Test Case Generation for Simulink Models: An Experience from the E-Bike Domain

Michael Marzella[1], Andrea Bombarda[1][0000−0003−4244−9319], Marcello Minervini[1], Nunzio Marco Bisceglia[1][0009−0000−7152−1389], Angelo Gargantini[1][0000−0002−4035−0131], and Claudio Menghi[1,2][0000−0001−5303−8481]

[1] University of Bergamo, Bergamo, Italy
{m.marzella,n.bisceglia1}@studenti.unibg.it
{andrea.bombarda,marcello.minervini,angelo.gargantini, claudio.menghi}@.unibg.it
[2] McMaster University, Hamilton, Canada

Abstract. Building reliable Cyber-physical systems often requires engineers to search for software defects. Search-based software testing (SBST) is a standard technology that supports this activity. To increase practical adoption, industries need empirical evidence of the usefulness of SBST techniques on different benchmarks. To address this need, this replication study reports on our experience assessing the usefulness of SBST in generating failure-revealing test cases. Our study subject is within the electric bike (e-Bike) domain and concerns the software controller of an e-Bike motor. We assessed the effectiveness and efficiency of HECATE, an SBST framework for Simulink® models. HECATE successfully identified failure-revealing test cases in practical time. We present the lessons learned, the relevance of our results for industrial applications, and the improvement in the state of practice.

Keywords: Motor Control, E-Bikes, Model Development, Simulink®, Search-based Software Testing

1 Introduction

Simulink® [19] is a modeling and simulation language widely used by the Cyber-physical system (CPS) industry [27]. It is used by more than 60% of engineers for CPS design and in many domains, such as automotive [29]. Simulink® appeals to engineers, due to its graphical language. It enables engineers to model their systems by specifying their components and connections [19] and offers many add-ons with components targeting different problems.

Search-based software testing (SBST) is widely applied during the development of CPSs to check for software defects. It is used in many domains, including real-time, embedded, and safety-critical systems [1]. SBST tools automatically generate test cases to check for violations of requirements [18].

To increase the *industrial applicability* of SBST, it is paramount to empirically evaluate its effectiveness and provide practitioners with guidelines and lessons learned that can help them choose the appropriate tools and assess their level of maturity [1]. It is also necessary to assess whether SBST techniques scale to realistic development artifacts [1]. Indeed, despite being widely recognized as useful tools, the effectiveness of SBST test generators strongly depend on the application domain [17]. Different domains may require different properties from the SBST frameworks. The research and industrial communities widely recognize the need for replicable experiments and empirical data assessing the benefits of software engineering approaches in practice [1,11,26,30,32,36]. The need for replicating experiments is of particular importance for Simulink® models [6, 8, 39], as Simulink® projects and models are typically created and maintained in an industrial context and are usually not publicly available due to confidentiality agreements or license restrictions [5]. Therefore, access to these models is limited, making results hard (if not impossible) to replicate [7].

Several *SBST tools for Simulink®* are available in the literature (e.g., [13, 14, 31]). Many of the tools are compared by the falsification category [25] of the ARCH competition, an international competition of verification tools for CPSs. In this paper, we consider HECATE [14]. Unlike other existing tools, HECATE generates test cases specified as Test Sequences [22], which are automatically adapted to search for requirement violations, and Test Assessments [21], representing the requirements of interest. Since HECATE directly works with Test Sequences and Test Assessments, which are built-in components within Simulink Test™ [20], it does not require engineers to learn new frameworks to specify their test oracles and generate their test cases. This makes HECATE particularly suitable for industrial applications. For example, HECATE has been successfully applied to support the development of a cruise controller for an industrial simulator [15]. Nevertheless, HECATE is still primarily an academic tool. Replicating the experiments to assess HECATE on a different study subject provides insightful results to industrial practitioners [24, 38]. Therefore, they are pivotal for technology transfer and support the industrial adoption of HECATE.

This work focuses on the e-Bikes domain. The global e-Bike market size was USD 27.15 billion in 2022 and is projected to grow to USD 82.84 billion by 2030 [23]. We focus on the (software) controller of the electrical motor of the e-Bike. The software runs on 100% of e-Bikes [41] and is often designed in Simulink® [40]. It governs the motor's responsiveness to the speed demand and regulates the battery management. Ensuring software reliability and performance is essential: It affects user experience, safety, and battery life. Software testing is necessary to check for software failures. This activity is normally performed by physically testing the e-Bikes within different scenarios [34].

In this paper, we replicate previous experiments (from the automotive domain [15]) assessing the effectiveness of SBST with HECATE in generating failure-revealing test cases by considering our e-Bike study subject. We considered a complex model of an e-Bike and two controllers: the Buck hardware controller and the PWM software controller. These controllers must satisfy three requirements (functional, regulatory, and safety). We also considered six different testing scenarios obtained from different Parameterized Test Sequences. We con-

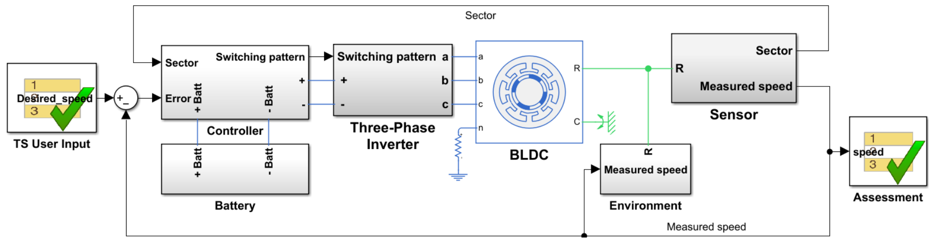

Fig. 1: Simulink® model for the e-Bike.

sidered two search algorithms: Uniform Random (UR) and Simulated Annealing (SA). Therefore, we conducted 72 experiments (2 models $\times$ 3 requirements $\times$ 6 Parameterized Test Sequences $\times$ 2 search algorithms). For each experiment, we ran HECATE and checked whether it could generate a failure-revealing test case. Our results show that HECATE could effectively generate failure-revealing test cases for $\approx 83\%$ (60 out of 72) of our experiments. We confirmed the failures we found by discussing them with the engineer who developed the models. HECATE required on average $5'\,51''$ ($min = 59''$, $max = 2h\,0'\,18''$, $StdDev = 8'\,54''$). We critically analyze our results, present the lessons learned, and discuss the relevance of our results for industrial applications. We also compare our results with those previously obtained in the automotive domain [15].

This paper is organized as follows. Section 2 presents our study subject. Section 3 describes HECATE. Section 4 presents our results, and Section 5 discusses them. Section 6 outlines the related work. Section 7 presents our conclusions.

2 E-Bike Study Subject

We describe the controlled system and its requirements (Section 2.1), and the two controllers (Section 2.2).

2.1 The Controlled System

Figure 1 presents the Simulink® model of our e-Bike study subject. The controlled system consists of the following components.

The *User Inputs* component collects the inputs from the rider. Its output (*Desired Speed*) models the speed selected by the rider. The desired speed is used to compute the *Error*, i.e., the difference between the *Desired speed* and the *Measured speed*, which is one of the inputs of the *Controller* of the e-Bike.

The *Environment* component represents external forces (e.g., aerodynamic drag). The input is the *Measured speed* of the e-Bike, and the output is a signal R simulating the effects of external forces.

The *Brushless Direct Current (BLDC) Motor* component converts the electrical energy into rotational motion. The inputs of the BLDC are the currents

Table 1: Requirements for the e-Bike.

ID	Description
R1	Motor speed shall always be positive or zero.
R2	Motor speed shall always be lower than 170 rpm.
R3	Motor speed shall not exceed that requested by the rider.

applied to the three phases (a, b, c) and the neutral phase (n). The outputs are the torque generated by the motor for the rotor (R) and the motor case (C).

The *Sensor* component monitors the torque of the e-Bike. It returns the active *Sector* of the BLDC and the *Measured speed*.

The *Battery* component stores electrical energy. A negative ($-Batt$) and positive ($+Batt$) current recharges and accesses the energy from the battery.

The *Inverter* component converts the direct current into alternating current and regulates the electrical energy that flows from the battery to the motor. When the e-Bike decelerates (e.g., during braking), the rotor keeps rotating due to the vehicle's inertia, and the produced energy ($-Batt$) recharges the battery. Otherwise, it flows from the battery to the inverter to help the rider ($+Batt$). The inverter acts on the battery and the motor depending on a direct current (DC) signal (*Switching pattern*) received from the *Controller*.

The *Controller* component selects the *Switching Pattern* based on the *Error* signal and the active sector (*Sector*).

Engineers designed the e-Bike controller (*Controller*) to satisfy the e-Bike requirements from Table 1.

Requirement R1 is a *functional* requirement: It demands the speed of the motor not to be negative. During braking phases, the motor shall rotate in the same direction while regenerating energy to be stored in the battery: A negative speed is not considered since the e-Bike is assumed to move only forward. The braking process, therefore, reduces the positive speed of the e-Bike.

Requirement R2 is a *regulatory* requirement: It enables electric bikes to assist riders only below 25 km/h (i.e., 170 rpm wheel speed, considering a 28 $inch$ wheel), as mandated by most European countries [37].

Requirement R3 is a *safety* requirement: The motor speed shall not exceed the speed requested by the rider.

2.2 Software Controllers

In this replication study, we considered two controllers for the e-Bike: The Pulse Width Modulation (PWM) software controller and the Buck hardware controller. Experts from electrical engineering developed these controllers in a project on improving "green" mobility solutions, including e-Bikes, and involving several companies, such as Brembo [33] and Pirelli [35]. Engineers have been developing these models to determine which architecture is more efficient. The development of these models took approximately 100 hours each (including testing) [10].

Step	Transition	Next
step_1 getSimTime() speed=Hecate_sp/5*getSimTime();	after(5,'sec')	step_2
step_2 speed=Hecate_sp	after(5,'sec')	step_3
step_3 speed=Hecate_sp-Hecate_sp/5*(getSimTime()-10);	after(5,'sec')	step_4
...	...	..
step_7 speed=Hecate_sp-Hecate_sp/5*(getSimTime()-30);	after(5,'sec')	

(a) Parameterized Test Sequence Block.

Step	Transition	Next
step_1	after(6,'sec')	Sp_H_1
Sp_H_1 verify(speed<=11)	after(4,'sec')	step_3
step_3	after(16,'sec')	Sp_H_2
Sp_H_2 verify(speed<=11)	after(4,'sec')	step_1

(b) Test Assessment Block.

Fig. 2: Test Blocks for our e-Bike model.

In this work, the e-Bike engineers provided us with two *intermediate versions* of the PWM and Buck controllers. Typically, engineers extensively test their controllers before deployment to check for failures. However, since these models are intermediate, they were only assessed by considering a limited set of test cases encoding standard profiles for the desired speed.

3 HECATE

HECATE is an SBST tool for Simulink® models. Unlike existing SBST tools, HECATE supports Simulink® Test blocks (i.e., Test Sequences and Test Assessments blocks). HECATE identifies failure-revealing test cases represented as Test Sequences using the information from Test Assessments blocks. Test blocks are embedded within the Simulink® model. For example, engineers add a Test Sequence to the model from Figure 1 by replacing the block *User Input* with the Test Sequence block *TS User Input* that generates a *Desired speed* signal. They also add the Test Assessment block (*Assessment*) that receives the measured speed of the vehicle as input. Figure 2 details the Test Sequence (Figure 2a) and a Test Assessment block (Figure 2b) from Figure 1.

A *Test Sequence* defines the test case's input. It consists of *steps* connected by *transitions*. The fragment of the Test Sequence from Figure 2, contains four steps (i.e., `step_1`, `step_2`, `step_3`, and `step_7`). Each test step defines the values to be assumed by the output signals of the Test Sequence. For example, the `step_1` from Figure 2 specifies that the value of the *speed* is *Hecate_sp/5*getSimTime()*. Transitions define the test steps change: They are labeled with a Boolean formula defining the condition for the transitions to be fired. When a transition is fired, the system reaches the step identified by the column **Next**. For example, the transition from the first row of the Test Sequence in Figure 2a specifies that the test switches from `step_1` to `step_2` after 5s. When the Simulink® model is executed, the Test Sequence generates a signal for its outputs.

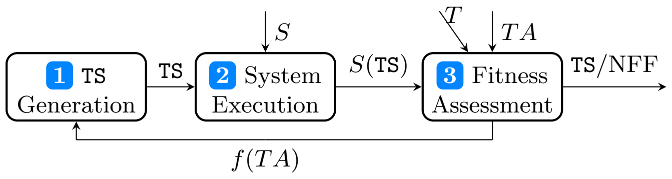

Fig. 3: Overview of the HECATE framework.

A *Test Assessment* follows the same structure as a Test Sequence. For example, the Test Assessment from Figure 2b contains four steps (i.e., `step_1` and `Sp_H_1`, `step_3` and `Sp_H_2`). Unlike Test Sequences, Test Assessment blocks allow engineers to use the `verify` construct. This construct verifies whether certain conditions are met when the Test Assessment is in the corresponding step. For example, when the Test Assessment is within the test step `Sp_H_1`, the Test Assessment Block verifies whether the condition `speed` ≤ 11 holds.

A *Test Case* consists of a Test Sequence block and a Test Assessment block. The Test Sequence block specifies input signals supplied to the Simulink® simulator. The model is then run with these inputs to simulate the corresponding Test Sequence. The Test Assessment block monitors the model's output signals to determine whether any of its `verify` expressions are violated. Typically, a Test Assessment is associated with one or more system requirements. For example, the Test Assessment in Figure 2b has been written to check the requirement R3 from Section 2.1. Engineers can inspect the satisfaction of these conditions using an appropriate GUI. If (at least) one of the conditions of the Test Assessment is violated, the test case is failure-revealing.

HECATE [14] extends this testing framework by supporting SBST. It enables the automatic generation of Test Sequences driven by a fitness function generated from the Test Assessment block. Specifically, HECATE requires engineers to extend their Test Sequences into Parameterized Test Sequences.

A *Parameterized Test Sequence* is a Test Sequence in which some values are parameters that are automatically assigned to values by HECATE. Figure 2a shows an example of a Parameterized Test Sequence. HECATE can assign different values to the parameter `Hecate_sp` to generate many test cases. To generate realistic Test Sequences, engineers can specify lower and upper bounds for the search parameters. For example, engineers specified that the lower and upper bounds for the values of the parameter `Hecate_sp` are 0 and 170 rpm.

HECATE iteratively performs the steps from Figure 3:

1 *Test Sequence Generation*: Generates a new (population of) Test Sequence (TS) by assigning values to the parameters in the Parameterized Test Sequence.

2 *System Execution*: Executes the system model S by providing the input generated from the Test Sequence (TS) and generates the output $S(\texttt{TS})$.

3 *Fitness Assessment*: Assesses a fitness function obtained from the Test Assessment w.r.t. the output $S(\texttt{TS})$. HECATE assesses if the fitness value is below a threshold level and, thus, is failure-revealing. Details on how the the fitness func-

Table 2: Experimental results for the e-Bike case study.

		UR									SA								
		TA_R1			**TA_R2**			**TA_R3**			**TA_R1**			**TA_R2**			**TA_R3**		
	PTS	**F**	**$\bar{S}$**	**$\hat{S}$**	**F**	**$\bar{S}$**	**$\hat{S}$**	**F**	**$\bar{S}$**	**$\hat{S}$**	**F**	**$\bar{S}$**	**$\hat{S}$**	**F**	**$\bar{S}$**	**$\hat{S}$**	**F**	**$\bar{S}$**	**$\hat{S}$**
PWM	*t-pyr-0*	10	1.6	1.0	10	9.6	4.0	10	1.9	1.5	10	1.8	1.5	10	8.2	7.5	10	1.8	1.0
	t-pyr-85	10	7.6	4.0	10	3.0	3.0	10	1.0	1.0	10	7.6	4.0	10	3.9	4.0	10	1.0	1.0
	t-pyr-130	10	10.6	8.5	10	1.1	1.0	10	1.0	1.0	10	4.4	3.0	10	1.8	1.0	10	1.0	1.0
	r-pulse-0	10	8.9	6.0	10	8.6	8.5	10	1.6	1.0	10	12.9	12.5	10	4.7	4.0	10	2.7	2.0
	r-pulse-85	10	14.6	12.5	10	3.0	2.5	10	1.0	1.0	10	15.2	13.5	10	2.6	2.0	10	1.0	1.0
	r-pulse-130	10	6.7	4.0	10	2.2	1.5	10	1.0	1.0	10	10.2	8.0	10	2.8	1.5	10	1.0	1.0
Buck	*t-pyr-0*	0	–	–	4	28.8	26.0	10	8.1	3.0	0	–	–	10	15.3	14.0	10	2.9	2.0
	t-pyr-85	0	–	–	5	17.6	22.0	10	4.3	4.0	0	–	–	10	19.5	18.0	10	3.1	2.5
	t-pyr-130	0	–	–	10	1.9	1.0	10	2.1	1.5	0	–	–	10	3.7	2.0	10	2.3	1.0
	r-pulse-0	0	–	–	10	2.9	2.0	10	7.3	7.5	0	–	–	10	4.1	4.0	10	7.1	8.0
	r-pulse-85	0	–	–	10	2.4	2.0	10	1.6	1.5	0	–	–	10	5.9	4.0	10	1.7	1.0
	r-pulse-130	0	–	–	10	2.7	3.0	10	1.0	1.0	0	–	–	10	2.6	2.5	10	1.0	1.0

tion is built and evaluated can be found in the corresponding publication [13]. HECATE terminates if a failure-revealing test is found or the maximum time T is reached.

4 Evaluation

In this replication study, we consider the following research questions:

RQ1: How *effective* is SBST in generating failure-revealing test cases?
RQ2: How *efficient* is SBST in generating failure-revealing test cases?

4.1 Experimental Setup

To assess the effectiveness of HECATE, we performed 72 experiments. Each experiment was obtained by considering one of the two models, one of the three Test Assessments, and one of the six Parameterized Test Sequences. Each experiment was repeated twice, considering two search algorithms: Uniform Random (UR) and Simulated Annealing (SA). The two models were the Simulink® models of the e-Bike obtained by considering the PWM and Buck controllers from Section 2.2. The three Test Assessments (TA_R1, TA_R2, TA_R3) encode the requirements (functional, regulatory, and safety, respectively) from Section 2.1. We considered six Parameterized Test Sequences producing truncated pyramidal and rectangular pulse behavior (*t-pyr-0*, *t-pyr-85*, *t-pyr-130*, *r-pulse-0*, *r-pulse-85*, *r-pulse-130*). We specifically designed these Parameterized Test Sequences to consider potential uses of e-Bikes in urban environments, where frequent acceleration and deceleration by riders due to traffic is common. As done in the study we replicate [15], we tested all requirements on all model versions: Each experiment was obtained by selecting one of two models, one of three requirements, one of six Parameterized Test sequences, and one of the search algorithms. Thus, we ran 72 experiments (2 models $\times$ 3 Test Assessment $\times$ 6 Parameterized Test Sequences $\times$ 2 search algorithms) in total.

For each experiment, we ran HECATE by setting the maximum number of search iterations to 50. Every run was repeated 10 times to account for the stochastic nature of the algorithms [2]. We executed experiments on an Intel(R) Core(TM) i7-9750H CPU 2.60 GHz, 6 cores, 12 MB SmartCache, 16 GB RAM. We recorded which of the 10 runs returned a failure-revealing test case.

Table 2 summarizes our results. Each row encodes the model and the Parameterized Test Sequence (**PTS**) selected for the experiment. The table is divided into two parts, containing the results for the UR and SA. The three vertical portions of the table identify the Test Assessment blocks (**TA_R1**, **TA_R2**, and **TA_R3**). For each experiment, the table reports the falsification rate (**F**), i.e., the number of runs (over a total of 10 runs) in which HECATE could find a failure-revealing test case, the average ($\hat{\mathbf{S}}$), and the median ($\tilde{\mathbf{S}}$) number of iterations required to identify that test case. Cells with gray background mark scenarios in which the two search algorithms significantly differ in terms of falsification rate.

4.2 Effectiveness (RQ1)

We discuss the results (Table 2) for the PWM and Buck controllers.

Functional Requirement R1. For the *PWM controller*, HECATE could generate a failure-revealing test case for all the experiments. Therefore, the PWM controller did not ensure that the speed was always greater than or equal to zero. For the *Buck Controller*, HECATE could not generate any failure-revealing test case for all the experiments.

Regulatory Requirement R2. For both the *PWM* and the *Buck* controller, HECATE generated a failure-revealing test case for all the experiments, showing that the e-Bike violates the regulatory requirement.

Safety Requirement R3. For both the *PWM* and the *Buck* controller, HECATE generated a failure-revealing test case for all the experiments, showing that the e-Bike violates the safety requirement.

Uniform Random (UR) vs Simulated Annealing (SA). SA performed better than UR: They both showed a 10/10 falsification rate in all the experiments except in the two cases (cells marked with gray background in Table 2) where UR showed a falsification rate of 4/10 and 5/10 while SA reached 10/10.

Expert feedback. The engineer who developed the models analyzed the results of our experiments and confirmed the failures we discovered.

For requirement R1, the engineer confirmed that the response to the reference signal is unstable: The controller (in some conditions) continuously switches between the two algorithms (i.e., for motoring and braking functions), losing stability and reaching negative speed.

For requirement R2, the engineer confirmed the limitations in the tracking speed during the acceleration and braking phases. For the acceleration phase, the PWM problem was caused by the comparison between the vehicle inertia, the motor's power, and the torque. The Buck controller reached high speed in the first step response due to some issues in the PI (Proportional Integral) controller in the speed loop. A more precise parameter tuning could fix this problem. During deceleration, the braking force was not sufficient to track the reference speed. The engineer confirmed that the e-Bike mechanical brakes were not considered in the models. They will be added in future versions of the model.

For requirement R3, the expert confirmed that the PWM speed tracking was not accurate due to the instability caused by switching between the two different algorithms for motoring and braking functions.

Our e-Bike model represents the high-level design developed to perform a preliminary comparison of the two controllers. Therefore, problems were expected.

Summary. HECATE could reveal a failure-revealing test case for all 36 experiments of the PWM controller and 24 experiments of the Buck controller. The engineer who developed the model confirmed the test cases as failure-revealing.

Effectiveness — RQ1

HECATE effectively generated failure-revealing test cases for $\approx$ 83% (60 out of 72) of our experiments.

4.3 Efficiency (RQ2)

To answer RQ2, we evaluated the efficiency of HECATE by analyzing the time required to detect a failure-revealing test case for each version of the model and requirement from Section 2.1. The boxplot from Figure 4 reports our results of our 10 runs. It does not include the Buck controller and the requirement R1, since HECATE could not produce any failure-revealing test case. Diamonds depict the average, and lines in boxes represent the median.

For each run on the PWM controller, regardless of the algorithm, HECATE required, on average, $9'\,6''$ ($min = 1'\,2''$, $max = 45'\,2''$, $StdDev = 8'\,23''$) for R1, $6'\,14''$ ($min = 59''$, $max = 2h\,0'\,18''$, $StdDev = 12'\,57''$) for R2, and $1'\,36''$ ($min = 1'\,1''$, $max = 7'\ 36''$, $StdDev = 1'\,10''$) for R3. For the Buck controller, regardless of the algorithm, it required, on average, $8'\,24''$ ($min = 1'\,4''$, $max = 56'\,59''$, $StdDev = 9'\,57''$) for R2, and $4'\ 8''$ ($min = 1'\,4''$, $max = 35'\,29''$, $StdDev = 4'\,59''$) for R3.

Uniform Random (UR) vs Simulated Annealing (SA). UR and SA required on average $6'\,4''$ (min=$1'\,1''$, max=$2h\,0'\,18''$, std=$10'\,49''$) and $5'\,38''$ ($min = 59''$, $max = 42'\,23''$, $StdDev = 6'\,34''$) to identify the failure revealing test cases. The Wilcoxon Rank Sum Test, performed with significance level $\alpha =$ 5%, shows that there is no statistical evidence that the number of iterations required by the UR and the SA algorithms belongs to two different populations. To compute the failure-revealing test cases, HECATE required, on average, $5'\,51''$ (min=$59''$, max=$2h\,0'\,18''$, $StdDev$=$8'\,54''$) across all failure-revealing runs.

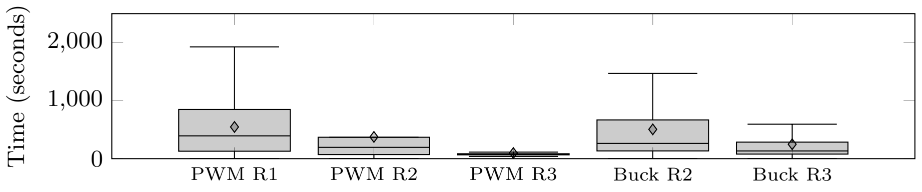

Fig. 4: Time required to generate the test cases for our benchmark.

Efficiency — RQ2

HECATE required on average $5'\,51''$ (min=$59''$, max= $2h\,0'\,18''$, $StdDev$= $8'\,54''$) to compute failure-revealing test cases of our models.

5 Discussion

In this paper, we replicate a study performed on an automotive cruise controller [15]. We applied the same methodology to an e-Bike study subject to assess whether previously reported findings generalize beyond the original domain. We present lessons learned (Section 5.1) and threats to the validity (Section 5.2).

5.1 Lessons Learned

The four main lessons we learned from this replication study are as follows:

L1 (Modeling). E-Bike engineers typically develop tests for assessing their models by manually defining inputs and visually inspecting the models' outputs. The engineer who developed the two models confirmed that the proposed Test Sequence and Test Assessment blocks helped reflect on plausible inputs and formalize the requirements of their system. Therefore, the engineer confirmed the usefulness of the test inputs and requirements formalization.

L2 (Testing Procedure). The outputs provided by the proposed testing framework helped the engineer identify flaws within the system design (Section 4.2). The time required to compute the failure-revealing test cases was practical for industrial applications (Section 4.3). Indeed, while some runs were time-consuming, our case study pertains to safety-critical systems, where the allocated time for testing is generally significant to prevent catastrophic failures.

L3 (Comparison of Solutions). HECATE was beneficial for assessing different controllers: We could assess when the PWM and Buck worked properly. Considering these lessons, the engineer is now using HECATE to automate their testing procedure. The engineer is also interested in evaluating the failure-revealing test case on the physical platform, which they are finalizing.

L4 (Generalizability of Results). Both our results and the ones for the previous study on the automotive cruise controller [15] confirm that HECATE is effective in finding failure-revealing test cases. However, the falsification rate of the original case study ($\approx 67\%$) is lower than the one we obtained ($\approx 83\%$). This difference may derive from system maturity: The e-Bike controller is still under development, while the cruise controller model is more stable. Interestingly, despite the higher falsification rate, our experiments required more time to find failure-revealing test cases than the ones from the original work ($5'\,51''$ vs $4'\,06''$). This difference suggests that domain-specific features influence SBST performance. It indicates that while the approach generalizes conceptually, practitioners should anticipate domain-specific characteristics.

Our results are relevant for *industrial application*. Technology transfer activities require empirical studies that assess the effectiveness of different technologies. Our results confirm the effectiveness of HECATE in detecting failure-revealing test cases for a complex model from a project with industrial partners.

Our results significantly *improved the state of practice*. Before our study, engineers manually developed test cases. Additionally, during the preliminary development stage of the proposed models, engineers tested their behavior by considering profiles for smooth, limited changes in the desired speed. This project improved the state of practice by showing the benefits of search-based testing. Our results benefit other practitioners who are considering the SBST technology.

For the *generalizability of our results*, we do not expect that applying HECATE to other systems will return exactly the same percentage of failure-revealing test cases. This value depends on the model, its development stage, and the Test Blocks. However, our results are *general*: They confirm existing results from the research literature obtained in other domains (e.g., space [31], automotive [12], biomedical [4], medical [9]), and previous results on HECATE [14,15].

The goal of this work *is not* to show the superiority of HECATE compared to other techniques, and therefore compare it with a baseline, but to assess whether its use is beneficial. Comparing different search-based solutions is out of scope: An extensive comparison of SBST techniques cannot be based on a single model and should rely on a larger set to have statistically significant results.

Simulink® Design Verifier (SLDV) is the only tool that we could have used as a baseline since it is the only one that can generate test cases from Test Assessment blocks [28]. However, SLDV can not process our model since it does not support some of its blocks (e.g., the Integrator block).

HECATE can be configured with different search algorithms. Our evaluation considered UR and SA. For our case study, the effectiveness and efficiency of these two algorithms were comparable. This result confirms that random and non-random solutions are complementary [3]: Random solutions are effective when failures are not difficult to find (like early development stages), while non-random solutions (e.g., SA) are more useful when failures are difficult to find since the use of a fitness metric to guide the search process is beneficial.

Our results are based on a single case study. Yet, they are significant. First, the development of our study subjects took considerable time. Second, finding industries sharing their models is complicated since they are usually part of their core business. Third, performing the testing activity for our study subject requires significant know-how: Understanding the model and its requirements, and discussing our results with the expert took considerable time (months).

Our results are based on the opinion of a single expert. Finding knowledgeable experts, especially in niche sectors (like e-Bikes), is complicated. In our case, the expert was the model developer: This provides significant value to our findings.

5.2 Threats to Validity

The Test Blocks we considered could threaten the *external validity* of our results. However, the requirements and the Test Sequences were defined in collaboration with the engineer who developed the model. The selection of our study subject could threaten the *external validity* of our results. We acknowledge that possible missing physical elements or the maturity level of our models can impact the validity of our experiments and conclusions. However, it is a representative model developed by expert engineers in a project with industrial partners.

The values we selected for the configuration parameters of HECATE, as well as the choice of the SBST algorithm, could threaten the *internal validity* of our results. To mitigate this threat, we reused the default values provided by HECATE.

6 Related Work

Numerous studies evaluated the effectiveness of SBST in identifying failure-revealing test cases for CPS development [9, 15]. In this work, we assessed the usefulness of SBST by considering the motor controller for an e-Bike, analyzing two different implementations, namely one based on a Buck converter and one controlled by using the PWM strategy.

Testing e-Bike motor controllers is of utmost importance. However, this activity is commonly performed by physically testing electric bikes or their components with different loads, pedaling profiles, roads, and scenarios [34]. Instead, in this work, we used HECATE [14] for model-in-the-loop testing.

7 Conclusion

This replication paper assesses the usefulness of HECATE in generating failure-revealing test cases for an e-Bike study subject. HECATE successfully identified failure-revealing test cases (confirmed by the engineer who developed the model) in practical time. We critically reflected on our results, presented lessons learned, and discussed their relevance for industrial applications. By repeating the original SBST experiment in a different context, this work strengthens the empirical foundation for search-based testing with Simulink® models and highlights the importance of multi-domain replication studies.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Data Availability A replication package with our data, results, and scripts is publicly available [16].

Acknowledgment

The work of Andrea Bombarda is supported by PNC - ANTHEM - PNC0000003 – CUP: B53C22006700001. The work of Nunzio Marco Bisceglia and Claudio Menghi was in part supported by the European Union – Next Generation EU. "Sustainable Mobility Center (Centro Nazionale per la Mobilitá Sostenibile - CNMS)", M4C2 – Investment 1.4, Project Code CN_00000023. The work of Claudio Menghi was in part supported by projects SERICS (PE00000014) under the NRRP MUR program and GLACIATION (101070141).

References

1. Ali, S., Briand, L.C., Hemmati, H., Panesar-Walawege, R.K.: A systematic review of the application and empirical investigation of search-based test case generation. IEEE Transactions on Software Engineering **36**(6), 742–762 (2010). https://doi.org/10.1109/TSE.2009.52
2. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: 2011 33rd International Conference on Software Engineering (ICSE). pp. 1–10 (2011). https://doi.org/10.1145/1985793.1985795
3. Arcuri, A., Iqbal, M.Z., Briand, L.: Random testing: Theoretical results and practical implications. IEEE transactions on Software Engineering **38**(2), 258–277 (2011)
4. Ayesh, M., Mehan, N., Dhanraj, E., et al.: Two simulink models with requirements for a simple controller of a pacemaker device. In: International Workshop on Applied Verification of Continuous and Hybrid Systems. vol. 90, pp. 18–25 (2022)
5. Badreddin, O., Lethbridge, T.C., Elassar, M.: Modeling practices in open source software. In: Open Source Software: Quality Verification. pp. 127–139. Springer (2013)
6. Boll, A., Brokhausen, F., Amorim, T., Kehrer, T., Vogelsang, A.: Characteristics, potentials, and limitations of open-source simulink projects for empirical research. Software and Systems Modeling **20**(6), 2111–2130 (2021)
7. Boll, A., Kehrer, T.: On the replicability of experimental tool evaluations in model-based development: Lessons learnt from a systematic literature review focusing on matlab/simulink. In: Systems Modelling and Management. p. 111–130. Springer (2020). https://doi.org/10.1007/978-3-030-58167-1_9
8. Boll, A., Vieregg, N., Kehrer, T.: Replicability of experimental tool evaluations in model-based software and systems engineering with matlab/simulink. Innovations in Systems and Software Engineering **20**(3), 209–224 (2024)
9. Bombarda, A., Bonfanti, S., Gargantini, A.: Automatic test generation with asmeta for the mechanical ventilator milano controller. In: Testing Software and Systems. p. 65–72. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-04673-5_5
10. Corti, F., Minervini, M., Giangrande, P., Reatti, A., Malighetti, P., Pugi, L.: Simulink-based simulation of electric bicycle dynamics and regenerative braking

for battery state of charge assessment. In: Mediterranean Electrotechnical Conference (MELECON). p. 803–808. IEEE (Jun 2024). https://doi.org/10.1109/melecon56669.2024.10608778

11. Dyba, T., Kitchenham, B.A., Jorgensen, M.: Evidence-based software engineering for practitioners. IEEE software **22**(1), 58–65 (2005)
12. Fainekos, G.E., Sankaranarayanan, S., Ueda, K., Yazarel, H.: Verification of Automotive Control Applications Using S-TaLiRo. In: 2012 American Control Conference (ACC). pp. 3567–3572 (2012). https://doi.org/10.1109/ACC.2012.6315384
13. Formica, F., Fan, T., Menghi, C.: Search-based software testing driven by automatically generated and manually defined fitness functions. ACM Transactions on Software Engineering and Methodology **33**(2) (2023). https://doi.org/10.1145/3624745
14. Formica, F., Fan, T., Rajhans, A., Pantelic, V., Lawford, M., Menghi, C.: Simulation-based testing of simulink models with test sequence and test assessment blocks. IEEE Transactions on Software Engineering **50**(2), 239–257 (Feb 2024). https://doi.org/10.1109/tse.2023.3343753
15. Formica, F., Petrunti, N., Bruck, L., et al.: Test case generation for drivability requirements of an automotive cruise controller: An experience with an industrial simulator. In: Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 1949–1960. ESEC/FSE, ACM (2023). https://doi.org/10.1145/3611643.3613894
16. FOSELAB: Experimental Results. https://github.com/foselab/HECATE-Bikes (2024), accessed: July 21, 2025
17. Harman, M., Jia, Y., Zhang, Y.: Achievements, open problems and challenges for search based software testing. In: IEEE International Conference on Software Testing, Verification and Validation (ICST). pp. 1–12 (2015). https://doi.org/10.1109/ICST.2015.7102580
18. Harman, M., Jones, B.F.: Search-Based Software Engineering. Information and Software Technology **43**(14), 833–839 (2001). https://doi.org/10.1016/s0950-5849(01)00189-6
19. Inc., M.: Design. Simulate. Deplo. https://www.mathworks.com/help/matlab/add-ons.html (2024), accessed: July 21, 2025
20. Inc., M.: Simulink test develop, manage, and execute simulation-based tests. https://www.mathworks.com/products/simulink-test.html (2024), accessed: July 21, 2025
21. Inc., M.: Test assessment. https://www.mathworks.com/help/sltest/ref/testassessment.html (2024), accessed: July 21, 2025
22. Inc., M.: Test sequence basics. https://www.mathworks.com/help/sltest/ug/introduction-to-test-sequences.html (2024), accessed: July 21, 2025
23. Insights, F.B.: E-bike drive unit market size, share & covid-19 impact analysis, by product type (mid-drive motors and hub motors), by application (oem and aftermarket), and regional forecasts, 2023-2030. https://www.fortunebusinessinsights.com/e-bike-drive-unit-market-107520 (2024), accessed: July 21, 2025
24. Juristo, N., Gómez, O.S.: Replication of software engineering experiments. Empirical Software Engineering and Verification: International Summer Schools, LASER 2008-2010, Revised Tutorial Lectures pp. 60–88 (2012)
25. Khandait, T., Formica, F., Arcaini, P., et al.: Arch-comp 2024 category report: Falsification. In: International Workshop on Applied Verification for Continuous and Hybrid Systems. EPiC Series in Computing, vol. 103, pp. 122–144. EasyChair (2024). https://doi.org/10.29007/hgfv

26. Kitchenham, B., Dyba, T., Jorgensen, M.: Evidence-based software engineering. In: International Conference on Software Engineering. pp. 273–281 (2004). `https://doi.org/10.1109/ICSE.2004.1317449`
27. Liebel, G., Marko, N., Tichy, M., Leitner, A., Hansson, J.: Model-based engineering in the embedded systems domain: An industrial survey on the state-of-practice. Software & Systems Modeling **17**, 91–113 (2018)
28. MathWorks: Generate and Export Tests from Requirements Table Blocks (September 2024), `https://www.mathworks.com/help/slrequirements/ug/export-generated-tests-from-requirements-table.html`
29. Matinnejad, R., Nejati, S., Briand, L.C., Bruckmann, T.: Automated Test Suite Generation for Time-Continuous Simulink Models. In: International Conference on Software Engineering. p. 595–606. ICSE, ACM (2016). `https://doi.org/10.1145/2884781.2884797`
30. Melo, S.M., Carver, J.C., Souza, P.S., Souza, S.R.: Empirical Research on Concurrent Software Testing: A Systematic Mapping Study. Information and Software Technology **105**, 226–251 (2019)
31. Menghi, C., Nejati, S., Briand, L., Parache, Y.I.: Approximation-Refinement Testing of Compute-Intensive Cyber-Physical Models: An Approach Based on System Identification. In: International Conference on Software Engineering. p. 372–384. ICSE, ACM/IEEE (2020). `https://doi.org/10.1145/3377811.3380370`
32. Mezhuyev, V., Al-Emran, M., Ismail, M.A., Benedicenti, L., Chandran, D.A.P.: The acceptance of search-based software engineering techniques: An empirical evaluation using the technology acceptance model. IEEE Access **7**, 101073–101085 (2019). `https://doi.org/10.1109/ACCESS.2019.2917913`
33. N.V., B.: Brembo homepage. `https://www.brembo.com/en/` (2024), accessed: July 21, 2025
34. Parastiwi, A., Al-Akbary, P.C.M., Safitri, H.K.: Analysis and Testing of DC Motor Control System for Electric Bike. Conference Series: Materials Science and Engineering **732**(1), 012055 (2020). `https://doi.org/10.1088/1757-899x/732/1/012055`
35. Pirelli: Pirelli homepage. `http://www.pirelli.com` (2024), accessed: July 21, 2025
36. Sayyad, A.S., Goseva-Popstojanova, K., Menzies, T., Ammar, H.: On parameter tuning in search based software engineering: A replicated empirical study. In: International Workshop on Replication in Empirical Software Engineering Research. p. 84–90. RESER, IEEE (2013). `https://doi.org/10.1109/RESER.2013.6`
37. Schepers, P., Wolt, K.K., Fishman, E.: The safety of e-bikes in the netherlands. International Transport Forum Discussion Paper 2018-02, Paris (2018). `https://doi.org/10.1787/21de1ffa-en`
38. Shepperd, M., Ajienka, N., Counsell, S.: The Role and Value of Replication in Empirical Software Engineering Results. Information and Software Technology **99**, 120–132 (2018)
39. Shrestha, S.L., Chowdhury, S.A., Csallner, C.: Replicability Study: Corpora For Understanding Simulink Models & Projects . In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 1–12. IEEE (2023). `https://doi.org/10.1109/ESEM56168.2023.10304867`
40. Stories, M.C.: Bosch ebike systems develops electric bike controller with model-based design. `https://www.mathworks.com/company/user_stories/bosch-ebike-systems-develops-electric-bike-controller-with-model-based-design.html` (2024), accessed: July 21, 2025
41. Times, E.: Software takes ebikes to new heights. `https://www.eetimes.eu/software-takes-ebikes-to-new-heights/` (2024), accessed: July 21, 2025