized vector (values between 0 and 1) [17]. Softmax function operates on a vector of multiple classes to get a vector of the potential probabilities of the outcome classes. Hence, the training is conducted on the sequences of IoT specifications having their predicted class labels. These classes are determined as per the common architectures of IoT-based systems based on the IoT system's standard components as: sensors and actuators, protocols and gateways, user devices and data processing [22][23].

The third layer boosts the learning model using the Backpropagation Through Time (BPTT) gradient technique [24]. BPTT is used to improve learning by enhancing the given weights to decrease the loss value and increase the accuracy of classification using a learning rate decay technique. It starts from the final output and moves backward through the same gates to re-train the classifier, where changes are applied to the weights. The network is then rolled up with the updated weights. This helps having a more accurate classifier. Thus, the learning decay rate determines the maximum number of possible call backs.

Finally, the Cross-Entropy Loss function is considered to evaluate the accuracy of the LSTM classifier as shown in (7) [25]:

$$CE = -\sum_{c=1}^{M} y_{o,c}.log(p_o, c) \tag{7}$$

Where CE is the Cross-Entropy, c is the current class, M is number of classes, $y_{o,c}$ is 1 in case the observed class p_o equals c, otherwise $y_{o,c}$ is 0, and $log(p_o, c)$ is the log of the probability of having class c. Thus, CE sums the loss value given the probabilities of all classes. The lower the loss value, the higher the accuracy of the classifier [21].

B. THE IOT TEST CASES CLASSIFICATION LAYER

This layer is responsible for classifying the TCs of the IoT system as per the IoT standard components using the trained model generated from the IoT Components Training Layer. The created classes would narrow the search space of TCs for selection and prioritization during the continuous regression and integration testing of the IoT system. The inputs are all previous test runs for this IoT system and the traceability matrix that has a list of IoT TCs mapped to requirements. It consists of several modules as follows:

- *IoT test cases attributes extraction*: the previous test runs and the traceability matrix of the system's specifications are considered to extract the main data identifying each TC, such as ID, name, description, coverage rate (CR), fault detection rate (FDR) and execution time (ET). The CR of a TC is the value of how many system's requirements are covered by this TC, the fault detection rate is the value of how many faults are detected by this TC compared to the total number of defects, and the execution time is the time taken by the TC to execute.

  The extracted TCs' attributes are used to classify TCs as per the IoT components, where the CR, FDR and ET are further utilized for TCs prioritization. The priority of a TC increases when its coverage and fault detection rates are higher while its execution time is lower [43][44]. The coverage rate is calculated for each TC by the summation of the covered requirements divided on the total number of the requirements as shown in (8) [45]:

$$CR_{TC_{ID}} = \sum_{R_{ID}=1}^{n} \frac{R_{ID}}{n} \tag{8}$$

  Where CR is the coverage rate, $TC_{ID}$is the TC identity, R_{ID} is the requirement identity and $n$is the total number of requirements. Using the previous IoT system's runs, the FDR and ET are extracted for each TC as per the traceability matrix. FDR is calculated by the summation of the faults that are covered by the TC as shown in (9) [46]:

$$FDR_{TC_{ID}} = \sum_{f_{ID}=1}^{n} \frac{F_{f_{ID}}}{F_n} \tag{9}$$

  Where FDR is the fault detection rate, TC_{ID} is the TC identity, F is the fault, f_{ID} is the fault identity and n is the total number of faults.

- *The test cases classification as per IoT components*, with the extracted attributes, as the TC name, ID, description, as well as the IoT components' trained model, are proposed to classify each TC as per the IoT standard components. The classification of TCs facilitates the selection of relevant TCs for the next layer, when a new module in case of integration testing or change requests in case of regression testing are encountered. The output of this layer is the classified TCs as per the IoT components.

C. THE IOT SELECTION AND PRIORITIZATION LAYER

This layer is responsible for selecting and prioritizing IoT TCs either for integration or regression testing based on the received inputs. TCs are prioritized and selected for a regression testing if the input is a change request, whereas if the inputs are the system architecture and new specifications, then TCs are selected and prioritized for an integration testing as follows.

- *Change request classification as per IoT components*, which indicates the application of regression testing for IoT TCs selection. The input is expected to be the change request, which should be classified as per the IoT components using the generated IoT components trained model. The classification of the change request determines the class of the IoT standard component for this change request to which it belongs to, whether sensors and actuators, protocols and gateways, user devices, or data processing. This helps reducing the costs and efforts required to select TCs by narrowing the search space out of the massive set of all TCs. After this classification, the IoT TCs selection starts based on the classified changed request and the classified IoT system TCs from the previous layer. Thus, IoT TCs are selected.

- *IoT Integration testing Recommender*, in which the Modules Interfaces Extraction receives a new module and the IoT system architecture, indicating the application of integration testing for IoT TCs selection. This optimizes integration testing for the nature of IoT systems. It extracts the modules' interfaces that are related to the integration between the input module's specifications and its adjacent modules. Modules Dependency Calculation is then responsible of calculating the dependency of modules by identifying the required number of stubs and drivers, in order to recommend the efficient integration testing approach to consider next, i.e. top-down, bottom-up or sandwich … etc., that decreases the number of needed stubs or drivers at the Integration Testing Handler. The integration testing approach is recommended based on the number of connected modules to this new module to test. If the number of connected follower modules are less than the connected ancestor modules but not yet implemented, then stubs are needed. If the number of the connected ancestor modules are less than the connected follower modules but not yet implemented, then drivers are needed. If the number of stubs and drivers is equal, then it is recommended to use stubs over drivers [40]. Otherwise, if all ancestor and follower modules are implemented, then all related TCs are selected without the need for any stubs or drivers.
- *Test cases selection using the classified IoT components*, in which the relevant TCs are selected during the regression or integration testing according to the related IoT standard components. In case of regression testing, the change request in IoT specifications that is under consideration should have been classified as per the IoT standard components. Accordingly, the TCs related to the classified IoT standard component are selected. In case of integration testing, TCs are selected based on the new IoT module under consideration with respect to the provided system architecture. The modules that are expected to be connected to the new module determine the set of TCs to select, which are classified according to IoT standard components.
- *Test Cases prioritization using multi search-based techniques*, in which the proposed IoT-CIRTF utilizes GAs [42] and SA [41] search-based techniques according to the extracted attributes CR, FDR and ET, to use in the prioritization Fitness Function (FF) [41]. By calculating FF for each TC, IoT-CIRTF decides whether this sequence of TCs is better than the previous sequence. The prioritization process starts by choosing a random TC, creating a sequence of ordered TCs. This sequence may not be the best solution, and accordingly, calculations keep processing until they stop as per a stop condition determined by the SBT algorithm to find the best sequence of ordered TCs. FF calculates each TCs weight as shown in (10):

$$FF(TC_{ID}) = \sum_{i=1}^{n} \frac{TC_{index} \cdot (CR_i + FDR_i)}{ET_i} \tag{10}$$

Where TC_{ID} is the TC identity, TC_{index} is the TC index indicating its position in the ordered sequence, n is the total number of TCs, CR_i is the coverage rate, FDR_i is the fault detection rate and ET_i is the execution time of the TC. The SBT algorithm keeps running until reaching the stopping condition, which represents the number of iterations to keep running for both SA and GA algorithms [48][49][50][51]. Algorithm 2 presents GA as the global search approach, while Algorithm 3 presents SA as the local search approach as applied in the proposed IoT-CIRTF.

Algorithm 2 Simulated Annealing for IoT TCs Prioritization in IoT-CIRTF.

```
Output: TCs prioritized, with the highest priority value.
1  Begin
2       Generate an initial solution (Sol)
3       Set Best solution (SolBest) = Sol
4       Set initial number of iterations value (T0)
        //T0>0, iterates until the end of iterations.
5       Loop
6          Select a random neighbor TC Solp
7          Calculate FF for Sol.
8          Calculate FF for Solp.
9          Calculate $\Delta$ = FF(Solp)−FF(Sol).
10         If $\Delta$<0 then
11            Sol = Solp.
12         End if
13         If $\Delta \geq 0$ then
14            Sol = Solp with probability P($\Delta$,T).
15         End if
16         If FF(Sol)≥FF(SolBest) then
17            SolBest = Sol.
18         End if
19         Upgrade the remained number of iterations (T)
           //using the initialized reduction rate.
20      End loop //iterations number (T0<=0).
21  End
22  Return SolBest//best sequence of ordered TCs.
```

Algorithm 3 Genetic Algorithm for IoT TCs Prioritization in IoT-CIRTF.

Output: TCs prioritized with the highest priority value.

```
1   Begin
2     Generate an initial population (P)
3     Initialize  iteration i: i=1
4     Generate n number of TCs
5       Loop
6         Calculate FF for P
7         Choose two TCs based on FF //Selection
8         Generate new offspring //Crossover combination
9         Perform mutation on each TC  //Mutation
10        Choose new Population of TCs //Acceptance
11        If (Optimal solution reached)
12            Break Loop
13        Else
14            Return to step 6 //Generate a new population
15        End if
16  End
17  Return prioritized TCs
```

SBT is used to search for an optimized sequence of ordered TCs. Local search approaches search for the best solution by looking to the nearby solution TC, while global search approaches find the optimum solution anywhere in the search space. IoT-CIRTF applies both approaches to validate the accuracy of finding the best sequence of TCs, in which the number of TCs in IoT systems should be reduced to effectively manage the time and cost needed for continuous testing. Both approaches are adapted by modifying FF as shown in (6), where the weights of TCs are calculated as per the three essential parameters CR, FDR, ET. The output of IoT-CIRTF is the selected prioritized TCs for the regression or integration testing in the IoT-based system.

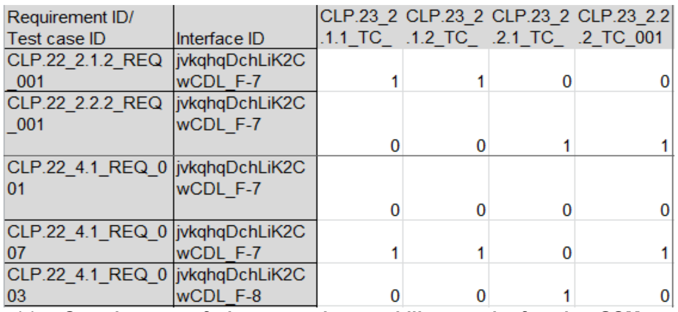
IV. IOT-CIRTF EXPERIMENTAL EVALUATION

Various experimentations were considered to evaluate the proposed IoT Continuous Integration and Regression Testing Framework (IoT-CIRTF) with respect to the accuracy encountered at three perspectives: the deep learning layer, the TCs classification layer, and the IoT TCs selection and prioritization layer while applying regression and integration testing for IoT-based systems.

A. THE DATASET

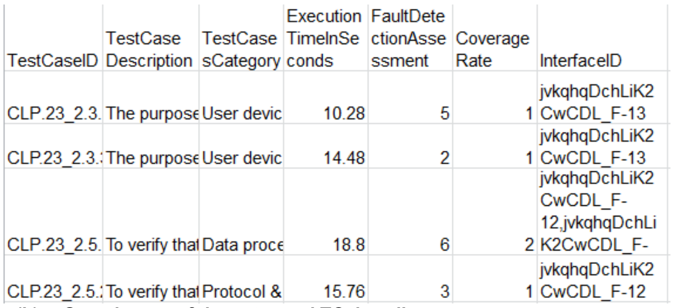
The global system for mobile communications (GSM) was used as our dataset [52]. It was selected for our experimentation as it includes IoT system specifications and their associated TCs. It supports mobile operators for cellular networks, where huge devices are connected to obtain services through the internet. To avoid failures when connecting IoT devices to mobile networks, GSM has provided two datasets: IoT device connection efficiency and Mobile IoT (MIoT). Both datasets include the architecture for the configuration of the mobile network controls, consisting of common TCs as per the IoT systems specifications. The IoT device connection efficiency dataset includes 80 requirements and 100 TCs [53] [54], whereas the MIoT dataset contains 51 requirements and 41 TCs [55].

The component diagram of this architecture shows the dependency between the components, which will be used for the integration testing evaluation as described later. We have created the traceability matrix (TR) for these datasets to present the related TCs of all requirements, and other documents showing all TCs with their previous runs information and extracted attributes [56]. Sample parts of the used datasets based on GSM are shown in Fig.2.



Requirement ID/ Test case ID	Interface ID	CLP.23_2.1.1_TC_	CLP.23_2.1.2_TC_	CLP.23_2.2.1_TC_	CLP.23_2.2.2_TC_001
CLP.22_2.1.2_REQ_001	jvkqhqDchLiK2CwCDL_F-7	1	1	0	0
CLP.22_2.2.2_REQ_001	jvkqhqDchLiK2CwCDL_F-7	0	0	1	1
CLP.22_4.1_REQ_001	jvkqhqDchLiK2CwCDL_F-7	0	0	0	0
CLP.22_4.1_REQ_007	jvkqhqDchLiK2CwCDL_F-7	1	1	0	1
CLP.22_4.1_REQ_003	jvkqhqDchLiK2CwCDL_F-8	0	0	1	0

(a) **Sample part of the created traceability matrix for the GSM dataset, after adding the IoT components' interfaces IDs.**



TestCaseID	TestCase Description	TestCasesCategory	ExecutionTimeInSeconds	FaultDetectionAssessment	Coverage Rate	InterfaceID
CLP.23_2.3.	The purpose	User devic	10.28	5	1	jvkqhqDchLiK2CwCDL_F-13
CLP.23_2.3.:	The purpose	User devic	14.48	2	1	jvkqhqDchLiK2CwCDL_F-13
CLP.23_2.5.	To verify that	Data proce	18.8	6	2	jvkqhqDchLiK2CwCDL_F-12,jvkqhqDchLiK2CwCDL_F-
CLP.23_2.5.:	To verify that	Protocol &	15.76	3	1	jvkqhqDchLiK2CwCDL_F-12

(b) **Sample part of the extracted TCs' attributes.**

FIGURE.2: **Sample parts of the used dataset based on GSM.**

B. THE EXPERIMENTAL METHODOLOGY

A system was developed for IoT-CIRTF using JAVA 8 and Python 3.7.3 programming languages. Experiments are performed on an Intel(R) Core (TM) i7-75500U CPU (2.40 GHz), 8 GB RAM and 64-bit Windows 8 operating system. Several runs were conducted to create and train LSTM classifier reaching best accuracy when using 4 layers/ nodes of LSTM and around 100 epochs to cover the whole IoT system specifications.

As for the prioritization, SBT (GA and SA) required to execute a number of iterations based on the stop conditions of both algorithms to reach the best solution of ordered TCs. The main evaluation criterion for TCs prioritization is measured by precision, where it is defined by the total number of TCs in the correct order, divided by the total number of TCs as shown in (11) by [57].

$$Pr\,e\,cision = \frac{TP}{TP+FP} \qquad (11)$$

Where TP is the total number of TCs correctly ordered, i.e. true positive, and FP is the total number of TCs incorrectly ordered, i.e. false positive. The experiments can be categorized into three perspectives to examine each layer at IoT-CIRTF as detailed below.

C. IOT COMPONENTS TRAINING LAYER EVALUATION

In order to examine the learning process of mapping the IoT system specifications to the IoT standard components (i.e. User device, protocols and gateways, sensors and actuators, data processing), the used datasets were split into 80% for training and 20% for testing. The learning was then applied using two layers of LSTM, i.e. Bi-directional algorithm. An average accuracy of 75% was obtained, where the large sentences in the IoT specifications led to the need for improving the results. Therefore, four layers of LSTM were applied [58], leading to an average accuracy of 96%.

D. IOT TEST CASES CLASSIFICATION LAYER EVALUATION

As for the evaluation of TCs classification as per the four standard IoT components based on the LSTM classifier, the TCs classification's accuracy was 96% on average, with an average of 100 epochs, after having an accuracy of 75% when using two layers of LSTM only. Other studies as in [59] have achieved accuracy of 92.84%, 97.2% in [60], 74% in [61], and (81.2%) in [62] when applying adaptations on the number of layers with long sequences of data in different prediction and classification applications. based on the best trained LSTM classifier [63].

Fig.3 shows the classification accuracy of both datasets through the loss rate using the Cross-Entropy Loss function for the four layers of LSTM. The results indicate that the loss rate percentage decreases as the number of LSTM classifier layers increases, in which using more layers prompts having less classification errors with the used classifier [64].

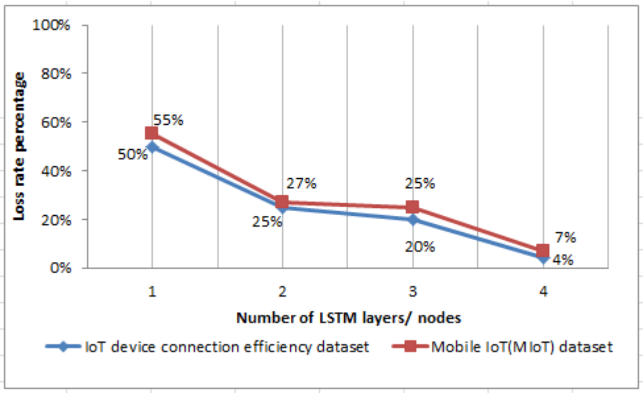


FIGURE.3: The loss rate for both datasets with respect to the LSTM layers.

V. IOT TEST CASES SELECTION AND PRIORITIZATION EXPERIMENTAL RESULTS

The experiments on this layer consider two main parts: for regression testing and for integration testing. The evaluation is based on the selection and prioritization results as follows.

A. REGRESSION TESTING EXPERIMENTAL RESULTS

The correct selection of TCs is determined by the IoT system's traceability matrix that matches the IoT standard component as per the classified change request. Fig.4 shows the results of TCs prioritization for regression testing, where the y axis represents the FF value of the selected prioritized TCs, and the x axis represents the number of iterations. Prioritization was applied using both SA and GA algorithms for both mentioned datasets. The results are shown through 10 iterations. The total number of iterations varied according to the used algorithm and stopping condition. The FF values of the MIoT dataset are lower than those of the IoT device efficiency dataset, as the size of the later dataset is larger [56]. The results indicate that GA works more accurate with large datasets, since it is a global SBT technique that pursues a population of values rather than singular neighbor values as in the case of local SBT algorithms like the SA algorithm.

The results of SA are near to the results of GA for the regression testing with the used datasets, where the number of selected TCs for regression testing is less than the TCs selected for integration testing. The increase of FF triggers better sequence of selected ordered TCs, due to the increase of FDR, CR and the decrease of ET. In addition, precision is calculated to validate the accuracy of the prioritization results. The average precisions achieved for the IoT device connection efficiency dataset were 72% and 88% for the SA and GA algorithms respectively. The average precisions achieved for the MIoT dataset were 90% and 81% using SA and GA algorithms respectively, resulted at the final iteration, as shown in Fig.6.

B. INTEGRATION TESTING EXPERIMENTAL RESULTS

The evaluation of Integration testing at IoT-CIRTF is determined by the efficiency of deciding the correct integration testing type. The architecture is read as an XML file, then the dependency is calculated between the newly implemented module to test and the ancestor and follower modules to determine the optimum integration testing type. Fig.5 shows the TCs prioritization accuracy results using both GA and SA algorithms, where the x axis presents the number of iterations and the y axis presents the FF value of the selected prioritized TCs.

The accuracy of the prioritized selected TCs using GA and SA for integration testing was examined by precision. In the case of the IoT device connection efficiency dataset, the precision average values were 80% and 92% for the SA and GA algorithms respectively. As for the MIoT dataset, the average precisions were 77% and 89% for SA and GA respectively. The precision values resulted from the final iteration of the used algorithms, as shown on Fig.6. GA proved more efficient percentages when used for the integration testing, where the number of selected TCs was more than those for the regression testing. Prioritization using global SBT such as GA is more efficient with larger selected TCs.

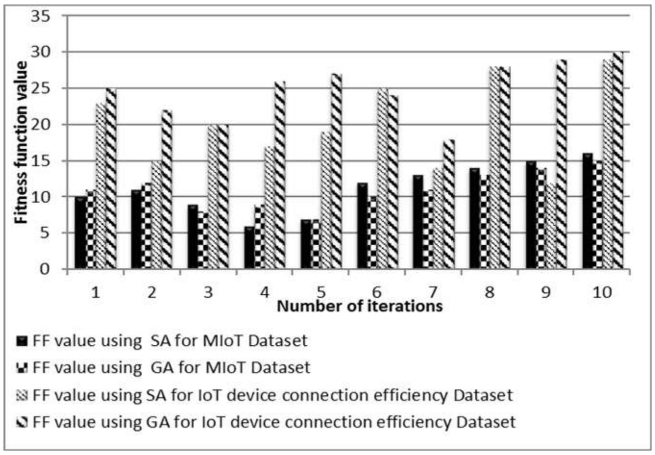


FIGURE.4: Results of TCs prioritization for regression testing for both used datasets using SA and GA.

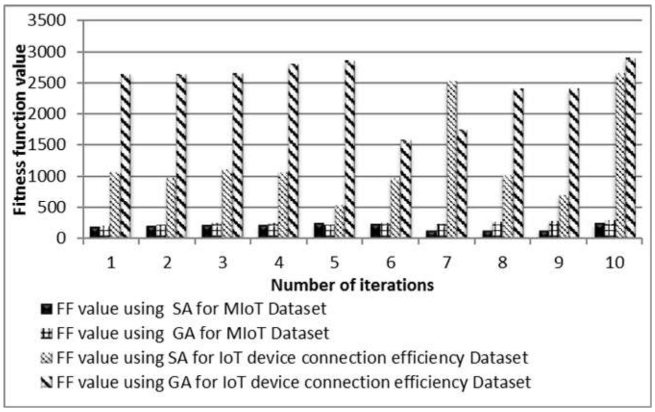


FIGURE.5: Results of TCs prioritization for integration testing when receiving a new IoT system module for both datasets using SA and GA.

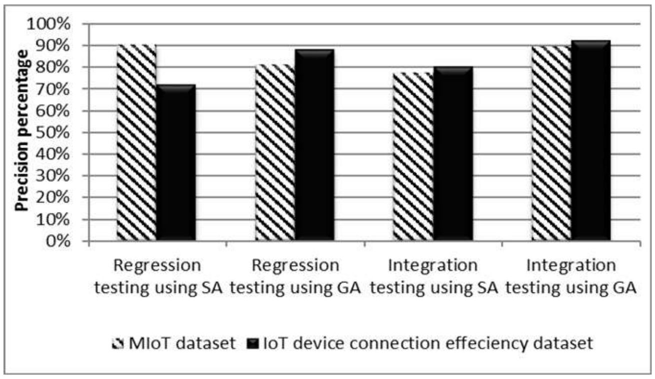


FIGURE.6: Results of precision percentages for prioritized selected TCs using SA and GA for both datasets.

VI. CONCLUSION

Challenges are usually faced during the integration and regression testing of IoT systems, in which the nature of these systems requires continuous testing. In this paper, the IoT Continuous Integration and Regression Testing Framework (IoT-CIRTF) is proposed for continuous regression and integration testing in IoT-based systems based on a hybrid combination of deep learning and search-based techniques. The LSTM classifier was trained for two IoT-based system specifications datasets, achieving an average accuracy of 96% using 100 epochs for 4 layers.

The classifier considered 4 standard components of IoT systems, with loss rates of 4% and 6% when applied for the IoT device connection efficiency and Mobile IoT (MIoT) datasets respectively, indicating the accuracy of the classifier. The prioritization accuracy of the selected TCs for both integration and regression testing was examined by calculating the precision. The results for regression testing were 72% and 88% using Simulated Annealing (SA) and Genetic algorithms (GAs) algorithms for the IoT device connection efficiency dataset, and 90% and 81% using SA and GA algorithms for the Mobile IoT (MIoT) dataset respectively. As for the integration testing, the precision results were 80% and 92% using SA and GA algorithms for the IoT device connection efficiency dataset, and 77% and 89% using SA and GA for the MIoT dataset respectively.

As for the future work, it is planned to extend our proposed IoT-CIRTF framework to include other IoT testing areas, such as compatibility testing, configuration testing, scalability testing and security testing.

REFERENCES

[1] F.Alkhabbas, R. Spalazzese, and P. Davidsson, Characterizing Internet of Things Systems through Taxonomies: A Systematic Mapping Study, Internet of Things, vol. 7, p. 100084, 2019.

[2] A. Vakaloudis and C. O. Leary, A framework for rapid integration of IoT Systems with industrial environments, 2019 IEEE 5th World Forum Internet Things, pp. 601–605, 2019.

[3] N. Medhat, S. Moussa, N. Badr and M. F. Tolba, Testing Techniques in IoT-based Systems. 2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS), Cairo, Egypt, 2019, pp.394-401,doi: 10.1109/ICICIS46948.2019.9014711.

[4] Fu, Y., Chen, H., Zheng, Q., Yan, Z., Kantola, R., Jing, X., Cao, J., Li, H., 2020, An Adaptive Security Data Collection and Composition Recognition method for security measurement over LTE/LTE-A networks. J. Netw. Comput. Appl. 155. https://doi.org/10.1016/j.jnca.2020102549

[5] R. Mohammad and H. Hamad, Automation Testing and Monitoring Lab on the Cloud for IOT Smart Fleet System (ATML & SFS). 2018.

[6] A. K. Gomez and S. Bajaj, Challenges of testing complex internet of things (IoT) devices and systems, Proc. 2019 11th Int. Conf. Knowl. Syst. Eng. KSE 2019, pp. 1–4, 2019.

[7] M. Noor and W. H. Hassan, “Current research on Internet of Things (IoT) security: A survey Computer Networks, 2018.

[8] M. Sirshar, Software Quality Assurance testing methodologies in IoT. no. December 2019, 2020.

[9] Z. B. Celik, Program Analysis of Commodity IoT Applications for. no. December, 2018.

[10] M. Leotta et al., An acceptance testing approach for internet of things systems. IET Softw., vol. 12, no. 5, pp. 430–436, 2018.

[11] M. Krichen, 2019. Improving Formal Verification and Testing Techniques for Internet of Things and Smart Cities. Mobile Networks and Applications.

[12] L. Gutiérrez-Madroñal, I. Medina-Bulo, and J. J. Domínguez-

Jiménez, IoT–TEG: Test event generator system. Journal of Systems and Software, vol. 137, pp. 784–803, 2018.

[13] M. T. An and E. Analysis, Cost and Effectiveness of Search-Based Techniques for. vol. 27, no. 4, pp. 601–622, 2017.

[14] Romero, A., Gatta, C., Camps-valls, G., Member, S., 2015. Sensing Image Classification. IEEE Trans. Geosci. Remote Sens. 1 Unsupervised 54, 1–14. https://doi.org/10.1109/TGRS.2015.2478379.

[15] Chen, Z., Zou, H., Yang, J.F., Jiang, H., Xie, L., 2020. WiFi Fingerprinting Indoor Localization Using Local Feature-Based Deep LSTM. IEEE Syst. J. 14, 3001–3010. https://doi.org/10.1109/JSYST.2019.2918678

[16] Yang, B., Sun, S., Li, J., Lin, X., Tian, Y., 2019. Traffic flow prediction using LSTM with feature enhancement. Neurocomputing 332, 320–327. https://doi.org/10.1016/j.neucom.2018.12.016

[17] Wang, Z., Zhu, Z., Li, D., 2020. Collaborative and geometric multi-kernel learning for multi-class classification 99. https://doi.org/10.1016/j.patcog.2019.107050

[18] Li, Z., He, D., Tian, F., Chen, W., Qin, T., Wang, L., Liu, T., 2018. Towards Binary-Valued Gates for Robust LSTM Training.

[19] Elsworth, S., Stefan, G., n.d. Time Series Forecasting Using LSTM Networks : A Symbolic Approach 1–12.

[20] Liu, G., Guo, J., 2019. PT US CR. Neurocomputing. https://doi.org/10.1016/j.neucom.2019.01.078

[21] Gelly, G., Gauvain, J.L., 2017. Spoken Language Identification using LSTM-based Angular Proximity 2566–2570.

[22] Lv, W., Meng, F., Zhang, C., Lv, Y., Cao, N., Jiang, J., 2017. A General Architecture of IoT System 659–664. https://doi.org/10.1109/CSE-EUC.2017.124

[23] Mocrii, D., Chen, Y., Musilek, P., 2018. Internet of Things IoT-based smart homes : A review of system architecture , software , communications , privacy and security. Internet of Things 1–2, 81–98. https://doi.org/10.1016/j.iot.2018.08.009

[24] Merity, S., Keskar, N.S., Socher, R., 2015. Regularizing and Optimizing LSTM Language Models.

[25] Zhang, Z., 2018. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels.

[26] W. Yu, M. Yi, X. Huang, X. Yi, and Q. Yuan, Make it directly: Event extraction based on tree-LSTM and Bi-GRU. IEEE Access, vol. 8, pp. 14344–14354, 2020.

[27] Sansano, E., 2020. A study of deep neural networks for human activity recognition 1–27. https://doi.org/10.1111/coin.12318

[28] Stuhlsatz, A., Lippel, J., Zielke, T., 2010. Feature extraction for simple classification. Proc. - Int. Conf. Pattern Recognit. 1525–1528. https://doi.org/10.1109/ICPR.2010.377

[29] O. Olajubu, S. Ajit, M. Johnson, S. Thomson, M. Edwards, and S. Turner, 2017. Automated test case generation from high-level logic requirements using model transformation techniques. Conf. CEEC 2017 - Proc , 178–182.

[30] Wang, X., Zeng, H., Gao, H., Miao, H., Lin, W., 2019. Location-Based Test Case Prioritization for Software Embedded in Mobile Devices Using the Law of Gravitation. Mobile Information Systems 2019, 1–14.

[31] S. Y. Shin, L. C. Briand, and F. Zimmer, Test Case Prioritization for Acceptance Testing of Cyber Physical Systems : A Multi-objective Search-Based Approach. pp. 49–60.

[32] Incki, K., Ari, I., 2018. Democratization of runtime verification for internet of things. Computers & Electrical Engineering 68, 570–580.

[33] M. Krichen and M. Lahami, "Towards a Runtime Testing Framework for Dynamically Adaptable Internet of Things Networks in Smart Cities," pp. 589–607, 2020.

[34] J. P. Dias, F. Couto, A. C. R. Paiva, and H. S. Ferreira, A brief overview of existing tools for testing the internet-of-things. Proc. 2018 IEEE 11th International Conference on Software Testing, Verification and Validation Workshops, ICSTW, pp. 104–109, 2018.

[35] B. Lima, Automated scenario-based integration testing of time-constrained distributed systems. Proc, 2019 IEEE 12th Int. Conf. Softw. Testing, Verif. Validation, ICST 2019 , 486–488.

[36] Bures et al., M., Klima, M., Rechtberger, V., Bellekens, X., Tachtatzis, C., Atkinson, R., Ahmed, B.S., 2020, Interoperability and Integration Testing Methods for IoT Systems: A Systematic Mapping Study.

[37] L. Zhang, "Intelligent Regression Testing for Internet of Things Wireless Device Using Mixed Machine Learning," pp. 0–10, 2019.

[38] Guseila, L.G., Bratu, D., Moraru, S., 2019. Continuous Testing in the Development of IoT Applications.

[39] Pallavi, K., Mallapur, J.D., Bendigeri, K.Y., 2018. Remote sensing and controlling of greenhouse agriculture parameters based on IoT. 2017 Int. Conf. Big Data, IoT Data Sci. BID 2017 2018-Janua, 44–48. https://doi.org/10.1109/BID.2017.8336571

[40] Pustogarov, I., Wu, Q., Lie, D., 2020. Ex-vivo dynamic analysis framework for Android device drivers,1088–1105. https://doi.org/10.1109/sp40000.2020.00094

[41] Mukherjee, R., Patnaik, K.S., 2019. Prioritizing JUnit Test Cases Without Coverage Information: An Optimization Heuristics Based Approach. IEEE Access 7, 78092–78107.

[42] W. Jun, Z. Yan, and J. Chen, Test case prioritization technique based on genetic algorithm. Proc. - 2011 Int. Conf. Internet Computer, 173–175.

[43] Mahdieh, M., Mirian-Hosseinabadi, S. H., Etemadi, K., Nosrati, A., & Jalali, S. (2020). Incorporating fault-proneness estimations into coverage-based test case prioritization methods. Information and Software Technology, 121(December 2019), 106269. https://doi.org/10.1016/j.infsof.2020.106269

[44] Rothermel, G., Untcn, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 27(10), 929–948. https://doi.org/10.1109/32.962562

[45] H. Srikanth, L. Williams, J. Osborne, C. Science, and N. Carolina, System Test Case Prioritization of New and Regression Test Cases. vol. 00, no. c, pp. 64–73, 2005.

[46] McMaster, S., Memon, A., 2007. Fault detection probability analysis for coverage-based test suite reduction. IEEE Int. Conf. Softw. Maintenance, ICSM 335–344. https://doi.org/10.1109/ICSM.2007.4362646

[47] Ye, Z., Xiao, K., Ge, Y., Deng, Y., 2019. Applying Simulated Annealing and Parallel Computing to the Mobile Sequential Recommendation. IEEE Trans. Knowl. Data Eng. 31, 243–256. https://doi.org/10.1109/TKDE.2018.2827047

[48] Murata, T., Ishibuchi, H., 1994. Performance evaluation of genetic algorithms for flowshop scheduling problems. IEEE Conf. Evol. Comput. - Proc. 812–817. https://doi.org/10.1109/icec.1994.349951

[49] Maheswari, R.U., Jeya Mala, D., 2015. Combined Genetic and simulated annealing approach for test case prioritization. Indian J. Sci. Technol. 8. https://doi.org/10.17485/ijst/2015/v8i35/81102

[50] Prakash, B., Viswanathan, V., 2020. A comparative study of meta-heuristic optimisation techniques for prioritisation of risks in agile software development. Int. J. Comput. Appl. Technol. 62, 175–188. https://doi.org/10.1504/IJCAT.2020.104688

[51] Balaprakash, P., Wild, S.M., Hovland, P.D., 2013. An experimental study of global and local search algorithms in empirical performance tuning. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 7851 LNCS, 261–269. https://doi.org/10.1007/978-3-642-38718-0_26

[52] Adewole AP, Egunjobi TO, 2012. Foundation of Computer Science FCS. Int. J. Appl. Inf. Syst. 4, 6–12.

[53] Haq, I., Rahman, Z.U., Ali, S., Faisal, E.M., 2017. GSM Technology: Architecture, Security, and Future Challenges. Int. J. Sci. Eng. Adv. Technol. 5, 70–74.

[54] GSM, 2014. IoT Device Connection Efficiency Guidelines 1–73.

[55] GSM, 2015. IoT Device Connection Efficiency Common Test Cases 30 January 2015 1–51.

[56] GSM, 2017. MIoT Test Requirements 1–24. TC, G., 2017. MIoT Test Cases 1–40

[57] Noha Medhat, Sherin Moussa, "loT Specifications, Traceability Matrix and Test Cases based on GSM", IEEE Dataport, 2020. [Online]. Available: http://dx.doi.org/10.21227/f3j5-nm63. Accessed: Oct. 15, 2020.

[58] Ye, X., Fang, F., Wu, J., Bunescu, R., Liu, C., 2019. Bug Report Classification Using LSTM Architecture for More Accurate Software Defect Locating. Proc. - 17th IEEE Int. Conf. Mach. Learn. Appl. ICMLA 2018 1438–1445. https://doi.org/10.1109/ICMLA.2018.00234

[59] Ullah, F.U.M., Ullah, A., Member, S., Haq, I.U., Member, S., 2019.

Short-Term Prediction of Residential Power Energy Consumption via CNN and Multilayer Bi-directional LSTM Networks. IEEE Access PP, 1. https://doi.org/10.1109/ACCESS.2019.2963045

[60] Ullah, A., Ahmad, J., Muhammad, K., Sajjad, M., Baik, S.W., Technology, C., Image, D., 2017. Action Recognition in Video Sequences using Deep Bi-directional LSTM with CNN Features 3536, 1–11. https://doi.org/10.1109/ACCESS.2017.2778011

[61] Khan, M.A., Karim, R., 2019. SS symmetry A Scalable and Hybrid Intrusion Detection System Based on the Convolutional-LSTM Network.

[62] Xia, T., Song, Y., Zheng, Y., Pan, E., Xi, L., 2020. Computers in Industry An ensemble framework based on convolutional bi-directional LSTM with multiple time windows for remaining useful life estimation. Comput. Ind. 115, 103182. https://doi.org/10.1016/j.compind.2019.103182

[63] Li, S., Li, W., Cook, C., Zhu, C., Gao, Y., 2018. Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN 5457–5466.

[64] Liu, G., & Guo, J. (2019). Bidirectional LSTM with attention mechanism and convolutional layer for text classification. Neurocomputing. https://doi.org/10.1016/j.neucom.2019.01.078