Using Machine Learning to Build Test Oracles: an Industrial Case Study on Elevators Dispatching Algorithms

Aitor Arrieta*, Jon Ayerdi*, Miren Illarramendi*, Aitor Agirre†, Goiuria Sagardui* and Maite Arratibel ‡
Mondragon Unibertsitatea*, Ikerlan †, Orona ‡
*{aarrieta, jayerdi, millarramendi,gsagardui}@mondragon.edu, †aagirre@ikerlan.es, ‡marratibel@orona-group.com

***Abstract*—The software of elevators requires maintenance over several years to deal with new functionality, correction of bugs or legislation changes. To automatically validate this software, test oracles are necessary. A typical approach in industry is to use regression oracles. These oracles have to execute the test input both, in the software version under test and in a previous software version. This practice has several issues when using simulation to test elevators dispatching algorithms at system level. These issues include a long test execution time and the impossibility of re-using test oracles both at different test levels and in operation. To deal with these issues, we propose DARIO, a test oracle that relies on regression learning algorithms to predict the Qualify of Service of the system. The regression learning algorithms of this oracle are trained by using data from previously tested versions. An empirical evaluation with an industrial case study demonstrates the feasibility of using our approach in practice. A total of five regression learning algorithms were validated, showing that the regression tree algorithm performed best. For the regression tree algorithm, the accuracy when predicting verdicts by DARIO ranged between 79 to 87%.**

***Index Terms*—Test Oracle, Regression Testing, Machine-Learning**

I. Introduction

Elevators are complex Cyber-Physical Systems (CPSs) that must be validated at different test levels. Their software has a long life-cycle, requiring maintenance over several years. This software maintenance deals with (1) new functional and non-functional requirements, (2) correction of bugs, (3) legislative changes and (4) hardware obsolescence and system degradation [1]. In a system of elevators, one critical subsystem is its traffic master, which is in charge of managing the passengers' flow in a building. It is composed of several software modules, such as the dispatching algorithm, which is the component in charge of deciding which elevator must attend each passenger by considering several aspects (e.g., estimation of the waiting time of each passenger or estimated time required to arrive to destination). New generations of these algorithms are also starting to consider additional aspects, such as energy consumption. The algorithms also include certain functionalities based on the building, such as controlling access to disable specific floors to unauthorized passengers or special calls for handicapped persons (e.g., by assigning additional space in the elevator or assigning them the closest elevator). Orona, one of the leading elevator companies in Europe, has a large suite of dispatching algorithms that are in constant maintenance and evolution to solve different customers' demands.

When changes are performed in these algorithms, Orona has a well established systematic verification and validation process at different test levels (Software-in-the-Loop (SiL), Hardware-in-the-Loop (HiL) and Operation). At SiL and HiL, besides unit tests, full-day passenger tests are carried out, which have as objective to verify the Quality of Service (QoS) of the dispatching algorithms (e.g., Average Waiting Time). To determine whether the quality of an algorithm is satisfactory or not, a regression test oracle is used. Specifically, the same traffic profile (i.e., test input) is executed in a previous version of the dispatching algorithm, and compared with the dispatching algorithm version under test. If the QoS is better or similar, the test is catalogued as PASS. Conversely, if the QoS is not good enough, the test is catalogued as FAIL.

This regression test oracle has two main problems. Firstly, the test needs to be executed both in the algorithm under test as well as in a previous version of the algorithm. This can be time-consuming as the execution is performed using simulation-based testing at system level and the test inputs usually simulate full-day passenger traffic profiles. This problem is exacerbated at the HiL test level, where the simulation is performed in real-time, and thus, in order to simulate a full-day traffic profile, two full-days are required (i.e., one full-day for the new software version under test and another one for the previous software version). The second problem involves that the test oracle cannot be re-used for testing the new version "on-the-fly" in operation. This is a problem because the faults not detected in the validation phases can manifest in the real installation.

To solve these issues, in this paper we propose DARIO (Dispatching AlgoRIthm Oracle), a test oracle that relies on regression learning algorithms to automatically validate elevators dispatching algorithms. Instead of using regression oracles, DARIO trains a regression learning algorithm by using data from a previously tested dispatching algorithm version. Our approach provides several advantages. Firstly, the training process of the employed regression learning algorithms is much faster than employing a regression test oracle. While the regression learning algorithms used in DARIO take a few seconds to train, the regression test oracles used in practice may take from minutes to hours at the SiL test level and

days at the HiL test level. Secondly, DARIO can be used as streamlined oracle at SiL, HiL and in operation, permitting the detection of potential inconsistencies within the real system. Thirdly, DARIO returns a quantitative verdict value over the simulation time, which provides information of how close the system was from failing at each simulation step or how severe a fault was. This opens the possibility to include new verification and validation paradigms in the context of dispatching algorithms, including falsification-based test generation [2], [3]. It can also be used for on-line testing, which could save a significant amount of testing time by stopping the simulation if a severe fault is detected, as proposed by Menghi et al. in a recent study [4]. We can summarize our contributions as follows:

- We propose DARIO, a novel approach for building test oracles for system-level simulation-based testing of dispatching algorithm. DARIO works on top of regression machine-learning algorithms. These algorithms aim at predicting the reference QoS values of a system of elevators based on the passenger traffic data.
- We perform an empirical evaluation by using an industrial case study provided by Orona, one of the leading elevator companies in Europe. In this evaluation, we used mutation testing to determine how accurate DARIO was when compared to traditionally employed regression test oracles. We compared five regression learning algorithms, among which Regression Tree algorithms performed best.
- We provide a set of lessons learned from applying DARIO in practice.

The rest of the paper is structured as follows: Section II provides relevant background of the case study and the current software testing process at Orona. Section III presents our approach to automatically test elevators dispatching algorithms. Section IV evaluates our approach by means of an empirical evaluation with an industrial case study. Section V discusses lessons learned extracted from the evaluation. We position our work with the current state of the art in Section VI. Lastly, Section VII concludes our paper.

II. Case study and software testing process at Orona

Figure 1 shows an overview of the software development process that Orona carries out for dispatching algorithms. The entire development process consists of a total of ten steps divided into three phases: SiL, HiL and Operation. To keep the paper at a reasonable size, this section focuses solely on the testing process. For further details of the entire software development process, refer to [1].

When a new algorithm is developed or modifications are performed into an existing algorithm, the first tests are performed in Step 3 at the SiL test level. In this case, two kinds of tests are performed: (1) short scenario tests and (2) full-day tests. In the short scenario tests, specific functional properties are tested in the most isolated way possible; the expected outcome of a test in this case is obtained by either implementing simple assertions or manually. In the full-day tests, scenarios that mimic a normal full-day (or sub-scenarios of it) in the life-cycle of the system of elevators are executed. The expected outcome of these tests is related to certain QoS values over time obtained by re-executing the test in another algorithm or in an older version of it. At this level, all the hardware is simulated by employing a domain-specific tool named Elevate.

At the HiL test level, these test processes are repeated, but involving real hardware (e.g., real target processors, communication systems, real-time operating systems, human-machine interfaces). At this test level, the same types of test cases are executed, with the difference being that the execution is in real-time, and therefore, much more expensive. After this phase, the software is deployed in operation and some manual validations are performed by the software maintainers, which does not permit the execution of as many tests as in the previous SiL and HiL phases.

A test case in the context of Orona for testing a dispatching algorithm version refers to the following fields:

- Building installation: It is the context configuration at which the SUT is being executed. It has different fields, such as the number of floors the building has, the number of elevators, which floor is served by each elevator, etc. For each elevator, there are also different fields, such as the maximum load, the energy it consumes, dynamic information (e.g., speed, acceleration and jerk), etc.
- Test input (call list profile): a test input in this context refers to a *.txt file that includes a list of passengers. For each passenger, this file includes (1) the arrival time (i.e., when the passenger requests an elevator), (2) arrival floor, (3) destination floor, (4) weight of the passenger, (5) capacity factor by mass, (6) the loading time, (7) the unloading time and (8) information related to the behavior of the passenger when not all elevators serve all floors.
- Expected output: based on the input, what should the behaviour of the system be. This differs depending on the type of testing done. At unitary functional level tests, the expected output is typically related to a functional behaviour of the elevator (e.g., elevator number 1 attends calls from passengers 1 and 2, elevator number 2 attends a call from passenger 3). For long full-day tests, this is related to certain QoS metrics, which is addressed in this paper.

This paper focuses on the oracles applied for, what is known in Orona internally as “long full-day tests”. In these kinds of tests, the test inputs encompass long full-day passenger data that simulate the passenger flow in an installation. During the execution of these tests, the expected output refers to a time series of QoS values over the time. Typical QoS measures include the Average Waiting Time (AWT), Average Time to Destination (ATTD) or the consumed energy. Traditionally, the most used QoS measure in Orona has been the AWT, because, according to certain studies, it is the most sensitive measure for a passenger to determine whether a system of elevators performs well or not [5]. These test cases can be differentiated

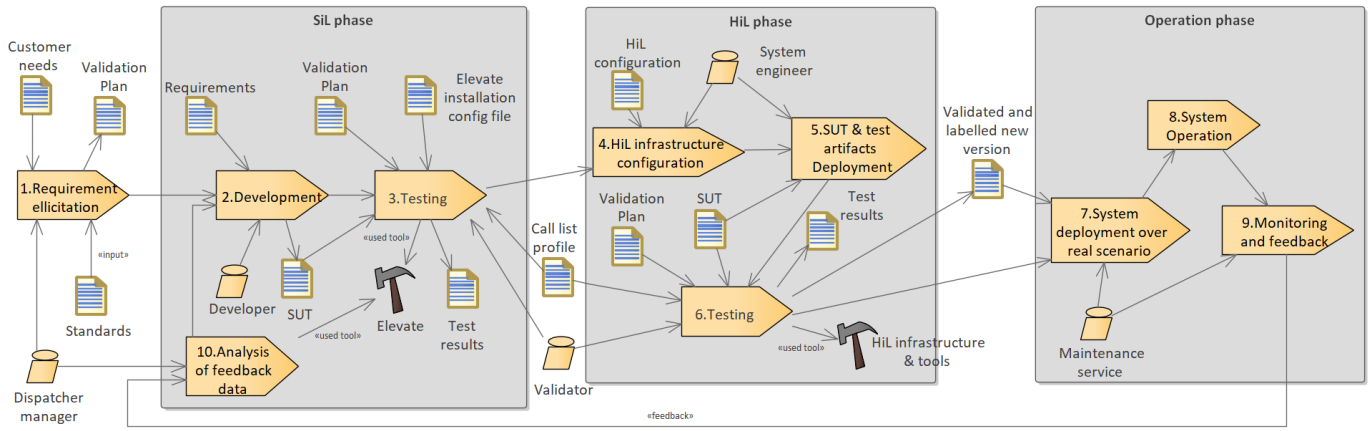

Fig. 1: Software development process of Orona's dispatching algorithms [1]

into two groups: theoretical and real.

On the one hand, theoretical passenger data based test cases provide test inputs based on theoretical studies of passenger flows in buildings. For instance, Figure 2 depicts a graph showing the number of up calls, down calls and inter-floor calls in a time window of five minutes for a simulation of 13 hours based on the Siikonen theory for a building of offices. As can be seen, in the first couple of hours, the number of up calls increases as passengers are arriving at the office. After 5 hours, a pattern is seen with an increase in the number of down calls followed by an increase in the number of up-calls, which represents the lunch time break. At the end of the day, there is another down peak, representing the end of the working day.

On the other hand, Orona uses data obtained from real installations. This helps with the validation of dispatching algorithms from several perspectives, such as the identification of certain traffic patterns not considered in theoretical traffic profiles or a higher customized validation. For instance, this could happen in a building where the canteen is on the top floor. Then, the traffic profile would significantly change as there would be a second peak in the number of up-calls followed then by an increase in the number of down-calls. An algorithm could also perform better than another depending on the traffic profile, and subsequently, the use of test inputs obtained from the field is a powerful method to validate these algorithms.

III. Regression learning-based test oracle for elevators dispatching algorithms

Figure 3 shows the overall architecture of the proposed approach. The proposed solution is divided into two main phases: (1) the training phase and (2) the testing phase. During the training phase, a regression learning algorithm is trained by using data from previous software versions. This data is the one catalogued by Orona as reference to validate other versions of the software. Subsequently, it is considered that the data used for training is from an error free version of a dispatching algorithm.

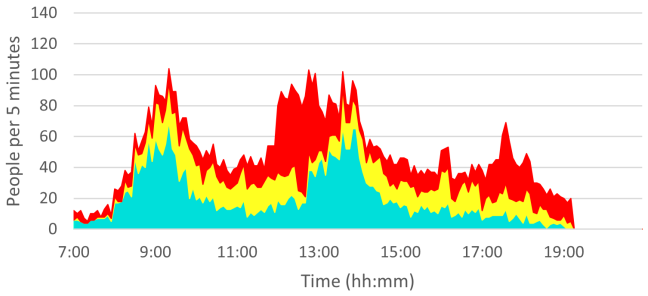

Fig. 2: A graph showing the number of up calls (blue), down calls (red) and inter-floor calls (yellow) in a theoretical traffic profile

The regression learning algorithm yields a model, which is used by DARIO in the testing phase. During this phase, Elevate executes a test case by using test input data, information of the building installation (i.e., different information such as, speed of elevators, structure of the building, etc.) and the SUT itself. When the test has finished, Elevate returns a set of files, which are treated by DARIO to extract (1) the passenger traffic profile and (2) the AWT over the simulation time. The former is used by the model predictor to predict the AWT over the simulation time. The latter is used by the arbiter of DARIO to yield the overall verdict and a quantitative verdict over the time.

A. Training phase

The first phase in our approach aims at feeding a machine-learning algorithm with labelled data to correctly train it. During the training phase, a machine-learning algorithm adapts some internal parameters based on training data so that it performs well on future unseen input data [6]. In Orona, the Verification and Validation activities are well documented, which gives availability of data from previously tested versions of the dispatching algorithms.

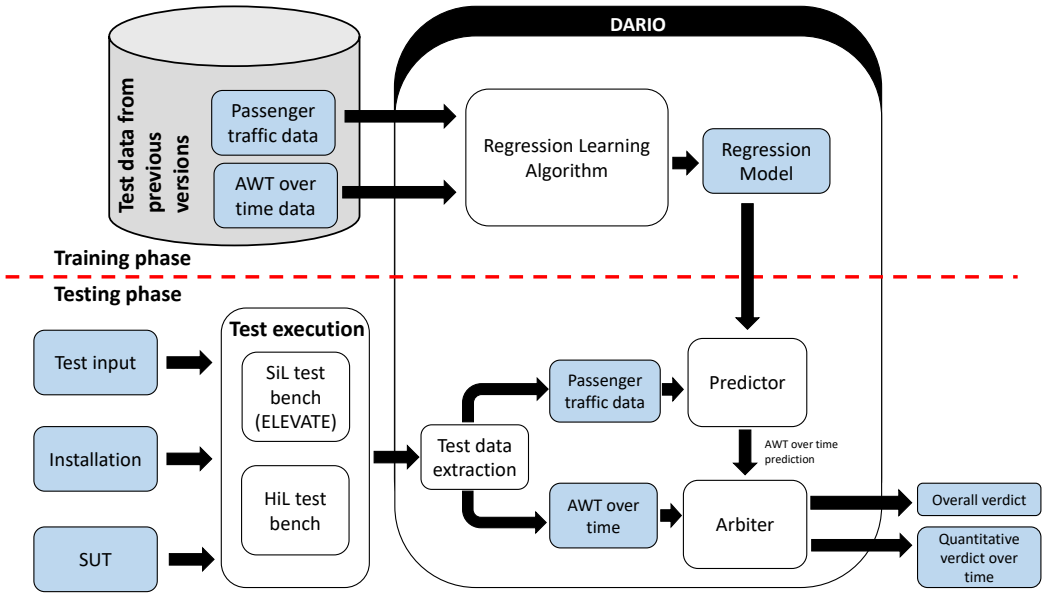

Fig. 3: Overview of the approach at the SiL test level

When testing dispatching algorithms at system level, the main QoS metric considered to label a test as PASS or FAIL by test engineers is the Average Waiting Time (AWT). This metric measures the average time each passenger needs to wait until their elevator arrives. The AWT can be a global value that measures the overall AWT for all passengers in the test input or a signal over the simulation time, indicating the AWT of the passengers in the test input for a specific time period. Elevate, one of the principal simulation tools for testing dispatching algorithms, provides information on both. DARIO uses the AWT for a specific time period to determine a verdict.

For training a machine-learning algorithm, the data is categorized into different domain-specific features related to passengers traffic data. As for the output feature, the AWT QoS metric is considered, as it is the measure that the dispatching algorithm under test used in this paper targets. All of them, for a time window of five minutes. A five minute time window was chosen based on the information provided by Elevate. We developed a script that is able to automatically extract this data from a database where Orona saves all the test history. When the data was extracted, the script launched the training phase by using the MATLAB machine-learning toolbox. The regression learning algorithm yields a trained regression model, which can later be used in the testing phase to predict the AWT.

It is noteworthy that typically the passenger traffic data in the historical test database is not the same as the test input in the testing phase because when changes are made in the dispatching algorithm, these changes typically include new functionalities or bug corrections. Subsequently, in the test inputs used during the testing phase the scenario testing the new functionality or a scenario that aims to trigger the fault is usually implemented. In addition, at the HiL test level, tests also might include scenarios where the test engineer tests the Human Machine Interface (HMI) of the system. In those cases, as the testing is manual, where by the system interacts with the tester, having the exact same test case is impossible.

B. *Testing phase*

When the regression learning algorithm is trained, it yields a trained regression model, which is used in the testing phase. For the current implementation, this phase has four steps: (1) test execution, where the dispatching algorithm is tested by using simulation-based testing, (2) test data extraction, where the test results and other necessary data is extracted, (3) prediction based on the regression model, which yields the expected AWT result, and (4), the arbitration process, which compares the AWT obtained by the regression algorithm with the AWT estimated by the regression model. We now explain all these steps in further detail.

1) Test execution: To execute a test, simulation-based testing is used. As previously mentioned, the test can be executed at two distinct levels: (1) at the Software-in-the-Loop test level and (2) at the Hardware-in-the-Loop test level.

The former refers to executing the test by using a tool named Elevate. This is a commercial tool used for testing dispatching algorithms in simulation. An executable file of the dispatching algorithm is generated, which is considered the System Under Test (SUT). The SUT is called by Elevate at each iteration, which simulates the rest of the parts of the elevators (i.e., speed, accelerations, opening and closing of the doors, etc.). The tool also gets as input data from the installation (e.g., building type, number of elevators, characteristics of each elevator), and the test input, which involves the passenger data.

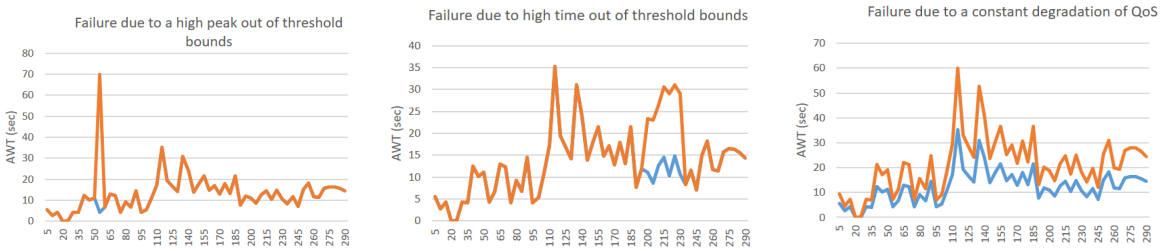

Fig. 4: The three reasons why a test can be catalogued as FAIL (blue signal refers to the reference AWT and orange signal refers to the AWT obtained by the software version under test)

The latter refers to executing the tests by using the real hardware and other real-time infrastructure. At this test level, the dispatching algorithm is integrated with other real-time infrastructure, such as the Linux real-time operating system, communication buses, drivers, etc. The real hardware that will later be used in the real elevator is used, including human-machine interface, target processors and CPUs, communication infrastructure, etc. The mechanical and electrical part of the elevators, though, are simulated within a real-time test bench. It is important to note that the execution of tests in this case is real-time. This test level also requires substantial manual effort for setting up the test bench, with activities including the deployment of the dispatching algorithm in the target, setting up hardware infrastructure, etc. Both test levels yield several files that include results from the simulation. These files include both, overall QoS measures (e.g., the overall AWT of the simulation, total energy consumed), as well as the QoS over the time. This information is provided to DARIO to carry out the validation process and provide the verdicts.

2) Test data extraction: After the test has been executed, DARIO extracts the necessary data from the testing files yielded by the test execution tools. At both test levels, i.e., SiL and HiL, both files are the same, which allows better reusability of the implemented test data extraction functionality. In the current version, DARIO needs to extract (1) the passenger traffic data over the time and (2) the AWT data over the time for the tested SUT version. The former involves input passengers' features data for each five minute time steps. This data is sent to the predictor, which provides an estimation of the AWT over the time required for each time step based on the trained regression model. The latter refers to the AWT data over the time obtained by the SUT for each of the five minute time steps.

3) AWT Prediction: The test data extraction provides the passenger traffic data profile over the time to the trained regression model. This model, estimates the AWT over the time based on the training produced during the training phase. This mimics the execution of the test case in the regression oracle. The AWT signal with a time step of 5 minutes is provided to the arbiter, which is the last component in charge of providing a test verdict.

4) Arbiter: For developing the test arbiter we discussed the reasons why a test can be catalogued as "FAIL" with test engineers that had the domain knowledge on testing dispatching algorithms. To this end, three reasons were identified, which are illustrated in Figure 4.

The first reason might be that at certain point, the software version under test shows a high peak on the AWT measure. This is because at a certain point, probably due to a bug, at least one passenger was unattended for a long period of time. The second reason is because the AWT measure for the software version under test exhibits a value higher than the specified threshold for a long period of time. The last scenario is related to a constant degradation of the AWT value throughout all the steps of the execution.

The developed arbitration algorithm aims at detecting these three scenarios. To this end, in a first step, the algorithm obtains the quantitative verdict for each simulation time step (in our case 5 minutes). We obtained this value by computing Equation 1.

A negative value means that the SUT version is performing worse than expected, whereas a positive value means that it is showing a better performance. It should be noted that if the referenceSignal were 0, the verdict would be either NaN or infinite (depending on the SUTSignal reference). To avoid this, the minimum referenceSignal value was established to 1 second.

$$verdict(t) = \frac{referenceSignal(t) - SUTSignal(t)}{referenceSignal(t)} \quad (1)$$

To detect failures of the first scenario, a threshold is specified and the arbiter checks whether the verdict exceeds this threshold in any step of the execution, which is the invariant expressed in Equation 2. We refer to this as the single-step arbiter.

$$\forall\, t \in [t_0, t_f] : verdict(t) \geq threshold_{single\_step} \quad (2)$$

where t_0 and t_f are the first and last steps of the execution, and $threshold_{single\_step}$ is the failure threshold for the verdict value defined for the single-step arbiter.

To detect failures of the second scenario, a different threshold is specified. When this threshold is exceeded, the arbiter checks the following steps in order to determine the duration of the anomaly. If this duration is longer than a specified maximum duration, the test is classified as FAIL. Equation

3 defines this invariant, which we refer to as the multiple-step arbiter.

$$\forall\, t_{start} \in [t_0, t_{f-D}] : \exists\, t \in [t_{start}, t_{start+D}] : \\ verdict(t) \geq threshold_{multiple\_step} \quad (3)$$

where $D+1$ is the maximum number of steps for multiple-step failures, and $threshold_{multiple\_step}$ is the failure threshold for the verdict value defined for the multiple-step arbiter.

For the last scenario, the average value of the verdict over time signal is obtained and compared against another threshold. Equation 4 defines this invariant, which we refer to as the average arbiter.

$$\frac{\sum^{t\in[t_0,t_f]} verdict(t)}{T} \geq threshold_{average} \quad (4)$$

where T is the number of steps in the execution.

Generally, the failure threshold for the arbiters is more tolerant for anomalies of shorter duration, since shorter duration samples may be less representative of the system. Therefore, the following will usually hold (note that threshold values are negative, and smaller values imply more tolerance):

$$threshold_{single\_step} < threshold_{multiple\_step} < threshold_{average} \quad (5)$$

C. Implementation

The tool was implemented in MATLAB. There are a few reasons behind this decision. The main reasons are that it provides support for a wide variety of algorithms. Furthermore, for all these algorithms, it provides a powerful C/C++ test generator, which would allow us to generate the code to execute DARIO within the real target processor in operation. The last major reason was that Elevate was integrated with BCVTB for co-simulation of the dispatching algorithm with other components of the system (e.g., the control of the elevators doors), for a higher fidelity level testing purposes [7]. BCVTB allows for the execution of MATLAB code, which permits us the execution of DARIO in this test bench with the goal of performing higher fidelity level simulation-based testing.

Additionally, although the approach is generalisable to any regression machine-learning algorithm, DARIO was implemented on top of the following ones: (1) Support Vector Machines (SVM), (2) Regression Decision Trees, (3) Ensemble, (4) Regression Gaussian Process (RGP) and (5) Stepwise Regression. The reason why these algorithms were chosen was (1) availability within the MATLAB framework and (2) appropriateness for our context in terms of prediction speed, training speed, memory usage and required tuning.

The selected algorithms have a fast prediction and training speed (unlike other algorithms such as neural networks); this is important in this case in order to speed up the verification and validation activities. In addition, these algorithms have a small memory usage, something important when deploying the oracles in operation, where embedded processors with limited resources are used. Lastly, the selected algorithms require minimal tuning, something that is paramount to ease the transfer of the approach to practitioners.

IV. Empirical Evaluation

In this section we empirically evaluate our approach. Our evaluation aims to answer the following Research Questions (RQs):

- **RQ1 – Training with theoretical data:** Which regression learning algorithm yields the most accurate test verdicts when trained with theoretical passenger test data?
- **RQ2 – Training with real passenger data:** Which regression learning algorithm yields the most accurate test verdicts when trained with real passenger test data?

A. Experimental setup

1) Case study: We used the Orona's Conventional Group Control (CGC) algorithm. This algorithm was selected as a case study because it is the most widely used algorithm. Furthermore, there are several versions of this algorithm available in Orona, which allowed us to have access to several sets of relevant test data for performing the experiments. Lastly, its complexity is high as it is continuously evolving. It is important to note that the algorithm is deterministic, and thus, it does not involve random variations, unlike other dispatching algorithms (e.g., genetic algorithms).

In the evaluation, we used a complex building installation that Orona typically uses to validate dispatching algorithms, which is related to a real installation named the communication city, in Madrid. The building has a total of 10 floors and six elevators, each having a capacity of 1250 Kg weight and 16 passengers. Another reason for choosing this building is that Orona has relevant data obtained from the real installation while in operation, which allowed us to answer RQ2. To train the machine learning algorithms of our oracles, we used available test data for testing a previous version of the CGC algorithm within the specific building. This data included ten theoretical passenger list test inputs and four real passenger list test input data.

2) Evaluation metrics: Mutation testing was used to seed faults through the dispatching algorithm under test. This technique has been found to be a good substitute of real faults [8]. The dispatching algorithms are programmed in C. Therefore, traditional mutation operators for the C programming language were used, such as relational operator changes, arithmetic operator changes, etc. These faults were introduced in a uniform manner throughout the sections of the source code that are relevant in the simulation environment.

We generated a total of 99 mutants. Although this is not a large number, it is important to note that as simulation-based testing was used, executing each mutant took a long time. This number is similar or larger to other studies where simulation-based testing was used to evaluate testing approaches [9], [10], [11]. From these 99 mutants, 18 were removed from the evaluation. The reason was that the inclusion of these mutants led the system to crashing or the simulation not lasting

because passengers were not attended. In practice, both types of failures are easily detected by test engineers in Orona, test oracles not being necessary. These 18 invalid mutants were removed from the initial set, using a total of 81 mutants in our evaluation. The 81 mutants were reviewed by a domain expert to check that they were not semantically equivalent to the original program.

When tests were executed, we obtained the AWT over the time by using Elevate, the simulation tool used by Orona to validate elevators dispatching algorithms. Based on a similar work [12], we selected four measures to evaluate the quality of the test oracles: precision (Equation 6), recall (Equation 7), f1 (Equation 8) and accuracy (Equation 9). Accuracy is especially important in our study as it is the only measure that considers True Negatives. In our context, classifying faults well is as important as classifying correct behaviour as correct.

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = \frac{2 \times (precision \times recall)}{precision + recall} \tag{8}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

For each mutant and each passenger list, we considered the overall verdict returned by DARIO, catalogued either as "PASS" or "FAIL". Additionally, we used the same passenger list with a regression test oracle (i.e., an original previous version under test), which is the current practice to determine if a test passes or fails by Orona.

Similar to other works tackling the test oracle problem [12], [13], the verdict provided by DARIO was considered a true negative (TN), a true positive (TP), a false negative (FN) or a false positive (FP) as defined below:

- TN: Both the test oracle (i.e., DARIO) and the regression test oracle returned a "PASS" verdict.
- TP: Both the test oracle (i.e., DARIO) and the regression test oracle returned a "FAIL" verdict.
- FN: The test oracle (i.e., DARIO) returned a "PASS" and the regression test oracle returned a "FAIL".
- FP: The test oracle (i.e., DARIO) returned a "FAIL" and the regression test oracle returned a "PASS".

The tests were executed in Elevate instead of in the HiL due to practicality (i.e., if the tests were executed using the HiL test bench, the experiments would take around 2 years).

3) Experimental Scenarios: A total of four experimental scenarios were designed for answering the two RQs:

Scenario 1: To answer the first RQ, we first used test cases that involve theoretical test inputs. These test inputs were automatically generated by Elevate, and are based on a study performed by Siikonen [14]. In total, 10 of these test cases were used. We employed the 10-fold cross validation to validate DARIO along with all the selected machine-learning algorithms.

Scenario 2: We used an additional scenario to answer the first RQ. The same type of test cases were used for training, but for testing, test cases obtained from the real installation were used. This scenario would emulate (1) how the theoretical passenger data performed in order to train the algorithms during validation when using data extracted from operation and (2) how the theoretical passenger data perform for training the algorithms when the oracle is used in the real installation.

Scenario 3: To answer the second RQ, we used test cases obtained with data extracted from the real installation in operation. In total, four of these test cases were used. We thus employed the 4-fold cross validation to validate DARIO along with all the selected machine-learning algorithms.

Scenario 4: Similar to Scenario 2, we used an additional scenario to answer the second RQ. In this case, we used the same test cases as in Scenario 3 to train the algorithms, but for testing we used the ten theoretical passenger-based test cases.

Table I summarizes the main characteristics of the test cases used in these four scenarios, including, the number of up calls, down calls, number of detected mutants by the test case, and the test case duration.

TABLE I: Main characteristics of the used test cases during the experimental scenarios

Test case	# of Up Calls	# of Down Calls	# of Detected Mutants	Simulation time (h:min)
real1	2756	1711	18	8:30
real2	3086	2366	18	9:10
real3	3438	3117	18	11:45
real4	3508	3050	21	13:35
theoretical1	3994	3377	20	12:55
theoretical2	3950	3379	18	12:55
theoretical3	3983	3379	26	12:55
theoretical4	3989	3402	18	12:55
theoretical5	3989	3387	18	12:55
theoretical6	3964	3384	19	12:55
theoretical7	3977	3386	21	12:55
theoretical8	3919	3433	21	12:55
theoretical9	3976	3354	18	12:55
theoretical10	3945	3407	20	12:55

B. Analysis of the results and discussion

Table II summarizes the obtained results for the four scenarios designed to answer the RQs. The first RQ aimed at answering how the selected machine learning algorithms performed when trained with theoretical passenger data based test cases. Scenario 1 aimed at answering this RQ, where a 10-fold cross validation was performed. When using the default parameters, in terms of precision, SVM was the technique showing best results followed by regression tree and stepwiselm. When considering the recall measure, regression tree and ensemble performed best, followed by SVM. In terms of accuracy and F-measure, SVM performed best, although the results by regression tree were close in both cases, unlike the remaining three machine learning algorithms, which dropped below 0.8 in terms of both accuracy and F-measure.

As for the second scenario, where the ten theoretical passenger data based test cases were used for training the machine

TABLE II: Summary of results for the four experimental scenarios within our industrial case study

Scenario	Metrics	SVM	Regression Tree	Ensemble	RGP	STEPWISELM
Scenario 1	Precision	0.89	0.83	0.68	0.76	0.82
	Recall	0.88	0.89	0.89	0.78	0.78
	Accuracy	0.89	0.87	0.69	0.71	0.79
	F-1	0.86	0.83	0.70	0.68	0.74
Scenario 2	Precision	0.74	0.80	0.41	0.61	0.41
	Recall	0.80	0.75	0.86	0.86	0.85
	Accuracy	0.74	0.75	0.39	0.58	0.38
	F-1	0.70	0.69	0.45	0.59	0.44
Scenario 3	Precision	0.60	0.76	0.29	0.59	0.70
	Recall	0.94	0.98	1.00	0.88	0.83
	Accuracy	0.59	0.79	0.37	0.58	0.74
	F-1	0.64	0.80	0.44	0.62	0.70
Scenario 4	Precision	0.25	0.25	0.25	0.30	0.25
	Recall	1.00	1.00	1.00	0.99	1.00
	Accuracy	0.25	0.25	0.25	0.37	0.28
	F-1	0.39	0.39	0.39	0.45	0.40

learning algorithms but real-world data for testing, in terms of the average precision, in this case regression tree performed best, followed by SVM. Results for the ensemble, RGP and stepwiselm algorithms in terms of precision were overall quite low. These three algorithms, however, slightly outperformed SVM and regression tree for the recall measure. Nevertheless, in terms of accuracy and F-measure, both SVM and Regression Tree stood out over the remaining three algorithms with significant difference.

Overall, the results for all the four measures in Scenario 2 were lower than those shown in Scenario 1. Our hypothesis behind this is related to the difference between the types of passenger traffic flow in the test cases that are based on theoretical traffic profiles and the ones obtained from the real installation. It might be possible that the theoretical passenger profiles do not explore areas which the real passenger profiles actually do. This could be the case, for instance, when the canteen or the bar is on a specific floor. The theoretical traffic profiles also make assumptions that might not hold for all offices building. For instance, the theoretical traffic profiles assume that there are lunch peaks from 12:00 to 15:00, where workers from an office building go to have lunch. Nevertheless, there might be companies and buildings where most of the workers have a continuous work day from 7:00 to 15:00. Thus, we could answer the first RQ as follows:

SVM and regression tree algorithms were the algorithms performing best when trained with theoretical passenger profiles based test inputs. Nevertheless, their performance decreased when trained with these profiles but later tests being executed with real test inputs.

The second RQ aims to analyze the performance of the different machine-learning algorithms when trained with real passenger data based test cases. Scenario 3 in Table II shows the results for the 4-fold cross validation when using the real passenger data based test cases both, for training and for testing. As can be seen, in this case, regression tree performed best in terms of precision, accuracy and F-measure. Furthermore, with a recall of 0.98, the regression tree algorithm was the second best, after ensemble. Nevertheless, the precision, accuracy and F-measure values for ensemble were all below 0.5, meaning that this algorithm had a high number of false positives.

Scenario 4 also used real passenger data based test cases to train the machine learning algorithms. Although all algorithms performed well in terms of recall, meaning that they produced none or a low number of false negatives, their results in terms of precision, recall and accuracy were below 0.3. This means that a high number of false positives were produced when following this strategy for training the algorithms but testing the dispatching algorithms with theoretical data.

The hypothesis in this case is similar to the one for RQ1. The traffic profiles obtained from the real building installation might not exercise areas or produce situations that are considered in the theoretical traffic profiles. This makes it difficult for the regression algorithms to accurately predict the reference AWT value.

The regression tree algorithm was the algorithm that stood out over the rest in Scenario 3. Nevertheless, its performance, as well as the rest of the algorithms, was quite low when training the algorithm with real passenger profile based test data and testing it with theoretical passenger profile based data.

C. Threats to Validity

Internal validity: A potential internal validity threat in our study might be related to the thresholds of the arbiter we designed, which are configurable. To reduce this threat, we discussed the parameters with domain experts to see which thesholds could be appropriate to consider a test as pass or fail. The selected machine-learning algorithms do also have some parameters. To reduce this threat we used the default parameters from the MATLAB framework for training the algorithms.

External validity: An external validity threat in our evaluation is related to using a single benchmark dataset based on test cases for testing dispatching algorithms. To reduce this threat, the dataset was obtained from actual test cases in Orona for testing dispatching algorithms. Furthermore, to avoid bias in the results, we did not use the same dataset for training an algorithm and for testing it, using the appropriate k-fold cross validation techniques in those scenarios where this was necessary (i.e., Exp. 1 and Exp. 3). Another external validity threat relates to the used case study. Although only a single case study was used, it is important to note that it is a real industrial case study, which provides a high degree of complexity to our evaluation. Furthermore, the used dispatching algorithm is the one which is most used in Orona's elevators.

V. Lessons Learned

By using this industrial case study and the experiment provided in Section 4, we derived the following lessons learned.

Lesson 1 – Training data: In order for the oracle to be accurate enough, the training data should be of the same type as used when testing the system. This means that when Orona uses theoretical data for testing their algorithms, they should also use theoretical test data for training DARIO. Conversely, if real data is used for testing their algorithms, they should use real passenger data for training DARIO.

Lesson 2 – Importance of real passenger data: In the study, we showed that the oracles performed best when using real data obtained from the real installation (even if they were trained only with three real passenger profiles). In other contexts, such as web-engineering, technologies like DevOps permit using data from operation at design-time to enhance software engineering processes (e.g., testing). The good performance of the proposed approach with field test data shows the importance of researching on adapting design-operation continuum techniques (e.g., DevOps) in the context of CPSs and in domains like elevation.

Lesson 3 – Uncertainty of the verdicts provided by DARIO: While the accuracy of DARIO with certain algorithms is relatively good, using these oracles in Orona increases the uncertainty in relation to the correctness of the verdicts. This might require at certain points re-executing tests in the regression test oracle to confirm verdicts. However, although this might increase the test cost, the test results can also be used to retrain the algorithms. This challenge is further exacerbated at operational level. A future direction could be to assess different metrics to measure the uncertainty and trustworthiness around the inferred oracles.

VI. Related Work

Despite having received significantly less attention than other software testing activities (e.g., test generation), the use of machine-learning algorithms to alleviate the test oracle problem is not new. A recent systematic survey performed by Durelly et al. identified a total of 10 studies where machine learning algorithms were used to construct oracles [15]. Similar to our approach, machine-learning algorithms were used to predict the expected outputs of the SUT in two short-papers [16], [17]. Specifically, Jin et al. [16] used artificial neural networks, whereas Singhal et al. used both regression trees and neural networks [17]. There are a few differences, however, between their approach and ours. Firstly, in their case, the approach is designed for unit testing, while our approach focuses on system-level testing. Secondly, their approach is based on classification, while our machine-learning algorithms are focused on regression to predict QoS reference values. Lastly, their evaluation is performed by using a toy example involving the triangle type problem (i.e., given the three sides of of a triangle, the oracle predicts which type of triangle it is), while ours has been performed by using a real-world industrial case study.

Our approach relies on simulation-based testing for verifying elevators dispatching algorithms at system level. In the context of simulation-based testing, there are recent studies that tackle the test oracle problem [4], [18]. For instance, Menghi et al. proposed a test oracle generation tool for Simulink models [4]. Their approach consists in a Domain Specific Language (DSL) with sufficient expressiveness to specify signal properties-based requirements. Later, a model-to-model transformation is performed in order to generate Simulink subsystems. Similar to our work, their oracles also provide a quantitative measure for the satisfaction degree of a requirement. However, our study is focused on generating a reference signal by a machine-learning algorithm trained with data from previously tested software versions, and later applying and arbitration mechanism. Although we could use their tool to generate oracles by specifying some requirements, this would be infeasible for the context of QoS measures, as inferring the relation between passenger traffic data and QoS measures (e.g., AWT) is nontrivial. Furthermore, this relation is highly dependent on the building installation characteristics, and would therefore require a manual change every time the dispatching algorithm is tested in a different context; In contrast, training DARIO with already available data is straightforward and fast.

Stocco et al. proposed a technique for testing self-driving cars which use deep neural networks to determine the driving parameters for the actuators of the vehicle [18]. Similar to our approach, their oracle employs simulation-based testing and determines a confidence value for the system at each step of the execution. However, the oracle they propose uses an unsupervised learning technique based on the camera images (i.e., the input) of the self-driving car, whereas DARIO uses supervised regression learning based on the QoS measures of a system of elevators (i.e., the test output in our context).

There are some studies that focus on testing software from elevators systems. Nicolas et al., proposed an encoder based on FPGAs for simulation-based testing of elevator controllers in real-time [19]; their goal was to test the position and speed of elevators. Sagardui et al., relied on model-based testing and feature models for testing configurable software systems in charge of controlling the doors of elevators [20]; in this paper the goal was to test refactored embedded code, and the non-refactored software acted as a golden oracle. In these cases the system was not the dispatching algorithm, but other software components of the elevators. In our previous paper, we used the technique Metamorphic Testing to test the dispatching algorithm of Orona [21], showing promising results. However, in this prior paper, the technique is mainly designed for short-scenario tests, whereas the technique shown in this paper is designed for long-scenario tests. While there are many studies in the field of elevators dispatching algorithms, where artificial intelligence algorithms adapted to this context are investigated (e.g., ant-colony optimization and neural networks [22], genetic algorithms [23]), to the best of our knowledge, this is the first study that proposes a method for testing them.

VII. Conclusion and Future Work

In this paper we have proposed DARIO, a test oracle that relies on machine-learning to automatically test elevators dispatching algorithms. Compared with the traditionally used

regression oracles, which have several disadvantages, DARIO trains machine-learning algorithms with previous test data. This training takes only a few seconds (always less than 3 seconds), whereas executing the regression test oracle takes minutes or hours at SiL (depending on the length of the test case), and hours or days at HiL (not being possible to correctly perform some tests, such as those involving HMI). In our evaluation, where an industrial dispatching algorithm from Orona was used, the accuracy of the proposed test oracle when labeling tests as PASS or FAIL ranged between 0.79 and 0.87, which is competent to transfer the tool to practitioners, although further investigations are required to enhance these results.

As future research lines, we would like to explore handling the uncertainty in oracles from different perspectives. Firstly, as explained in the lessons-learned section, using DARIO increases the uncertainty related to the correctness of the verdicts. In deep learning algorithms, Kim et al., used the surprise adequacy [24]. Similar metrics adapted to the used algorithms by DARIO could be employed to measure such uncertainty. Other related uncertainties could be those where the CPS is exposed to, especially in operation. For instance, a sensor might have noise and its data could not be fully reliable. In the future, we foresee to tackle these problems.

ACKNOWLEDGEMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement no. 871319. A. Arrieta, J. Ayerdi, M. Illarramendi and G. Sagardui are part of the Systems and Software Engineering research group of Mondragon Unibertsitatea (IT1326-19), supported by the Department of Education, Universities and Research of the Basque Government.

REFERENCES

[1] J. Ayerdi, A. Garciandia, A. Arrieta, W. Afzal, E. P. Enoiu, A. Agirre, G. Sagardui, M. Arratibel, and O. Sellin, "Towards a taxonomy for eliciting design-operation continuum requirements of cyber-physical systems," in *28th IEEE International Conference on Requirements Engineering, RE 2020, Zurich, Switzerland, 2020*, 2020.

[2] C. Menghi, S. Nejati, L. C. Briand, and Y. I. Parache, "Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification," in *International Conference on Software Engineering (ICSE)*, 2020.

[3] S. Nejati, K. Gaaloul, C. Menghi, L. C. Briand, S. Foster, and D. Wolfe, "Evaluating model testing and model checking for finding requirements violations in simulink models," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, 2019, pp. 1015–1025. [Online]. Available: https://doi.org/10.1145/3338906.3340444

[4] C. Menghi, S. Nejati, K. Gaaloul, and L. C. Briand, "Generating automated and online test oracles for simulink models with continuous and uncertain behaviors," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, 2019, pp. 27–38. [Online]. Available: https://doi.org/10.1145/3338906.3338920

[5] G. Barney and L. Al-Sharif, *Elevator traffic handbook: theory and practice*. Routledge, 2015.

[6] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for machine learning*. Cambridge University Press, 2020.

[7] G. Sagardui, J. Agirre, U. Markiegi, A. Arrieta, C. F. Nicolás, and J. M. Martín, "Multiplex: A co-simulation architecture for elevators validation," in *Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2017 IEEE International Workshop of*. IEEE, 2017, pp. 1–6.

[8] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?" in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 654–665.

[9] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Test generation and test prioritization for simulink models with dynamic behavior," *IEEE Trans. Software Eng.*, vol. 45, no. 9, pp. 919–944, 2019. [Online]. Available: https://doi.org/10.1109/TSE.2018.2811489

[10] A. Arrieta, S. Wang, U. Markiegi, A. Arruabarrena, L. Etxeberria, and G. Sagardui, "Pareto efficient multi-objective black-box test case selection for simulation-based testing," *Information & Software Technology*, vol. 114, pp. 137–154, 2019. [Online]. Available: https://doi.org/10.1016/j.infsof.2019.06.009

[11] A. Arrieta, S. Wang, A. Arruabarrena, U. Markiegi, G. Sagardui, and L. Etxeberria, "Multi-objective black-box test case selection for cost-effectively testing simulation models," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '18. New York, NY, USA: ACM, 2018, pp. 1411–1418. [Online]. Available: http://doi.acm.org/10.1145/3205455.3205490

[12] A. E. Genç, H. Sözer, M. F. Kıraç, and B. Aktemur, "Advisor: An adjustable framework for test oracle automation of visual output systems," *IEEE Transactions on Reliability*, 2019.

[13] W. K. Chan, J. C. Ho, and T. Tse, "Finding failures from passed test cases: Improving the pattern classification approach to the testing of mesh simplification programs," *Software Testing, Verification and Reliability*, vol. 20, no. 2, pp. 89–120, 2010.

[14] M.-L. Siikonen, "On traffic planning methodology," *Elevator technology*, vol. 10, pp. 267–274, 2000.

[15] V. H. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. Dias, and M. P. Guimaraes, "Machine learning applied to software testing: A systematic mapping study," *IEEE Transactions on Reliability*, vol. 68, no. 3, pp. 1189–1212, 2019.

[16] H. Jin, Y. Wang, N.-W. Chen, Z.-J. Gou, and S. Wang, "Artificial neural network for automatic test oracles generation," in *2008 International Conference on Computer Science and Software Engineering*, vol. 2. IEEE, 2008, pp. 727–730.

[17] A. Singhal and A. Bansal, "Generation of test oracles using neural network and decision tree model," in *2014 5th International Conference-Confluence The Next Generation Information Technology Summit (Confluence)*. IEEE, 2014, pp. 313–318.

[18] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *Proceedings of 42nd International Conference on Software Engineering*, ser. ICSE '20. ACM, 2020, p. 12 pages.

[19] C. F. Nicolas, I. Ayestaran, I. Martinez, and P. Franco, "Model-based development of an fpga encoder simulator for real-time testing of elevator controllers," in *2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2016, pp. 53–60.

[20] G. Sagardui, L. Etxeberria, J. A. Agirre, A. Arrieta, C. F. Nicolas, and J. M. Martin, "A configurable validation environment for refactored embedded software: An application to the vertical transport domain," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2017, pp. 16–19.

[21] J. Ayerdi, S. Segura, A. Arrieta, G. S. Arratibel, and M. Arratibel, "Qos-aware metamorphic testing: An elevation case study," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 104–114.

[22] J. Liu and Y. Liu, "Ant colony algorithm and fuzzy neural network-based intelligent dispatching algorithm of an elevator group control system," in *2007 IEEE International Conference on Control and Automation*. IEEE, 2007, pp. 2306–2310.

[23] B. Bolat, P. Cortés, E. Yalçin, and M. Alişverişçi, "Optimal car dispatching for elevator groups using genetic algorithms," *Intelligent Automation & Soft Computing*, vol. 16, no. 1, pp. 89–99, 2010.

[24] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1039–1049.