Efficient Online Testing for DNN-Enabled Systems using Surrogate-Assisted and Many-Objective Optimization

Fitash Ul Haq
University of Luxembourg
Luxembourg
fitash.ulhaq@uni.lu

Donghwan Shin
University of Luxembourg
Luxembourg
donghwan.shin@uni.lu

Lionel C. Briand
University of Luxembourg
Luxembourg
University of Ottawa
Ottawa, Canada
lionel.briand@uni.lu

ABSTRACT

With the recent advances of Deep Neural Networks (DNNs) in real-world applications, such as Automated Driving Systems (ADS) for self-driving cars, ensuring the reliability and safety of such DNN-enabled Systems emerges as a fundamental topic in software testing. One of the essential testing phases of such DNN-enabled systems is *online testing*, where the system under test is embedded into a specific and often simulated application environment (e.g., a driving environment) and tested in a closed-loop mode in interaction with the environment. However, despite the importance of online testing for detecting safety violations, automatically generating new and diverse test data that lead to safety violations presents the following challenges: (1) there can be many safety requirements to be considered at the same time, (2) running a high-fidelity simulator is often very computationally-intensive, and (3) the space of all possible test data that may trigger safety violations is too large to be exhaustively explored.

In this paper, we address the challenges by proposing a novel approach, called SAMOTA (Surrogate-Assisted Many-Objective Testing Approach), extending existing *many-objective* search algorithms for test suite generation to efficiently utilize *surrogate models* that mimic the simulator, but are much less expensive to run. Empirical evaluation results on Pylot, an advanced ADS composed of multiple DNNs, using CARLA, a high-fidelity driving simulator, show that SAMOTA is significantly more effective and efficient at detecting unknown safety requirement violations than state-of-the-art many-objective test suite generation algorithms and random search. In other words, SAMOTA appears to be a key enabler technology for online testing in practice.

CCS CONCEPTS

• **Software and its engineering** → **Empirical software validation**; **Search-based software engineering**.

KEYWORDS

DNN testing, online testing, many-objective search, surrogate-assisted optimization, self-driving cars

ACM Reference Format:
Fitash Ul Haq, Donghwan Shin, and Lionel C. Briand. 2022. Efficient Online Testing for DNN-Enabled Systems using Surrogate-Assisted and Many-Objective Optimization. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510188

1 INTRODUCTION

Given the recent achievements of Deep Neural Networks (DNNs) in many applications, including image classification [19] and object detection [44], they are increasingly used in autonomous systems, such as Automated Driving Systems (ADS) for self-driving cars. Therefore, ensuring the reliability of DNN-Enabled Systems (DES) in such contexts emerges as a fundamental topic in software testing.

As discussed by Haq et al. [14], DNN testing has two distinct phases: *offline testing* where DNNs are tested as individual units based on test datasets obtained without involving the DNNs under test, and *online testing* where DNNs are embedded into a specific application environment (e.g., a driving environment, often simulated given its safety-critical nature) and tested in a closed-loop mode in interaction with the application environment. In other words, online testing is a priori more realistic and considers that predictions generated by the DNNs when observing an environment state at time t impact the environment states after t. For example, only online testing can observe the accumulation of minor prediction errors over time, eventually causing a critical safety violation (e.g., colliding with a pedestrian).

However, despite the importance of online testing for detecting safety violations, automatically generating new and diverse test data that leads to safety violations entails several challenges. First, there can be many safety requirements, often independent from each other, to be considered at the same time. Second, running a high-fidelity simulator to check safety violations is typically computationally-intensive; the higher the fidelity of a simulator, the more time it takes to simulate, which directly impacts the cost of online testing. Third, the space of all possible test data that may trigger safety violations is too large to be exhaustively explored.

Contributions. To address the above challenges, we propose a novel approach, called SAMOTA (Surrogate-Assisted Many-Objective Testing Approach), that combines two distinct techniques: (1) *many-objective search* [2, 29] to effectively achieve many independent objectives (i.e., causing safety violations) within a limited time budget and (2) *surrogate-assisted optimization* [17] to efficiently search for critical test data using surrogate models that mimic the simulator, to a certain extent, but are computationally much less expensive. In particular, following state-of-the-art surrogate-assisted optimization algorithms [21, 39, 43], SAMOTA uses two search phases: *global search* (with global surrogate models) that *explores* the search space by capturing the global outline of the fitness landscape using global surrogate models, and *local search* (with local surrogate models) that *exploits* the local details around promising areas found by the global search. We also improve the local search performance using a novel, clustering-based approach that generates one local surrogate model for test data belonging to the same promising area.

Though SAMOTA can be applied to any DES that should be verified with online testing, we evaluated the efficiency and effectiveness of the approach in the context of ADS. Specifically, we use CARLA [7], a high-fidelity driving simulator, and Pylot [13], an advanced DNN-enabled ADS (DADS) composed of multiple DNNs capable of various tasks, such as traffic light detection, traffic sign detection, object tracking, and object classification. More than 300 computing hours of experimental results show that SAMOTA is significantly more effective and efficient at detecting unknown safety violations than state-of-the-art many-objective test suite generation algorithms and random search, within the same time budget.

The contributions of this paper are summarized as follows:

- SAMOTA, a novel approach to automatically and efficiently generate test data for online testing by carefully combining many-objective search and surrogate-assisted optimization;
- An extensive empirical evaluation of SAMOTA, in terms of efficiency and effectiveness, and its comparison with state-of-the-art alternatives and random search;
- a publicly available replication package of the evaluation, including the implementation of SAMOTA (see Section 6.6).

Significance. In many cyber-physical domains, it is clearly important to identify potential safety violations in a DES through online testing, involving a high-fidelity simulator in the loop, especially when there are complex interactions between the system and its environment. SAMOTA provides an efficient and effective online testing approach using surrogate models while considering many safety requirements at the same time. It is a practical enabler for the online testing of complex DES in realistic contexts. Our research is also an important step towards the scalable testing of DES, a very active research area in software testing.

Paper Structure. The rest of the paper is organized as follows. Section 2 provides background information on search-based testing and surrogate models. Section 3 formalizes the problem of test suite generation for the online testing of DES. Section 4 positions our work with respect to related work. Section 5 describes our approach. Section 6 evaluates our approach in the context of ADS. Section 7 concludes the paper.

2 BACKGROUND

2.1 Search-based Testing

Search-based software testing (SBST) [26] is arguably one of the most successful fields in software testing research. The key idea is to recast a testing problem as an optimization problem by properly defining fitness functions considering the objectives of the testing problem. For example, Panichella et al. [29] recasted the problem of test suite generation for branch coverage as a many-objective optimization problem, where the objectives are to cover individual branches and the fitness function for each objective estimates the likelihood of covering a branch. To efficiently achieve each objective individually as much as possible, Panichella et al. [29] introduced a many-objective search algorithm, named MOSA, that is tailored for test suite generation. Later, Abdessalem et al. [2] presented an alternative algorithm for many-objective test suite generation, named FITEST, by extending MOSA to dynamically reduce the size of populations during the search and thus improves its efficiency. Both MOSA and FITEST are carefully designed for the test suite generation problem by considering many objectives at the same time and are known to be effective in this application context [2, 29].

2.2 Surrogate Models

Evolutionary Algorithms (EAs) have been successfully applied to many complex engineering problems [9]. However, the fitness function evaluation of such complex problems often involves computationally expensive simulations or calculations [18]. To address this issue, many researchers have investigated *surrogate models* that can replace the computationally expensive function evaluations with much less expensive approximations. Among them, we briefly introduce the most widely used surrogate model types, i.e., Kriging [33], polynomial regression [34], radial basis function networks [5], and their ensemble [12].

2.2.1 Kriging. Kriging (also known as gaussian process regression) is one of the most widely used surrogate models since the 1970s. Similar to regression analysis, it predicts the value of a function as a combination of linear functions using a stochastic process. While it provides the error value for each prediction, it is relatively more time-consuming for training than other surrogate models.

2.2.2 Polynomial Regression. As a form of statistical regression analysis, polynomial regression models the relationship between the independent variable x and the dependent variable y in the form of nth degree polynomial in x. Though it is simple and intuitive, the existence of a few outliers can severely distort the approximation in nonlinear problems [28].

2.2.3 Radial Basis Function Network. A radial basis function network is an artificial neural network that consists of three layers: input, hidden, and output layers. Radial basis functions are used as activation functions, and the output of the network is a weighted sum of radial basis functions. It is known to provide both computational efficiency and reasonable training accuracy [21].

2.2.4 Ensemble. Though many surrogate models have been studied, there is no single surrogate model that consistently performs well for all problems [16]. To mitigate the issue, the ensemble of different surrogate models can be considered. Specifically, given an ensemble model $\mathbf{m}$ composed of member models $m_1, \ldots, m_k$, the final output of $\mathbf{m}$ for an input x, denoted with $\hat{y}_{\mathbf{m}}(x)$, is the weighted sum of all k member outputs as $\hat{y}_{\mathbf{m}}(x) = \sum_{i=1}^{k} w_i \times \hat{y}_{m_i}(x)$ where $\hat{y}_{m_i}(x)$ is the output of m_i for x and w_i is a weight for $\hat{y}_{m_i}(x)$. Following Goel et al. [12], w_i is defined as $w_i = \frac{(\sum_{j=1}^{k} e_j) - e_i}{(k-1)\sum_{i=1}^{k} e_i}$ where e_i is the training error of m_i.

Another advantage of using ensemble models is that we can easily compute the *uncertainty* of a prediction of an ensemble model.

Specifically, the uncertainty of $\hat{y}_{\mathbf{m}}(x)$, denoted with $\delta_{\mathbf{m}}(x)$, is defined as $\delta_{\mathbf{m}}(x) = max_{i,j}(|\hat{y}_{m_i}(x) - \hat{y}_{m_j}(x)|)$ for $i, j \in \{1, 2, \ldots, k\}$ and $i \neq j$. In other words, the uncertainty is calculated using the maximum difference between the outputs of the k member models.

3 PROBLEM DEFINITION

In this section, we provide a general but precise problem description regarding test suite generation for DES in the context of online testing. We use DADS (DNN-enabled ADS) as an example of DES, but the description can easily be generalized to all DES.

In online testing, the DADS under test are embedded into a driving environment, often in a loop with a simulator due to the high cost and risk associated with the real-world testing of vehicles. Using a simulator also enables the generation of various driving scenarios using the simulator's controllable attributes. In a simulator, the DADS receives sensor data (e.g., an image capturing a driving scene) generated by the simulator and produces control commands (e.g., steering angle, throttle, and brake) to drive the ego vehicle. Since the control commands actually drive the ego vehicle being simulated in online testing, predictions generated by the DNNs of the DADS at time t impact the sensor data to be generated after t. Therefore, it is essential to run the simulator for a specific driving scenario and check if a safety violation occurs. The goal of online test suite generation for DADS is to generate a minimal set of driving scenarios that cause the system under test to violate as many safety requirements as possible.

More specifically, let s be a driving scenario that defines the road topology, weather condition, and the trajectory of other mobile objects (e.g., other vehicles and pedestrians) in a virtual environment. The detailed definition of s (i.e., the definition of test input space) can vary depending on the configurable attributes of the simulator. By embedding a DADS d into a simulator and running it for s, at time t, the simulator generates d's sensor data i_t, such as an image capturing the driving scene (status) $\mathbf{v}_{s,t}$ taken by the front-facing camera mounted on the ego vehicle's dashboard. By taking i_t, d produces controls $d(i_t) = (st_t, ac_t, br_t)$ where st_t, ac_t, and br_t represent steering, acceleration, and braking commands, respectively. The simulator then updates $\mathbf{v}_{s,t+1}$ by taking into account $d(i_t)$. For each $\mathbf{v}_{s,t}$, we can verify if a safety requirement is violated or not. For example, regarding the safety requirement of lane-keeping (i.e., the ego vehicle should be in the center of the lane), we can measure the distance of the ego vehicle from the center of the lane in $\mathbf{v}_{s,t}$. If the distance is larger than a certain threshold, the safety requirement is violated. In general, let R be a set of safety requirements and $r(\mathbf{v}_{s,t})$ be the degree of the violation for a safety requirement $r \in R$ in $\mathbf{v}_{s,t}$. We say that d violates r in s if $r(\mathbf{v}_{s,t}) > \epsilon_r$ for any t during the simulation time of s, where ϵ_r is a threshold for r. Though that may not be possible, test suite generation attempts to generate a minimal set of test scenarios TS that satisfies $r(\mathbf{v}_{s,t}) > \epsilon_r$ for all $r \in R$ for some test scenarios $s \in TS$.

Test suite generation for DADS in online testing entails several challenges. First, the test input space is too large to be exhaustively explored because there are many attributes having many options that can be selected for generating a certain scenario. For example, road type has multiple options, such as straight, curved, and cross junction. Second, there are many safety requirements, usually independent of each other. For example, complying with traffic lights is independent from keeping the center of a lane. This further increases the complexity of the problem. Third, running a simulator for a scenario is often time-consuming due to the intensive computations required to dynamically update mobile objects and render driving scenes in virtual worlds, with high fidelity; for example, in our case study, one scenario execution in online testing takes 5-10 minutes. Fourth, depending upon the accuracy of the system under test, it may be infeasible to find a scenario causing violations for some safety requirements. Considering a limited time budget, if such infeasibility is observed at run time, it is essential to dynamically and efficiently distribute computation resources to the other safety requirements. Last but not least, DNNs in DADS are often developed by a third party, with expertise in ML, who does not provide access to internal information of the DNNs. Therefore, online testing should be conducted in a black-box manner without relying on internal information.

To address the challenges mentioned above, as detailed in Section 5, we suggest combining two distinct techniques: (1) many-objective search algorithms to effectively achieve many independent objectives (i.e., causing safety violations) within a limited time budget and (2) surrogate models that mimic the simulator, to the extent possible, while being computationally much less expensive. Furthermore, since it is black-box, this approach is DNN agnostic, e.g., it does not make any assumptions about the DNN architecture.

4 RELATED WORK

4.1 Online Testing for DNN-Enabled Systems

In recent years, online testing for DNN-Enabled Systems (DES) has attracted more attention, especially in the context of ADS.

Gambi et al. [11] presented AsFault, a tool for automatically generating road networks based on a genetic algorithm to test if the ego vehicle under test keeps the center of the lane while driving in a simulated environment. Majumdar et al. [23] presented Paracosm, a language and tool to systematically define and generate test scenarios for autonomous driving simulations. They used a fuzzing-based test input generation strategy to achieve high combinatorial coverage for the attribute values that define the test input space. Tuncali et al. [36] presented Sim-ATAV, a testing framework to generates test cases, using covering arrays and requirements falsification methods, for autonomous vehicle with machine learning components. Seymour et al. [31] presents an empirical study, in which they generated test cases, using metamorphic and equivalent partitioning techniques, to test DADS in black-box settings. Riccio and Tonella [30] presented DeepJanus, a search-based approach to generate similar input pairs, which causes the DADS under test, with a focus on lane keeping, to mispredict for one input and work fine for the other input. Despite this significant body of work, no existing study focuses on performing efficient online testing for DES, with many objectives (requirements), using surrogate models.

One notable exception is the study of Abdessalem et al. [1] as they used ML models (i.e., decision trees) to better guide the search process towards promising areas for efficient online testing for vision-based DADS. Nevertheless, their approach is inherently different from ours since their models are not surrogate models that can replace the computationally expensive fitness evaluations with much less expensive approximations but classification models that help focus the search on the critical test input space. Furthermore, they used the classification models outside the search algorithm (i.e., NSGA-II) to reduce the search space, whereas we use surrogate models inside the search algorithm to calculate fitness scores without using expensive simulators. In fact, their approach is based on learnable evolutionary search [27], whereas our approach is based on surrogate-assisted optimization [17]. Therefore, the two approaches are orthogonal, meaning that one can easily combine the two approaches together; for example, one can use SAMOTA instead of NSGA-II for the approach of Abdessalem et al. [1] and iteratively reduce the search space using decision trees while reducing the execution time of fitness evaluations using surrogate models in the main search algorithm.

4.2 Surrogate-Assisted Optimization

Addressing computationally expensive optimization problems using surrogate models (see section 2.2 for details) has been widely studied in the field of surrogate-assisted optimization [17].

Among many studies, Zhou et al. [45] presented a pioneering idea of combining global and local surrogate models to accelerate evolutionary optimization. Specifically, they used global surrogate models to filter some promising individuals, and utilized local surrogate models that represent the local fitness landscape in the vicinity of the individual to accelerate convergence. The experimental results on multimodal benchmark functions and a real-world aerodynamic shape design problem showed that the idea of combining global and local searches yields significant savings in computational cost when compared to alternatives.

Following up on the idea of combining global and local searches, Wang et al. [39] additionally used the prediction uncertainty of global surrogate models to select candidates for actual fitness evaluations. By providing the actual fitness scores of the most uncertain candidates, it maximizes the information gain of the surrogate models, making them more accurate faster. Recently, Liu et al. [21] also experimentally confirmed that using the most uncertain candidates in addition to the best predicted candidates was a promising strategy for solving benchmark problems with up to 30 dimensions.

However, no existing study addresses surrogate-assisted optimization, combining global and local surrogate models, applied to the problem of test suite generation for DES online testing.

5 SURROGATE-ASSISTED MANY-OBJECTIVE SEARCH FOR TEST SUITE GENERATION

This section provides a solution to the problem of test suite generation for DADS, described in Section 3, by combining many-objective search and surrogate models. In the following subsections, we first describe how many-objective search can be used for the problem of test suite generation for DADS. We then present our novel algorithm for surrogate-assisted many-objective search.

5.1 Test Suite Generation using Many-Objective Search

As described in Section 2.1, MOSA [29] and FITEST [2] have been introduced in the context of software testing to maximally cover individual test targets (e.g., branches). Therefore, we can apply the algorithms to our problem by carefully defining a set of corresponding test targets (objectives) and fitness functions.

Based on the problem definition in Section 3, we can define test objectives as violations of safety requirements. Specifically, for a set of safety requirements $R = \{r_1, \dots, r_n\}$ and a DADS d, the objectives to be achieved by a many-objective search algorithm are $r_i(\mathbf{v}_{s,t}) > \epsilon_{r_i}$ for all $i = 1, \dots, n$ where $r_i(\mathbf{v}_{s,t})$ is the degree of violation of d for a safety requirement r_i in scenario s at time t and ϵ_{r_i} is a violation threshold pre-defined by domain experts for each requirement r_i. In other words, the set of objectives is defined as $O = \{o_1, \dots, o_n\}$ where o_i is $r_i(\mathbf{v}_{s,t}) > \epsilon_{r_i}$ for $i = 1, \dots, n$.

Using the set of objectives, we can apply MOSA or FITEST whose pseudo-code is presented in Algorithm 1. It takes as input a set of objectives $O = \{o_1, \dots, o_n\}$, a population size p_s, and a set of thresholds $E = \{\epsilon_{r_1}, \dots, \epsilon_{r_n}\}$; it returns an archive (i.e., a test suite) A that aims to maximally achieve individual objectives in O.

Algorithm 1: Many-Objective Search for Test Suite Generation

```
Input  : Set of Objectives $O$
         Population Size $p_s$
         Set of Error Threshold $E$
Output : Archive $A$
1 Archive $A \leftarrow \emptyset$
2 Set of Uncovered Objectives $U \leftarrow O$
3 Set of Test Cases $P \leftarrow initialPopulation(p_s)$
4 while not (stopping-condition) do
5     Set of Test Cases $Q \leftarrow generateOffspring(P)$
6     Set of Test Cases $W \leftarrow calculateFitnessSim(P \cup Q)$
7     $A, U \leftarrow updateArchive(A, W, E, O)$
8     $P \leftarrow generateNextGen(W, U)$
9 return $A$
```

The algorithm begins with initializing A, a set of uncovered objectives U, and a set of test cases (i.e., test scenarios in our context) P of the size p_s (lines 1–3). Until the stopping criterion is met (e.g., a predefined computational time budget is exhausted), the algorithm repeats the following: (1) generating a new set of test cases Q using genetic operators from P using *crossover* and *mutation* (line 5), (2) generating another set of test cases W by merging P and Q and calculating their fitness scores by executing a simulator for every candidates in W (line 6), (3) updating A and U using W and E such that U excludes the objectives that are covered (achieved) by W for E and A includes the test cases from $A \cup W$, which are best at achieving the covered objectives (line 7), and (4) generating the next generation P from W considering U using *selection* (line 8). The algorithm ends by returning A (line 9).

The main differences between MOSA and FITEST are in the *initialPopulation* function (line 3) and the *generateNextGen* function (line 8): MOSA initializes P as a set of randomly generated test cases and keeps $|P| = p_s$ (typically p_s is set to $|O|$), whereas FITEST uses an adaptive random generation technique [6] to promote initial diversity and keeps reducing $|P|$ as $|U|$ decreases.

5.2 Surrogate-Assisted Many-Objective Search

Algorithm 1 appears to be suitable for solving the problem of test suite generation for DADS for online testing. However, iteratively evaluating the fitness scores of candidate test scenarios using simulations may take a prohibitive amount of time, preventing the generation of an effective test suite within a reasonable time budget. To address this, we present a novel algorithm, called Surrogate-Assisted Many-Objective Test suite generation Algorithm (SAMOTA), that extends Algorithm 1 to effectively utilize surrogate models.

Following state-of-the-art surrogate-assisted optimization algorithms [21, 35, 39], SAMOTA uses the idea of iterating two search phases, namely *global* search and *local* search. Briefly speaking, global search (with global surrogate models) first *explores* the search space by capturing the global outline of the fitness landscape, and then local search (with local surrogate models) *exploits* the promising areas found by the global search. Only the best predicted test cases that are found through cooperation between global and local search are evaluated through expensive simulations, whose results will be used to build more accurate surrogate models in the next iteration, thus iteratively finding more promising test cases. This search process continues until the computational budget is exhausted. Since it is usually difficult to accurately approximate the whole search space relying on global surrogate models only, such cooperation between global and local search is more effective at achieving the search objectives [45]. The details of global and local searches will be provided in § 5.2.1 and § 5.2.2, respectively.

In addition to applying the state-of-the-art, surrogate-assisted optimization algorithms to the important software engineering problem of test suite generation, we address its limitations in our context. We observed that these algorithms generate either too many local surrogate models that tend to degrade performance or only one local surrogate model that cannot accurately capture the local fitness landscape of individual promising areas. To address the issues, we introduce a novel, clustering-based approach that generates one local surrogate model per cluster composed of test cases that belong to the same promising area. Before we move on to the details, we first provide an overview of SAMOTA below.

Similar to Algorithm 1, SAMOTA (whose pseudo-code is shown in Algorithm 2) takes as input a set of objectives O, a population size p_s, and a set of error thresholds E, plus the maximum numbers of iterations for global search g_{max} and local search l_{max} (as stopping criteria for global and local search), the percentage of test cases to be used for training local surrogate models η, the minimum number of test cases in a cluster c_m for local search, and a database D that keeps all test cases already evaluated by the simulator (if any); SAMOTA then returns an archive A as in Algorithm 1 and the database D updated during the execution. The updated database can be used as input for future execution.

The algorithm begins with initializing A, the set of uncovered objectives U, and the set of test cases P of size p_s (lines 1-3). For the initialization of P while promoting diversity, an adaptive random generation technique [6] is used. The fitness scores of the test cases in P are then computed by executing a simulator (line 4) and updates A and U using P and E such that U excludes the objectives that are covered (achieved) by P with respect to the given error thresholds E and A includes the test cases from $A \cup P$, which are best at achieving the covered objectives (line 5). The algorithm also updates D to include P since the simulator is executed for all test cases in P (line 6). Until the stopping criterion is met, the algorithm repeats the surrogate-assisted global search (lines 8–11) and the

Algorithm 2: SAMOTA

```
Input :Set of Objectives $O$
       Population Size $p_s$
       Set of Error Thresholds $E$
       Max Iteration for Global Search $g_{max}$
       Max Iteration for Local Search $l_{max}$
       Percentage of Test Cases for Local Search $\eta$
       Minimum Number of Test Cases in Cluster $c_m$
       Database $D$
Output:Archive $A$
       Updated Database $D$
1  Archive $A \leftarrow \emptyset$
2  Set of Uncovered Objectives $U \leftarrow O$
3  Set of Test Cases $P \leftarrow \mathit{InitialPopulation}(p_s)$
4  $P \leftarrow \mathit{calculateFitnessSim}(P)$
5  $A, U \leftarrow \mathit{updateArchive}(A, P, E, O)$
6  $D \leftarrow \mathit{updateDatabase}(D, P)$
7  while not (stopping-condition) do
8      Set of Test Cases $\hat{T}_g \leftarrow \mathbf{GS}(D, U, p_s, g_{max}, E)$
9      Set of Test Cases $T_g \leftarrow \mathit{calculateFitnessSim}(\hat{T}_g)$
10     $A, U \leftarrow \mathit{updateArchive}(A, T_g, E, O)$
11     $D \leftarrow \mathit{updateDatabase}(D, T_g)$
12     Set of Test Cases $\hat{T}_l \leftarrow \mathbf{LS}(D, U, l_{max}, c_m, \eta)$
13     Set of Test Cases $T_l \leftarrow \mathit{calculateFitnessSim}(\hat{T}_l)$
14     $A, U \leftarrow \mathit{updateArchive}(A, T_l, E, O)$
15     $D \leftarrow \mathit{updateDatabase}(D, T_l)$
16 return $A, D$
```

surrogate-assisted local search (lines 12–15). For the global search, the algorithm generates the set of test cases $\hat{T}_g$ that are expected to satisfy U with respect to E using algorithm **GS** (line 8, detailed in § 5.2.1), calculates the fitness scores of the test cases in $\hat{T}_g$ by executing the simulator to generate the set of test cases T_g that also contains their actual fitness scores (line 9), and updates A, U, and D using T_g, as done for P (lines 10-11). For the local search, the algorithm repeats the same procedures as for the global search, except that it generates the set of test cases $\hat{T}_l$, that are expected to better satisfy U than $\hat{T}_g$, with respect to E using algorithm **LS** (line 12, detailed in § 5.2.2). The algorithm ends by returning A and D (line 16). Note that $|\hat{T}_g|$ and $|\hat{T}_l|$ decrease as $|U|$ decreases, resulting in further reducing the number of expensive executions of the simulator.

5.2.1 *GS (Global Search).*

This algorithm aims to explore the search space using global surrogate models for uncovered objectives. It basically uses the same search framework as Algorithm 1 and returns the *best* test case for each uncovered objective in terms of the fitness score *predicted* by the global surrogate model trained using all the test cases in the database. Recall that the resulting test cases will be evaluated using the simulator (line 9 in Algorithm 2) to calculate their actual fitness scores, and the database will be updated to include the test cases with their actual fitness scores, leading to more accurate surrogate models in the next iteration of **GS**. To further improve the accuracy, **GS** additionally finds and returns the *most uncertain* test case for each uncovered objective based on the uncertainty of the surrogate model predictions, which can be measured, for example, according to the magnitude of the disagreement among the outputs of the members of an ensemble surrogate model as explained in section 2.2.

Specifically, the **GS** algorithm (Algorithm 3) takes as input the database D, the set of uncovered objectives U, the population size p_s, the maximum number of iterations g_{max}, and the set of error thresholds E; it returns a set of resulting test cases $\hat{T}_g$ found by global surrogate models trained using D. $\hat{T}_g$ consists of the most promising test case for each uncovered objective $u \in U$ and the most uncertain test case for each u, leading to $|\hat{T}_g| \leq |U| \times 2$.

Algorithm 3: GS (Global Search)

```
Input  : Database $D$
         Set of Uncovered Objectives $U$
         Population Size $p_s$
         Max Iteration $g_{max}$
         Set of Error Thresholds $E$
Output : Set of Test Cases $\hat{T}_g$
1  Set of Global Surrogates $M_g \leftarrow trainGlobals(D, U)$
2  Set of Best Test Cases $\hat{T}_b \leftarrow \emptyset$
3  Set of Most Uncertain Test Cases $\hat{T}_n \leftarrow \emptyset$
4  Integer Counter $i \leftarrow 0$
5  Set of Test Cases $P \leftarrow initialPopulation(p_s)$
6  while $i < g_{max}$ do
7      Set of Test Cases $Q \leftarrow genOffspring(P)$
8      Set of Test Cases $W \leftarrow calcFitnessGS(P \cup Q, M_g)$
9      $\hat{T}_b, \hat{T}_n, U \leftarrow update(\hat{T}_b, \hat{T}_n, W, U, E)$
10     $P \leftarrow generateNextGen(W, U)$
11     $i \leftarrow i + 1$
12 return $\hat{T}_b \cup \hat{T}_n$
```

The algorithm begins with training the set of global surrogate models M_g (one per an uncovered objective in U) using all the test cases in D (line 1). The algorithm then initializes the set of best test cases $\hat{T}_b$, the set of most uncertain test cases $\hat{T}_n$, the counter i, and the set of test cases P of size p_s (lines 2–5). While $i < g_{max}$, the algorithm repeats the following steps: (1) generate the offspring Q from P (line 7), (2) generate the set of test cases W by merging P and Q and predicting their fitness scores using M_g while recording the uncertainty of individual predictions (line 8), (3) update $\hat{T}_b$, $\hat{T}_n$, and U such that $\hat{T}_b$ includes the best test case from $\hat{T}_b \cup W$ for each $u \in U$, $\hat{T}_n$ includes the most uncertain test case from $\hat{T}_n \cup W$ for each $u \in U$, and U excludes the objectives covered by $\hat{T}_b$ (line 9), (4) generate the next generation P from W for U (line 10), and (5) increase i by 1 (line 11). The algorithm ends by returning $\hat{T}_g = \hat{T}_b \cup \hat{T}_n$ (line 12).

Note that each global surrogate model should be able to provide the uncertainty of individual predictions in addition to predicted fitness scores. An ensemble model already satisfies the requirement as the disagreement among the outputs of the ensemble members can be used to measure uncertainty (see section 2.2 for more details). Following the widely used surrogate models in the area of surrogate-assisted optimization [21, 39], we combine Kriging, polynomial regression, and radial basis function network models into an ensemble surrogate model for global search to accurately provide fitness score predictions and easily calculate the uncertainty of individual predictions.

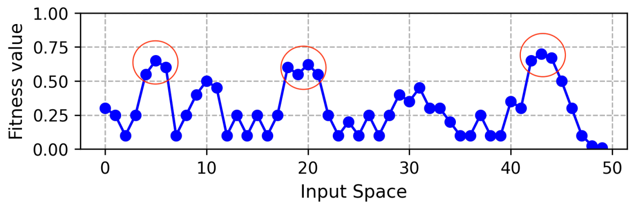

Figure 1: Illustration of clustering-based, local surrogate model generation

5.2.2 *LS (Local Search).*

This algorithm aims to exploit promising areas, found by the global search, using local surrogate models. For each promising area for each uncovered objective, it finds and returns the best predicted test case based on a single-objective search. Since inexpensive surrogate models are used for fitness evaluations, we can use population-based optimization algorithms, such as Genetic Algorithm [41], rather than single-state optimization algorithms to increase the search performance. An important challenge is how to train a local surrogate model that accurately captures the local fitness landscape of a certain area.

An approach proposed by Zhou et al. [45] builds a local surrogate model using the m nearest data points in the database for each of the top η% individuals (in terms of their actual fitness scores) in the database. While each surrogate model can intuitively represent the local fitness landscape in the vicinity of a good individual, the number of the surrogate models can be an issue (especially considering the growth of the database) because the local search should be iterated for each individual and uncovered objective to find the best test case. Another approach proposed by Wang et al. [39] builds one local surrogate model for all top η% individuals at once. While this is clearly better than the former solution in terms of number of surrogate models, the top η% individuals can be too widespread, making the surrogate model unable to accurately capture the *local* fitness landscape.

To address the limitations of existing approaches in our context, we introduce a clustering-based approach for local surrogate model generation, which combines the benefits of the two approaches described above by limiting the number of surrogate models while avoiding the combination of top individuals that are too far away from each other. Our approach once again first selects the top η% individuals. But it then clusters the selected individuals based on their vicinity in the fitness landscape. Based on the clustering results, our approach builds one local surrogate model for each cluster. Taking Figure 1 as a simple uni-dimensional example where the dots denote all test cases in the database and the red circles indicate the clusters generated for the top 10 test cases. While the individuals are widespread in the fitness landscape, a local surrogate model is built based on the test cases in each cluster, to allow the local search to exploit the best candidates located in a specific area.

For clustering test cases, we use Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [25], with an optional parameter specifying the minimum number of data points in each cluster. One can select the minimum number if the minimum amount of training data is already known for a certain surrogate model type to be used; otherwise, the default value of 5 provided by HDBSCAN can be used.

Algorithm 4: LS (Local Search)

```
Input  : Database $D$
         Set of Uncovered Objectives $U$
         Max Iteration $l_{max}$
         Percentage for Training Surrogate Models $\eta$
         Minimum Number of Test Cases in Cluster $c_m$
Output : Set of Test Cases $\hat{T}_l$
1  Set of Test Cases $\hat{T}_l \leftarrow \emptyset$
2  foreach Objective $u \in U$ do
3      Set of Clusters $C \leftarrow generateClusters(D, \eta, u, c_m)$
4      foreach Cluster (Set of Test Cases) $P \in C$ do
5          Surrogate Model $m_u \leftarrow trainLocal(P)$
6          Test Case $\hat{T}_b \leftarrow null$
7          Integer Counter $i \leftarrow 0$
8          while $i < l_{max}$ do
9              Set of Test Cases $Q \leftarrow genOffspring(P)$
10             Set of Test Cases $W \leftarrow calcFitnessLS(P \cup Q, m_u)$
11             $\hat{T}_b \leftarrow updateBestPredicted(\hat{T}_b, W)$
12             $P \leftarrow generateNextGen(W)$
13             $i \leftarrow i + 1$
14         $\hat{T}_l \leftarrow \hat{T}_l \cup \{\hat{T}_b\}$
15 return $\hat{T}_l$
```

Algorithm 4 presents the pseudo-code for local search, including our clustering-based local surrogate model generation. It takes as input the database D, the set of uncovered objectives U, the maximum number of iterations l_{max}, the percentage of test cases in D to be used for training local surrogate models η, and the minimum number of data points in a cluster c_m; it returns a set of test cases $\hat{T}_l$ that are expected to be the best at satisfying U according to local surrogate models. Note that $\hat{T}_l$ may contain multiple test cases for one objective if there are multiple promising areas for one objective.

Algorithm 4 begins with initializing a set of test cases $\hat{T}_l$ (line 1). For each uncovered objective $u \in U$, the algorithm finds the best test cases (possibly many if there are many promising areas) to be added into $\hat{T}_l$ (lines 2–14). Specifically, the algorithm generates clusters, with the minimum number of test cases c_m in each cluster, from the top η% test cases in D[1] for u (line 3), trains a local surrogate model for each cluster (lines 4–5), and finds the best predicted test case $\hat{T}_b$ using the local surrogate model (lines 6–13). The algorithm ends by returning $\hat{T}_l$ (line 15).

Unlike **GS**, **LS** does not use the uncertainty of surrogate models' predictions. Therefore, considering computationally efficiency, we can use any of the non-ensemble models described in section 2.2. We will show how to find the best configuration for **LS**, including the surrogate model type, in our empirical evaluation (see section 6.2).

6 EMPIRICAL EVALUATION

This section reports the empirical evaluation of our approach for efficient DNN testing when applied to an open-source DADS. Specifically, we investigate the following research questions:

RQ1: What is the best configuration for LS?

RQ2: How do alternative approaches fare in terms of test effectiveness?

RQ3: How do alternative approaches fare in terms of test efficiency?

RQ1 aims to find the best configuration for LS before we investigate the effectiveness and efficiency of SAMOTA. As explained in § 5.2.2, we propose a new clustering-based approach for better local surrogate model generation in LS. However, compared to existing approaches, we need to assess our clustering-based approach in terms of the effectiveness of LS. Furthermore, its impact may vary depending on the types of surrogate models (e.g., Kriging, polynomial regression, and radial basis function network). To answer these questions, we compare the combinations of surrogate model generation approaches and surrogate model types in terms of the ability of LS to find the most critical test inputs for a given time budget. Notice that we do not investigate the best configuration for GS since this has already been investigated in existing studies [20, 21] and we will therefore rely on reported results.

Using the best configuration for LS resulting from answering RQ1, RQ2 and RQ3 aim to investigate the effectiveness and efficiency of SAMOTA, respectively, in comparison to "naive" approaches that do not use surrogate models, such as MOSA, FITEST, and Random Search (with archive). To answer RQ2, we investigate how many safety violations are found by test suites generated using different approaches given a time budget. To answer RQ3, we investigate how quickly target safety violations are found by test suites generated using different approaches. The answers will show how effective and efficient SAMOTA can be by adapting the idea of surrogate-assisted optimization.

We conducted our evaluation on Ubuntu 18.04 running on Intel i9-9900K CPU with RTX 2080 Ti (11 GB) and 32 GB memory.

6.1 Case Study Subjects

We use Pylot [13], a publicly available DADS, as our case study subject. To enable simulation-based testing, we also use CARLA [7], a high-fidelity, open-source simulator for ADS.

Pylot is a DADS for developing and testing autonomous vehicle components (e.g., perception, prediction, planning) on the CARLA simulator and real-world vehicles [13]. Given a driving environment (either simulated or real), it drives the ego vehicle by controlling its acceleration, braking, and steering according to input data dynamically collected though sensors (e.g., camera and LiDAR). To achieve this, it consists of multiple components providing various functions of an autonomous vehicle, such as traffic light detection, lane detection, and object tracking. For each component, Pylot provides the implementations of state-of-the-art approaches based on pre-trained DNNs. For example, SSD (Single Shot Detector) [22] is used for object detection while SORT (Simple Online and Realtime Tracking) [4] and DeepSORT [42] enable obstacle tracking.

CARLA [7] is an open-source simulator based on the Unreal Engine [8], designed to support training, development, and validation of ADS. CARLA provides hand-crafted, high-fidelity virtual maps having various static environments, such as different road types (e.g., straights and curves), different sizes of buildings, different

[1] If the number of the top η% test cases in D is less than c_m, then the top c_m test cases in D are used.

shapes of trees, and different positions for traffic lights. Such configurable attributes can be used to define the test input space of DADS. In our evaluation, we consider as many configurable attributes as possible: road type, start/end-point on maps, the presence of other vehicles in front/same/opposite lane, other vehicle types, vehicle speed, the density of pedestrians, the presence of trees and buildings, time of day, and weather condition. To avoid invalid scenario generation (e.g., the road type is 'straight' while the start-point on a map is 'at the start of a curve road'), we use additional pre-defined constraints on the attribute values. More details are provided in the supporting materials [37].

Considering the capability of CARLA to compute related metrics (e.g., the distance between vehicles), we use the following six (safety) requirements for Pylot: (1) follow the center of the lane, (2) avoid collision with other vehicles, (3) avoid collision with pedestrians, (4) avoid collision with static objects (e.g., traffic signs), (5) abide by traffic rules (e.g., traffic lights), and (6) reach the destination within a given time. More details about the requirements and their implementations are provided in the supporting materials.

We selected the combination of Pylot and CARLA as our case study subject since (1) Pylot is an advanced, DNN-based ADS composed of multiple autonomous driving components, (2) Pylot is designed to be easily usable in simulated environments created by CARLA, and (3) both are publicly available. While Apollo[2] is another DADS one could rely on in our investigation, we do not use it because of compatibility issues (e.g., not compatible with the latest version of CARLA) and incomplete implementations (e.g., its camera perception module is not available).

6.2 RQ1: Best Configuration for Local Search

6.2.1 Setup. To answer RQ1, we generate a set of test cases by executing **LS** using different configurations and measure the *Local Search Effectiveness* (*LSE*) of the test set. Since **LS** aims to return a set of test cases that are expected to be the best at satisfying the given search objectives (i.e., violating safety requirements) by *predicting* the fitness scores of the test cases, the *LSE* of a test set can be measured by its *actual* fitness scores for all objectives. Specifically, we measure the *LSE* of a set of test cases T as $LSE(T) = \frac{\sum_{o \in O} \max_{t \in T} f(t,o)}{|O|}$ where O is a set of objectives and $f(t,o)$ is the (normalized) actual fitness score of a test case $t \in T$ for an objective $o \in O$. In other words, we calculate the maximum actual fitness score achieved when running all test cases in T for each objective in O and average these scores across all objectives.

Regarding the **LS** configurations, we consider the combinations of surrogate model types and generation approaches. Surrogate model types include KriGing (KG) [33], Polynomial Regression (PR) [34], and Radial basis Function network (RF) [5] as they are the most widely used in the literature [12, 21, 39]. For RF, we set the number of neurons in the hidden layer to 10 since it gave the best accuracy at predicting the fitness scores of test cases in our preliminary evaluation. Similarly, we set the degree of polynomial models for PR to 2 based on our preliminary evaluation. Regarding surrogate model generation, as already discussed in § 5.2.2, there are three different approaches for a set of data points (i.e., test cases with actual fitness scores): (1) build one surrogate model using *all* the data points; (2) build one surrogate model for each data point and its *neighbors*; and (3) build one surrogate model for each cluster after *clustering* the data points. The first and second approaches (i.e., *all* and *neighbors*) come from existing work [39, 45] while the third approach is newly proposed in this paper. However, we decided to exclude the second approach because it generates too many surrogate models to process as the number of data points provided to **LS** increases during the execution of SAMOTA. As a result, we consider a total six configurations (i.e., the combination of 3 surrogate model types and 2 surrogate model generation approaches). For simplicity, an **LS** configuration is denoted by X_Y where $X \in \{RF, PR, KG\}$ refers to the surrogate model type and $Y \in \{al, cl\}$ refers to the surrogate model generation approach (i.e., *all* and *clustering*). For example, KG_{cl} denotes the configuration that uses Kriging and the *clustering* approach for model generation.

[2]https://github.com/ApolloAuto/apollo

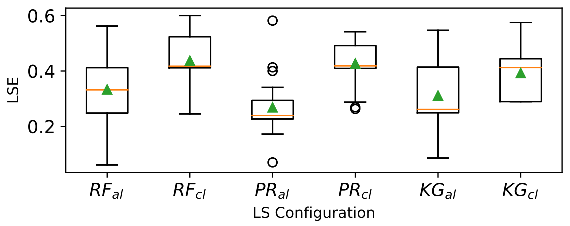

Figure 2: Distribution of *LSE* values for different LS configurations

To run **LS**, we need to set its inputs: the database D, the set of target objectives U, the maximum number of iterations l_{max}, the percentage of data points in D to be used for training local surrogate models η, and the minimum number of data points in a cluster c_m. To generate diverse test scenarios in D, we use 4-way combinatorial coverage for all the attributes used to define the test input space, resulting in $|D| = 587$. We select $l_{max} = 200$, based on our preliminary evaluations, since increasing l_{max} above 200 no longer significantly increases the predicted fitness scores of the test cases generated by **LS**. We select $\eta = 20\%$ following the recommendations of a recent paper [21]. For c_m, we use the default value of 5 provided by HDBSCAN [25].

To account for randomness in **LS**, we repeat the experiment 20 times. We apply the non-parametric Mann–Whitney U test [24] to assess the statistical significance of differences in *LSE* across **LS** configurations. Since we statistically test five hypotheses for each **LS** configuration (as we compare the six **LS** configurations pairwise), we use a level of significance $\alpha = 0.05/5 = 0.01$ by applying the Bonferroni correction [40] to reduce the risk of Type 1 errors. We also measure Vargha and Delaney's $\hat{A}_{AB}$ [38] to capture the effect size of the difference, which can be typically characterized as small, medium, and large when the $\hat{A}_{AB}$ value exceeds 0.56, 0.64, and 0.71, respectively. Note that $\hat{A}_{AB} = 1 - \hat{A}_{BA}$ and $\hat{A}_{AB} = \hat{A}_{BA} = 0.5$ means there is no statistical difference between the two compared configurations.

6.2.2 Results. Figure 2 shows the distribution of the *LSE* values for the six **LS** configurations. The orange bar and the green triangle in the middle of each box represent the median and the average,

Table 1: Statistical comparison results for LS configurations

A	B	p-value	$\hat{A}_{AB}$	A	B	p-value	$\hat{A}_{AB}$
RF_{cl}	RF_{al}	0.001	0.78	RF_{cl}	KG_{cl}	0.162	0.59
RF_{al}	PR_{al}	0.015	0.70	PR_{cl}	PR_{al}	0.000	0.90
PR_{cl}	RF_{al}	0.001	0.78	KG_{al}	PR_{al}	0.033	0.67
RF_{al}	KG_{al}	0.347	0.54	KG_{cl}	PR_{al}	0.000	0.86
KG_{cl}	RF_{al}	0.019	0.69	PR_{cl}	KG_{al}	0.001	0.79
RF_{cl}	PR_{al}	0.000	0.90	PR_{cl}	KG_{cl}	0.308	0.55
RF_{cl}	PR_{cl}	0.495	0.50	KG_{cl}	KG_{al}	0.003	0.76
RF_{cl}	KG_{al}	0.001	0.78	-	-	-	-

respectively. Table 1 additionally shows the results of statistical comparisons between different **LS** configurations. The columns *A* and *B* indicate the two configurations being compared. The columns *p-value* and $\hat{A}_{AB}$ indicate the statistical significance and effect size, respectively, when comparing *A* and *B* in terms of *LSE*.

Let us first compare the two surrogate model generation approaches, i.e., *al* and *cl*, for the same surrogate model type. In Figure 2, for all surrogate model types, the *cl* approach seems better than the *al* approach. In Table 1, given a level of significance $\alpha = 0.01$, we can see that the difference between *cl* and *al* for the same surrogate model type is statistically significant in all cases. Furthermore, the $\hat{A}_{AB}$ value is always greater than 0.71, meaning that *cl* is largely better than *al* in terms of *LSE*. In particular, the difference between PR_{cl} and PR_{al} is extreme (p-value = 0.000 and $\hat{A}_{AB} = 0.90$). This is because the *al* approach yields outliers by considering all data points at once, thus making the surrogate models (especially PR) inaccurate, whereas the *cl* approach does not thanks to clustering. As a result, for the same surrogate model type, using the *cl* approach is clearly better, showing the practical usefulness of our clustering approach in local surrogate model generation.

The ranking of the surrogate model types, based on their average *LSE* values over 20 runs, is RF_{cl}, PR_{cl}, and KG_{cl}, respectively. However, with $\alpha = 0.01$, the differences between them are all insignificant, meaning that it does not make a difference, in terms of *LSE*, whether RF_{cl}, PR_{cl}, or KG_{cl} is used. Nevertheless, the results do not imply that there is no significant difference among RF, PR, and KG for all problems. If possible, in practice, it is better for engineers to determine the best surrogate model type for **LS** before running SAMOTA. While it requires a dataset D containing diverse test cases with actual fitness scores, such a dataset could be available if the system under test already went through system testing and the test results regarding safety requirements were recorded. Otherwise, one can opt for RF_{cl} as it is known to provide both computational efficiency and reasonable training accuracy [21]. In our evaluation, we therefore use RF_{cl} for the remaining RQs.

To conclude, the answer to RQ1 is that our clustering-based approach (*cl*) for surrogate model generation is significantly better than the existing approach (*al*) in all cases but there is no practical difference overall between different surrogate model types. In practice, it is therefore better to experimentally determine the best local surrogate model type for a given system under test, while relying on *cl* for surrogate model generation. Otherwise, RF_{cl} can be the default option when this is not possible.

Table 2: Statistical comparison results for different search approaches

A	B	p-value	$\hat{A}_{AB}$	A	B	p-value	$\hat{A}_{AB}$
MO	FI	0.377	0.53	SI	MO	0.000	0.77
FI	RS	0.411	0.52	SE	MO	0.000	0.75
SI	FI	0.001	0.76	SI	RS	0.000	0.82
SE	FI	0.002	0.74	SE	RS	0.000	0.81
MO	RS	0.229	0.56	SI	SE	0.210	0.56

6.3 RQ2: Test Effectiveness

6.3.1 Setup. To answer RQ2, we generate a test suite using SAMOTA and its alternatives (e.g., MOSA and FITEST) for a fixed time budget and measure the *Test Effectiveness* (*TE*) of the test suite, defined as the proportion of safety requirements that are violated when running the test suite over the total number of safety requirements. *TE* ranges between 0 and 1, where higher values are desirable.

For SAMOTA, if we have a database that keeps test cases and their actual fitness scores computed in previous testing sessions, the database can be provided as input (see Algorithm 2). Though SAMOTA can start without it, providing a non-empty database as an initial input can boost the effectiveness of SAMOTA by improving the accuracy of surrogate models early. To better understand this, we use two configurations, i.e., SAMOTA starting with an empty database (SAMOTA-E) and SAMOTA starting with an initial database (SAMOTA-I). For the initial database, we use 4-way combinatorial coverage based on all the attributes used to define the test input space to generate diverse test cases, as we did for RQ1.

As alternatives to SAMOTA, we use MOSA [29] and FITEST [2], as they are the state-of-the-art many-objective test suite generation algorithms. We also use Random Search (RS) that randomly generates test cases for each iteration, as a baseline. Similar to MOSA and FITEST, we use an archive in RS so that it keeps the best at satisfying individual objectives in the archive until the search ends; the resulting archive is a test suite generated by RS. RS will provide insight into how easy the search problem is and will help us evaluate the impact of using advanced search algorithms, such as MOSA, FITEST, and SAMOTA, on test effectiveness.

The initial population size of MOSA, FITEST, and SAMOTA is the number of objectives. To be consistent in terms of population size, we set the number of newly generated test cases at each iteration of RS to be the number of objectives. For the other parameters, such as mutation and crossover rates in MOSA, FITEST, and SAMOTA-E/I, we adapt the default values used by Fraser and Arcuri [10].

To account for randomness in all approaches, we repeat the experiment 20 times. For each run, we use the same budget of two hours, as we found that it was long enough to converge in our preliminary evaluation. We apply the Mann–Whitney U test [24] to assess the statistical significance of differences in *TE* among approaches. Since we statistically test four hypotheses for each approach (as we compare RS, MOSA, FITESET, SAMOTA-E, and SAMOTA-I pairwise), we use a level of significance $\alpha = 0.05/4 = 0.0125$ by applying the Bonferroni correction [40] as we did for RQ1. We also measure Vargha and Delaney's $\hat{A}_{AB}$ [38] to capture the effect size of the difference.

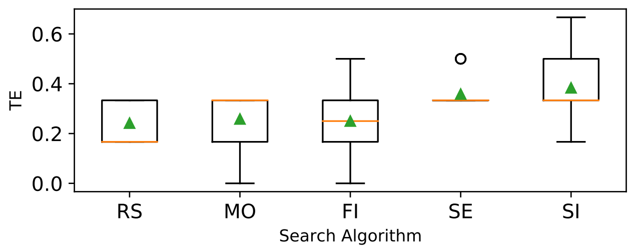

Figure 3: Distribution of *TE* values for different search approaches

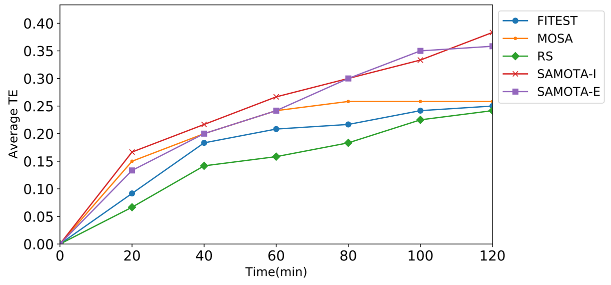

Figure 4: Test efficiency for different search approaches

6.3.2 Results. Figure 3 shows the distribution of *TE* values achieved by RS, MOSA (MO), FITEST (FI), SAMOTA-E (SE), and SAMOTA-I (SI) over 20 runs. Again, the orange bar and the green triangle in the middle of each box represent the median and average, respectively. Table 2, whose format is the same as Table 1, presents the results of statistical comparisons between different approaches. Notice that some safety requirements may never be violated, and therefore absolute *TE* values cannot be interpreted; we use these values only for comparison purposes.

Overall, the results show that SAMOTA-I is the best in terms of the average *TE* value for 20 runs. With a level of significance $\alpha = 0.0125$, the difference between SAMOTA-I and SAMOTA-E is insignificant (p-value = 0.210), but the differences between SAMOTA-I/E and the others are all significant with large effect sizes. This means that, by leveraging surrogate models, SAMOTA can be significantly more effective than the state-of-the-art test suite generation approaches and random search in terms of revealing unknown safety violations within a reasonable time budget.

Interestingly, the differences between MOSA and RS and between FITSET and RS are insignificant (p-values are 0.229 and 0.411, respectively), meaning that MOSA and FITEST are *not* significantly better than RS in terms of *TE*. A detailed analysis of the results shows that this is because the two-hour time budget is not enough for MOSA and FITEST to evaluate and evolve candidate test cases many times; on average, across 20 runs, only around five generations were completed during each run of MOSA and FITEST. In contrast, SAMOTA went through more than 800 generations using global and local surrogate models, within the same time budget, and therefore yielded significantly higher *TE* values than MOSA and FITEST as a result. This also confirms that the surrogate models of SAMOTA are sufficiently accurate to effectively guide the search towards test cases that cause safety violations. However, we would expect the difference between SAMOTA and MOSA or FITEST to diminish with a much longer time budget allowing them to go through many more generations. Nevertheless, such a scenario is unrealistic in practice as SAMOTA is likely to remain significantly more effective than its alternatives for practical time budgets.

Another interesting result is that there is no statistical difference between SAMOTA-I and SAMOTA-E, meaning that providing an initial database in SAMOTA does not lead to a significant improvement in detecting safety violations. This implies that SAMOTA can generate good enough test suites even without an initial database, an importance practical consideration. In RQ3, we will further compare SAMOTA-I and SAMOTA-E in terms of test efficiency (i.e., how fast safety violations are detected).

To conclude, the answer to RQ2 is that, in our context, SAMOTA is significantly more effective than other many-objective search algorithms tailored for test suite generation. Furthermore, SAMOTA can achieve acceptable test effectiveness without an initial database.

6.4 RQ3: Test Efficiency

6.4.1 Setup. To answer RQ3, we use the same approaches and setups as in RQ2 (i.e., SAMOTA-I/E, MOSA, FITEST, and RS). We generate a test suite using each approach and measure its *execution time* to achieve specific *TE* values (i.e., 1/6, 2/6, ..., 6/6 as there are six safety requirements in total).

To account for randomness, as we did in RQ2, we repeat the experiment 20 times. Notice that we cannot calculate the average execution time for 20 runs to achieve a specific *TE* value because not all 20 runs necessarily achieve such *TE* value (even for *TE* = 1/6). Therefore, we compute how the average *TE* values for 20 runs vary over time from 20*min* to 120*min*, in steps of 20*min*.

6.4.2 Results. Figure 4 shows the relationship between the execution time and the average *TE* values for 20 runs across all approaches. For example, RS is always at the bottom, meaning that, on average, RS achieves the lowest *TE* values compared to the others over the same time period.

Comparing SAMOTA-I and SAMOTA-E, we can see that SAMOTA-I achieves higher *TE* values than SAMOTA-E for the first 60*min* on average, but this difference vanishes after 80*min*. This is because the surrogate models of SAMOTA-E are relatively inaccurate in the beginning, as no initial database was provided, but they get more accurate over time as the database grows. Such growth also explains why MOSA achieves a higher average *TE* value than SAMOTA-E after 20*min* and why this trend is reversed after 80*min*.

Comparing SAMOTA-I/E and alternatives, SAMOTA-I is always at the top, meaning that it is always faster than the alternatives to achieve the same level of test effectiveness. This is the same for SAMOTA-E, except for the first 40*min* where it is slower than MOSA for the reason provided above.

To conclude, the answer to RQ3 is that SAMOTA is more efficient than alternative test suite generation approaches using many-objective search as soon as its surrogate models become sufficiently accurate. An initial database can boost the efficiency of SAMOTA in the initial search phase and allow it to surpass other techniques right from the start.

6.5 Threats to Validity

Since we use a specific DES (i.e., Pylot), coupled with a simulator (i.e., CARLA), external validity is our main challenge here. However, Pylot and CARLA are respectively representative of advanced DADS and high-fidelity driving simulators, in terms of accuracy, fidelity, and performance [7, 13]. Furthermore, no other realistic DADS, coupled with a high-fidelity simulator, is publicly available at this point, which is indeed an impediment to further experiments on this topic. Note that such experiments would likely be highly computationally-intensive, given that it took more than 300 computing hours in our case. Nevertheless, further experiments with different DES and high-fidelity simulators would be required to strengthen the generalizability of our results.

We want to remark that the applicability of SAMOTA is not limited to a specific DES since none of the algorithms (2-4) assumes specific DES properties. As long as there are many safety requirements (possibly independent from each other) for a DES under test and the fitness evaluation for each requirement is expensive, SAMOTA would show promising results as compared to the other algorithms that do not use surrogate models.

6.6 Data Availability

The replication package of our experiments — including the implementation of search algorithms, simulator, and the details of the experimental setup — is available on Figshare [37].

7 CONCLUSION

In this paper, we present SAMOTA, a novel approach to effectively and efficiently generate test data for DNN-enabled systems in the context of online testing. In essence, it provides a strategy to effectively combine surrogate-assisted optimization and many-objective search. Empirical evaluation results on an advanced DNN-enabled ADS, with a high-fidelity driving simulator, show that SAMOTA is significantly more effective and efficient, with a large effect size, than the state-of-the-art many-objective test suite generation algorithms and random search.

As part of future work, we plan to further investigate and explain the safety violations detected by our approach; for example, we can derive association rules between the test scenario attribute values and the resulting safety violations using association rule mining algorithms [15]. We also plan to increase the number of case studies by using high-fidelity simulators in the domain of autonomous drones, such as PEDRA [3] and AirSim [32], to increase the generalizability of our results.

ACKNOWLEDGMENTS

This work has received funding from Luxembourg's National Research Fund (FNR) under grant BRIDGES2020/IS/14711346/FUNTASY, the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277), IEE S.A. Luxembourg, and NSERC of Canada under the Discovery and CRC programs. Donghwan Shin was also partially supported by the Basic Science Research Programme through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2019R1A6A3A03033444).

REFERENCES

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2018. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 1016–1026.

[2] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Autonomous Cars for Feature Interaction Failures Using Many-Objective Search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/3238147.3238192

[3] Aqeel Anwar and Arijit Raychowdhury. 2020. Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning. *IEEE Access* 8 (2020), 26549–26560. https://doi.org/10.1109/ACCESS.2020.2971172

[4] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*. IEEE, 3464–3468.

[5] David S Broomhead and David Lowe. 1988. *Radial basis functions, multi-variable functional interpolation and adaptive networks.* Technical Report. Royal Signals and Radar Establishment Malvern (United Kingdom).

[6] Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, and T.H. Tse. 2010. Adaptive Random Testing: The ART of test case diversity. *Journal of Systems and Software* 83, 1 (2010), 60–66. https://doi.org/10.1016/j.jss.2009.02.022 SI: Top Scholars.

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 78)*, Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (Eds.). PMLR, 1–16. https://proceedings.mlr.press/v78/dosovitskiy17a.html

[8] Epic Games. 2019. Unreal Engine. https://www.unrealengine.com

[9] P.J Fleming and R.C Purshouse. 2002. Evolutionary algorithms in control systems engineering: a survey. *Control Engineering Practice* 10, 11 (2002), 1223–1241. https://doi.org/10.1016/S0967-0661(02)00081-3

[10] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Transactions on Software Engineering* 39, 2 (2013), 276–291. https://doi.org/10.1109/TSE.2012.14

[11] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically Testing Self-Driving Cars with Search-Based Procedural Content Generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) *(ISSTA 2019)*. Association for Computing Machinery, New York, NY, USA, 318–328. https://doi.org/10.1145/3293882.3330566

[12] Tushar Goel, Raphael T Haftka, Wei Shyy, and Nestor V Queipo. 2007. Ensemble of surrogates. *Structural and Multidisciplinary Optimization* 33, 3 (2007), 199–216. https://doi.org/10.1007/s00158-006-0051-9

[13] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Matthew A. Wright, Joseph E. Gonzalez, and Ion Stoica. 2021. Pylot: A Modular Platform for Exploring Latency-Accuracy Tradeoffs in Autonomous Vehicles. arXiv:2104.07830 [cs.RO]

[14] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. 2021. Can Offline Testing of Deep Neural Networks Replace Their Online Testing? *Empirical Software Engineering* 26, 90 (2021). https://doi.org/10.1007/s10664-021-09982-4

[15] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. 2000. Algorithms for Association Rule Mining — a General Survey and Comparison. *SIGKDD Explor. Newsl.* 2, 1 (June 2000), 58–64. https://doi.org/10.1145/360402.360421

[16] Ruichen Jin, Wei Chen, and Timothy W Simpson. 2001. Comparative studies of metamodelling techniques under multiple modelling criteria. *Structural and multidisciplinary optimization* 23, 1 (2001), 1–13. https://doi.org/10.1007/s00158-001-0160-4

[17] Yaochu Jin. 2011. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation* 1, 2 (2011), 61–70. https://doi.org/10.1016/j.swevo.2011.05.001

[18] Yaochu Jin and Bernhard Sendhoff. 2009. A systems approach to evolutionary multiobjective structural optimization and beyond. *IEEE Computational Intelligence Magazine* 4, 3 (2009), 62–76. https://doi.org/10.1109/MCI.2009.933094

[19] Shutao Li, Weiwei Song, Leyuan Fang, Yushi Chen, Pedram Ghamisi, and Jón Atli Benediktsson. 2019. Deep Learning for Hyperspectral Image Classification: An Overview. *IEEE Transactions on Geoscience and Remote Sensing* 57, 9 (2019), 6690–6709. https://doi.org/10.1109/TGRS.2019.2907932

[20] Dudy Lim, Yaochu Jin, Yew-Soon Ong, and Bernhard Sendhoff. 2010. Generalizing Surrogate-Assisted Evolutionary Computation. *IEEE Transactions on Evolutionary Computation* 14, 3 (2010), 329–355. https://doi.org/10.1109/TEVC.2009.2027359

[21] Qunfeng Liu, Xunfeng Wu, Qiuzhen Lin, Junkai Ji, and Ka-Chun Wong. 2021. A novel surrogate-assisted evolutionary algorithm with an uncertainty grouping based infill criterion. *Swarm and Evolutionary Computation* 60 (2021), 100787.

https://doi.org/10.1016/j.swevo.2020.100787

[22] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

[23] Rupak Majumdar, Aman Mathur, Marcus Pirron, Laura Stegner, and Damien Zufferey. 2019. Paracosm: A Language and Tool for Testing Autonomous Driving Systems. arXiv:1902.01084 [cs.SE]

[24] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[25] Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software* 2, 11 (2017), 205.

[26] Phil McMinn. 2011. Search-Based Software Testing: Past, Present and Future. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. 153–163. https://doi.org/10.1109/ICSTW.2011.100

[27] Ryszard S Michalski. 2000. Learnable evolution model: Evolutionary processes guided by machine learning. *Machine learning* 38, 1 (2000), 9–40. https://doi.org/10.1023/A:1007677805582

[28] Harvey J. Motulsky and Lennart A. Ransnas. 1987. Fitting curves to data using nonlinear regression: a practical and nonmathematical review. *The FASEB Journal* 1, 5 (1987), 365–374. https://doi.org/10.1096/fasebj.1.5.3315805 arXiv:https://faseb.onlinelibrary.wiley.com/doi/pdf/10.1096/fasebj.1.5.3315805

[29] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2015. Reformulating Branch Coverage as a Many-Objective Optimization Problem. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. 1–10. https://doi.org/10.1109/ICST.2015.7102604

[30] Vincenzo Riccio and Paolo Tonella. 2020. Model-Based Exploration of the Frontier of Behaviours for Deep Learning System Testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 876–888. https://doi.org/10.1145/3368089.3409730

[31] John Seymour, Dac-Thanh-Chuong Ho, and Quang-Hung Luu. 2021. An Empirical Testing of Autonomous Vehicle Simulator System for Urban Driving. *arXiv preprint arXiv:2108.07910* (2021).

[32] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2018. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*, Marco Hutter and Roland Siegwart (Eds.). Springer International Publishing, Cham, 621–635.

[33] Michael L Stein. 2012. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media.

[34] Stephen M Stigler. 1974. Gergonne's 1815 paper on the design and analysis of polynomial regression experiments. *Historia Mathematica* 1, 4 (1974), 431–439.

[35] Chaoli Sun, Yaochu Jin, Jianchao Zeng, and Yang Yu. 2015. A two-layer surrogate-assisted particle swarm optimization algorithm. *Soft computing* 19, 6 (2015), 1461–1475. https://doi.org/10.1007/s00500-014-1283-z

[36] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. 2018. Simulation-based Adversarial Test Generation for Autonomous Vehicles with Machine Learning Components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. 1555–1562. https://doi.org/10.1109/IVS.2018.8500421

[37] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Replication Package For "Efficient Online Testing for DNN-based Systems using Surrogate-Assisted and Many-Objective Optimization". https://doi.org/10.6084/m9.figshare.16468530

[38] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. https://doi.org/10.3102/10769986025002101 arXiv:https://doi.org/10.3102/10769986025002101

[39] H. Wang, Y. Jin, and J. Doherty. 2017. Committee-Based Active Learning for Surrogate-Assisted Particle Swarm Optimization of Expensive Problems. *IEEE Transactions on Cybernetics* 47, 9 (2017), 2664–2677. https://doi.org/10.1109/TCYB.2017.2710978

[40] Eric W Weisstein. 2004. Bonferroni correction. *https://mathworld. wolfram. com/* (2004).

[41] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85. https://doi.org/10.1007/BF00175354

[42] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*. 3645–3649. https://doi.org/10.1109/ICIP.2017.8296962

[43] C. Yang, J. Ding, Y. Jin, and T. Chai. 2020. Offline Data-Driven Multiobjective Optimization: Knowledge Transfer Between Surrogates and Generation of Final Solutions. *IEEE Transactions on Evolutionary Computation* 24, 3 (2020), 409–423. https://doi.org/10.1109/TEVC.2019.2925959

[44] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. 2019. Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems* 30, 11 (2019), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

[45] Zongzhao Zhou, Yew Soon Ong, Prasanth B Nair, Andy J Keane, and Kai Yew Lum. 2006. Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37, 1 (2006), 66–76. https://doi.org/10.1109/TSMCC.2005.855506