"Know What You Know": Predicting Behavior for Learning-Enabled Systems When Facing Uncertainty

Michael Austin Langford and Betty H.C. Cheng
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
{langfo37, chengb}@msu.edu

***Abstract*—Since deep learning systems do not generalize well when training data is incomplete and missing coverage of corner cases, it is difficult to ensure the robustness of safety-critical self-adaptive systems with deep learning components. Stakeholders require a reasonable level of confidence that a safety-critical system will behave as expected in all contexts. However, uncertainty in the behavior of safety-critical Learning-Enabled Systems (LESs) arises when run-time contexts deviate from training and validation data. To this end, this paper proposes an approach to develop a more robust safety-critical LES by predicting its learned behavior when exposed to uncertainty and thereby enabling mitigating countermeasures for predicted failures. By combining evolutionary computation with machine learning, an automated method is introduced to assess and predict the behavior of an LES when faced with previously unseen environmental conditions. By experimenting with Deep Neural Networks (DNNs) under a variety of adverse environmental changes, the proposed method is compared to a Monte Carlo (i.e., random sampling) method. Results indicate that when Monte Carlo sampling fails to capture uncommon system behavior, the proposed method is better at training behavior models with fewer training examples required.**

***Index Terms*—deep learning, evolutionary computation, software engineering, novelty search, uncertainty**

I. Introduction

As artificial intelligent software systems migrate from the purview of research and development into commercial applications with widespread use, trust in the ability for such systems to operate as intended and safely has become paramount. When deployed in the real world, autonomous systems will likely encounter adverse conditions not considered during their design, such as zero-day security attacks or system degradation from inclement weather. This problem is exacerbated when the system being tested is also self-adaptive or relies on a learning component. As system behavior adapts, new test cases may need to be considered and existing test cases may become obsolete [1] [2]. Furthermore, when a run-time[1] context diverges from training experience for a Learning-Enabled System (LES),[2] it can be difficult to establish confidence in the system's behavior. This paper proposes an automated approach to synergistically combine evolutionary computation with machine learning to model and predict system behavior for such uncertain contexts (i.e., "known unknowns"). This approach can be used by autonomous systems to predict and mitigate failure of a learning component at run time by deploying fail-safes for situations in which the component has not been adequately trained. By leveraging such context-aware self-assessment, use of the proposed techniques can help instill confidence that an autonomous system will recognize and use learning components in contexts for which it can handle.

It is difficult to determine the limitations of safety-critical systems that rely on Deep Neural Networks (DNNs) [4], especially for situations that deviate from training experience [5]. In real-world environments, adverse corner cases (e.g., the effect of a scratched camera lens or a lens flare caused by the setting sun) are frequently missing when establishing training and validation examples for an LES. Even for known adverse environmental factors included in data, confirmation and selection bias [6] can negatively influence learned behavior. Generative Adversarial Networks (GANs) [7] have been used to synthetically augment datasets to fill in gaps (e.g., image-to-image translation [8]), however the quality of such GANs is highly contingent on existing examples of the adverse phenomena. When examples of the adverse phenomena are lacking, other solution strategies are necessary to increase training/validation coverage for safety-critical LESs.

This paper introduces the Enlil[3] framework to produce a *behavior oracle* [10], which is used to predict how an LES will respond to operating conditions not covered by existing datasets. Enlil combines evolutionary computation with deep learning to support the automatic creation of a wide range of test cases for an LES, to assess how robust an LES is to specific forms of sensory noise, and to predict system behavior under uncertainty. Enlil synthesizes examples of new operating contexts with newly-introduced sensory phenomena by transforming existing data. Enlil then generates diverse synthetic examples that correspond to predefined behavior

[1]This paper hyphenates predicative compound adjectives (e.g., "run-time"). When used as nouns, such phrases will not be hyphenated (e.g., "run time").

[2]An LES is any system with functionality that is refined and optimized based on information gathered through experience (i.e., learning models) [3].

[3]Enlil is a Sumerian deity that possesses the "Tablet of Destinies [9]."

categories, which are then used to train a predictive behavior model via machine learning. The resulting behavior model can then be used to predict how the target LES will respond to sensory data with the newly-introduced phenomena. Machine learning plays two roles in this work. First, the LES uses machine learning to accomplish its system objectives (e.g., detect obstacles). Second, ENLIL uses machine learning to create a behavior oracle for predicting the behavior of an LES when facing unexpected conditions. Ultimately, this work aims to enable the construction of a more robust safety-critical LES through automated methods that assess and predict how the LES will behave under previously untested conditions. ENLIL's behavior oracles can enable a safety-critical system to preemptively determine when learning components are inadequate for a given run-time context, thereby prompting transitions/adaptations to mitigate faults (i.e., fail-safes).

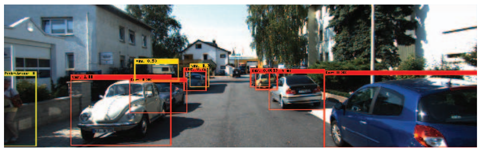
Experiments have been conducted to evaluate use of the ENLIL framework on an object detector for an autonomous vehicle under inclement environmental conditions (e.g., poor lighting, rain, and fog). Specifically, ENLIL is used to predict the object detector's performance in the presence of the newly-introduced environmental conditions. Fig. 1 provides an illustrative use case. Fig. 1(a) shows how a trained object detector can correctly identify all vehicles in a clear image (outlined in boxes). Fig. 1(b) shows that when a synthetic raindrop is introduced onto the camera lens, the object detector fails to detect the occluded vehicles. Finally, Fig. 1(c) shows how a behavior model trained by ENLIL can perceive the raindrop and predict that it will disrupt object detection (marked with red shading). This paper assumes that these environmental conditions have not been sufficiently covered by default training/test data, and therefore, it is unknown how they will impact the object detector's performance *a priori*. For validation, example training cases created by ENLIL have been compared to those generated by a baseline Monte Carlo random sampling method, and the accuracy of ENLIL's predictive models have been evaluated against models trained only on Monte Carlo training cases.

Results show that a significant percentage of system failures can be predicted preemptively with the ENLIL framework, which can enable developers to better implement mitigation strategies. Furthermore, results demonstrate that ENLIL generates balanced sets of training cases with respect to the types of system behavior induced. As such, ENLIL can train comparably accurate or better predictive models with fewer training examples for the considered object detector. The remainder of this paper is organized as follows. Sec. II overviews background information. Sec. III describes the ENLIL framework in detail. Sec. IV presents results from an empirical evaluation of ENLIL. Sec. V reviews related work. Finally, Sec. VI provides a conclusion with discussion of future work.

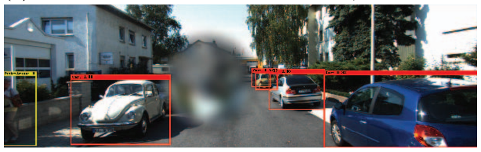
II. Background

This section provides a brief background on the topics of deep learning, challenges with software engineering for deep learning systems, and search-based software engineering.

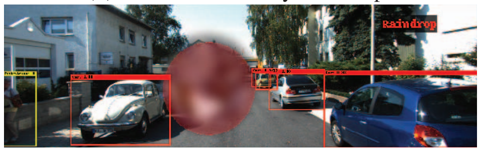
Demonstration of ENLIL Behavior Prediction



(a) Successful Detection in Unaltered, Clear View



(b) View Obscured by Raindrop



(c) ENLIL Detection of Raindrop & Predicted Failure

Fig. 1: Example of an ENLIL use case. An object detector is given a clear image and detects all vehicles (a). When a synthetic raindrop is introduced, occluded vehicles are not detected (b). The ENLIL behavior oracle can correctly identify the interfering raindrop (c) and predict possible failure.

A. *Deep Learning*

Deep learning, a branch of machine learning, solves tasks by learning a representation of the problem as a series of simpler expressions that map raw input data to appropriate corresponding solutions [4]. Typically, multilayered perceptrons (i.e., DNNs) are used as learning models for deep learning. DNNs comprise multiple hidden layers of linear transformations followed by non-linear activations, which transform input data and extract higher-level features that can determine the appropriate matching response (i.e., solution) for any given input. DNNs can be trained offline via supervised learning, where provided training examples include raw input data with matching solutions. A typical training process for a DNN uses gradient descent optimization to determine weights that minimize the error observed when processing training data. Deep learning has been shown to be successful for tasks that are difficult for more traditional algorithmic approaches, such as image recognition [11], object detection [12], image segmentation [13], and natural language translation [14].

B. *Deep Learning for Software Engineering*

Deep learning techniques are increasingly being used to address a range of software engineering problems [15]. Example development tasks include translating prototype design images into skeleton code for user interfaces [16], detecting functional similarity between two sources of code [17], and inferring function types from natural language descriptions [18]. For software security purposes, deep learning has been shown to detect vulnerabilities [19], automatically identify code authorship [20], and classify detected malware [21]. For software maintenance, deep learning has been proposed to localize bugs [22] and improve code readability [23]. Methods have also been proposed to use deep learning for software defect prediction to improve quality assurance for software [24] [25]. In this paper, we combine deep learning with evolutionary computation to develop a behavior oracle to predict how an LES will behave when exposed to uncertainty.

C. *Software Engineering Challenges for Deep Learning*

In contrast to traditional software programs, deep learning poses new challenges for verification and validation. The quality of a deep learning component is highly dependent on the quality of the training data available. To ensure that a DNN can correctly process any given input, a DNN must learn to *generalize* from its given training data [4]. However, the discovery of specially-crafted "adversarial examples" has shown that DNNs can be highly sensitive to tiny deviations from known training examples [26] [27] [28]. By adding small amounts of noise to input data (potentially imperceptible to humans), adversarial examples can force DNNs to make false negative predictions [29]. Adversarial examples have also been constructed entirely from noise to force DNNs to make false positive predictions [30]. Furthermore, DNNs can be susceptible to superficial details of an image's composition rather than the high-level semantics of the image's contents [31] [32]. Therefore, in addition to showing that a DNN is statistically accurate for an independent set of validation data, it is also important to show that a DNN is *robust* to small deviations from known training/validation data [33]. When only partial information is known about the operating environment of a DNN, it is challenging to determine how well the distribution of training/validation data will correspond to run-time scenarios. If both training and validation data fail to properly represent adverse run-time phenomena at run time, it is uncertain how the DNN will respond.

D. *Search-Based Software Engineering*

Search-Based Software Engineering (SBSE) techniques attempt to solve software engineering problems through the use of search-based optimization algorithms [34]. The search space is defined by specific aspects of a software system, where an optimal solution for the software engineering problem is determined by *metaheuristics* of the software system [34]. For automated software testing, evolutionary computation [35] has been used to automatically generate test cases [36]. These techniques define a space of candidate test cases for the software system and use metaheuristics of the software to find one or more "ideal" test cases. When defined in the framework of an Evolutionary Algorithm (EA), a *fitness* objective is derived from the chosen metaheuristics. The EA then evolves a *population* of test cases through *selection*, *crossover*, and *mutation* operations to uncover new candidate test cases that better fit the objective. EvoSuite [37] and SAPIENZ [38] both implement fitness-driven EAs for automated testing. These fitness-driven EAs have been found to be effective for problem spaces with generally *convex* fitness landscapes, but as greedy methods, they can suffer from providing sub-optimal solutions.

One variation to the traditional EA for search is *novelty search* [39] [40]. With novelty search, populations are not driven to optimize any explicit fitness objective. Instead, populations are evolved to pressure individual candidates to exhibit increasingly unique characteristics with respect to the rest of the population. By encouraging *diversity* in the population, the search no longer suffers from an attraction to a single local optimum in the fitness landscape. A wider coverage of the search space provides more opportunities to uncover latent or unexpected results. Novelty search has been used to test object-oriented systems and monitor the utility of self-adaptive systems [41] [42]. In prior work, novelty search has been used to train more robust versions of existing learning systems with ENKI [43] [44]. In this work, novelty search enables a self-assessing behavior oracle to determine whether systems are sufficiently robust at run time.

III. METHODOLOGY

This section describes the ENLIL framework, an automated method to create a more robust LES through predictive behavior modeling. First, ENLIL automatically assesses LES behavior under synthetic environmental phenomena to generate a collection of operating contexts that lead to specific *behavior categories* (i.e., types of resulting behavior). Next, ENLIL uses the generated contexts to train a predictive behavior model of the LES (i.e., a behavior oracle [10]) that can be used either internally or by an external system (e.g., an adaptation manager [45]) to determine how the LES will behave when faced with any given context of the environmental phenomena.

Fig. 2 provides a data flow diagram (DFD) for the ENLIL framework, where circles represent computational steps, boxes represent external entities, arrows represent data flow, and data stores are represented within parallel lines. ENLIL has two primary objectives: to generate a diverse variety of operating contexts for the LES (***Steps 1 and 2***) and to infer a behavior model of the LES based on the generated contexts (***Step 3***).

For example, Fig. 3 demonstrates how rainfall can impact an object detector for an autonomous vehicle to the point of failure. The object detector can successfully identify an automobile and a cyclist in the absence of rainfall (Fig. 3(a)). Light synthetic rainfall is introduced with no impact on detection (Fig. 3(b)). However, more intense rainfall can result in a failure to detect the cyclist (Fig. 3(c)). When default training data does not include any such examples of rainfall, the ENLIL framework generates synthetic examples and trains a behavior model to predict the object detector's performance under rainfall (i.e., if it is likely to miss an object).

Step 1: *Context Generation*

To assess the behavior of an LES under a previously unseen environmental condition, data must be collected to determine how the introduced condition impacts the behavior of the LES. To accomplish this task, the ENLIL framework requires an *operating specification*, *behavior specification*, *behavior categories*, and *evaluation procedure* from the user.

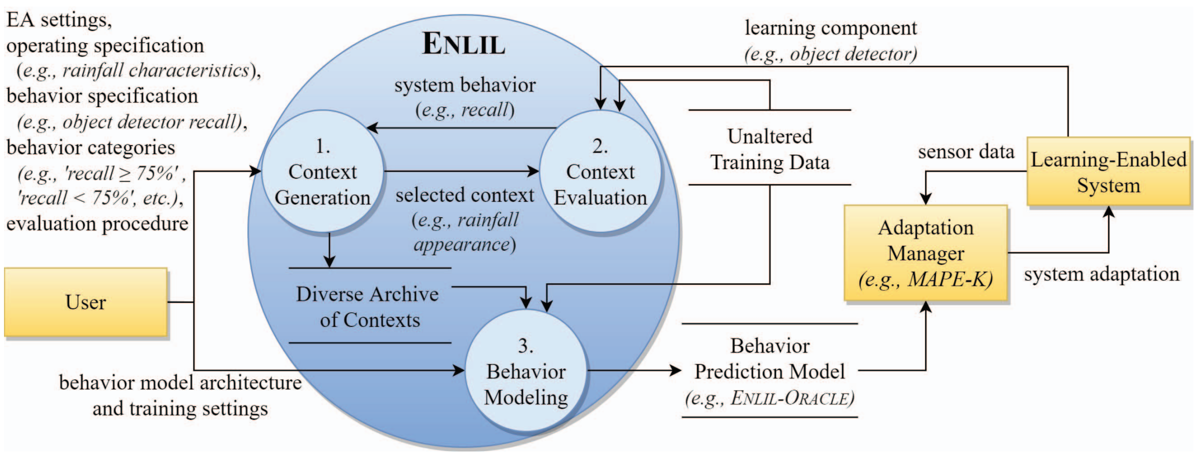
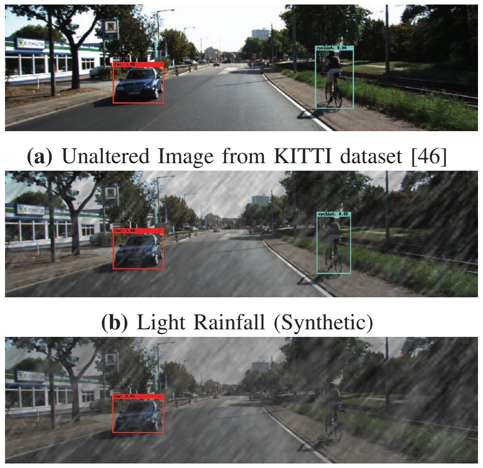


Fig. 2: High-level DFD of the ENLIL framework. Computational steps are shown as circles. Boxes represent external entities. Directed lines indicate data flow. Persistent data stores are marked within parallel lines.

The Impact of Rainfall on an Object Detector



(a) Unaltered Image from KITTI dataset [46]

(b) Light Rainfall (Synthetic)

(c) Heavy Rainfall (Synthetic)

Fig. 3: Illustrative example for how rainfall can affect an object detector. Detected objects are marked with bounding boxes in (a). Synthetic rainfall is introduced in (b) and (c). As rainfall intensity increases, the effectiveness of the object detector to detect the cyclist decreases to the point of failure (c).

a) Operating specification: The operating specification defines each configurable parameter of the LES's operating environment (e.g., rainfall angle, intensity, etc.). A label and permissible value ranges must be specified for each parameter, where values can be numeric or categorical. Each candidate operating context generated by ENLIL is sampled from the values defined in this specification. For the example rainfall phenomena (Fig. 3), an operating specification (OP) could be defined as follows:

$$\text{OP} = \begin{bmatrix} rainfall\_angle & \in & [\frac{1}{4}\pi, \frac{3}{4}\pi] \\ rainfall\_density & \in & [0,1] \\ rainfall\_intensity & \in & [0,1] \\ rainfall\_seed & \in & [0,100] \end{bmatrix}$$

b) Behavior specification: The behavior specification defines a set of observable behavior metrics for the LES. For an object detector, this may include properties such as *precision*[4] and *recall*[5] for a set of validation images, specified as follows.

$$\text{BV} = \begin{bmatrix} precision & \in & [0,1] \\ recall & \in & [0,1] \end{bmatrix}$$

In addition to these component-level properties, system-level properties may also be considered when *utility functions* [48] exist to evaluate functional objectives at run time.

c) Behavior categories: The ENLIL framework assumes that a set of behavior categories has been established by the user, which are defined in relation to the behavior metrics in the behavior specification. For example, behavior categories could be specified as "expected object misdetection" (CAT 1), "expected pedestrian misdetection" (CAT 2), and "expected successful detection" (CAT 3), where each category is based on thresholds (θ) for the recall (r) according to the following user-specified mapping.

$$\text{BV-CAT}(r) = \begin{cases} \text{CAT } 1, & \text{if } r_{total} < \theta_{total} \\ \text{CAT } 2, & \text{if } r_{total} \geq \theta_{total} \land r_{ped} < \theta_{ped} \\ \text{CAT } 3, & \text{if } r_{total} \geq \theta_{total} \land r_{ped} \geq \theta_{ped} \end{cases}$$

d) Evaluation procedure: The evaluation procedure specifies how ENLIL must instantiate the simulation environment and monitor the LES's behavior for each given operating context (see ***Step 2***).

To efficiently search the space of all possible operating contexts and uncover a wide range of examples for each specified behavior category, the ENLIL framework uses an EA search procedure. The EA generates candidate operating contexts, assesses the behavior of an LES under each candidate context, and archives the most diverse contexts found to

[4]*Precision* is the ratio of correctly detected objects to all detected objects (i.e., the ratio of *true positives* to both *true positives* and *false positives*) [47].

[5]*Recall* is the ratio of correctly detected objects to all detectable objects (i.e., the ratio of *true positives* to both *true positives* and *false negatives*) [47].

match the given behavior category. The result is an archive of operating contexts with widely varying appearance but similar resulting LES behavior to match each respective behavior category. The intent is to apply this technique for situations where the full impact of an operating condition is not known for the LES, and it is not possible to exhaustively assess every explicit context. When considering how rainfall might impact a camera-based object detector for an autonomous vehicle, it is not practical to evaluate every potential appearance of rainfall. Given a parameterized simulation procedure to emulate rainfall, ENLIL evolves a population of multiple rainfall contexts in parallel and encourages each generation to exhibit increasingly different appearances of rainfall (e.g., *angle* and *intensity*) that result in the same behavior category for the LES (e.g., "expected pedestrian misdetection").

ENLIL's EA for context generation is described in Algorithm 1. Each candidate context is represented by a *genome* and *phenome*. Genomes encode the operating characteristics of each context (e.g., rainfall angle, intensity, etc.), where the number of specified operating parameters will determine the length of each genome. Phenomes encode both the operating characteristics and resulting system behavior for each context (e.g., angle and intensity values for rainfall and resulting object detector accuracy). An initial *population* of operating contexts is generated with random operating characteristics. Operating characteristics for each context in the population are manipulated through the recombination and mutation of selected genomes. After modifying genomes, a population's phenomes are determined by evaluating the behavior of the LES for each individual context. To encourage diversity, the population is compared to an archive that tracks phenomes that are mutually unique and fit the given behavior category.

ENLIL ranks each context in the current population based on how its phenomes compare to the archived contexts. Context diversity is determined by the Euclidean distance between an individual's phenome and the phenomes of its nearest neighbors as follows,

$$\mathrm{nov}(\mathbf{p}, \mathbf{P}, k) = \mathrm{mean}\Big(\text{min-k}\big(\|\mathbf{p} - \mathbf{p}_i\|^2 \ \forall \mathbf{p}_i \in \mathbf{P} : \mathbf{p}_i \neq \mathbf{p}\big)\Big)$$

where the *novelty score* (nov) is computed as the average distance between a phenome ($\mathbf{p}$) with its k nearest neighbors in the set of all archived phenomes ($\mathbf{P}$). Contexts with phenomes that are more distant from the closest matching phenomes in the archive (i.e., less similar) are given priority. Since ENLIL aims to generate contexts with diverse appearance but similar resulting system behavior, an additional step is included to filter out any candidate contexts from the population that do not match the specified behavior category (Line 9 in Algorithm 1). After ranking each context in a population, any context that results in behavior other than the target behavior category is excluded from the population, such that it will not contribute to future generations. Thus, as ENLIL evolves the population over consecutive generations, individual contexts within the population will be favored for survival and reproduction when they exhibit operating characteristics that are different from those found in the archive yet produce similar behavior in the LES (e.g., "expected pedestrian misdetection"). In essence, ENLIL attempts to generate the broadest set of example contexts that yield the same resulting system behavior for a given behavior category.

Algorithm 1 ENLIL: Context Generation (***Step 1***)

```
1:  function EVOLUTIONARY-SEARCH($n\_generations$, $behavior\_pattern$, $eval\_func$)
2:    $archive \leftarrow \emptyset$
3:    $pop \leftarrow$ RANDOM-POPULATION()
4:    for 0 to $n\_generations$ do
5:      $pop \leftarrow$ SELECTION($pop$)
6:      $pop \leftarrow$ RECOMBINATION($pop$)
7:      $pop \leftarrow$ MUTATION($pop$)
8:      $pop \leftarrow$ EVALUATION($pop, eval\_func$)
9:      $pop \leftarrow$ FILTER($pop, behavior\_pattern$)
10:     $archive, pop \leftarrow$ COMMIT-TO-ARCHIVE($archive, pop$)
11:   return $archive$
12: function COMMIT-TO-ARCHIVE($archive$, $pop$)
13:   $scores \leftarrow$ RANK-INDIVIDUALS($archive, pop$)
14:   $pop \leftarrow pop : scores > novelty\_threshold$
15:   $archive \leftarrow$ TRUNCATE($archive \cup pop$)
16:   return $archive, pop$
```

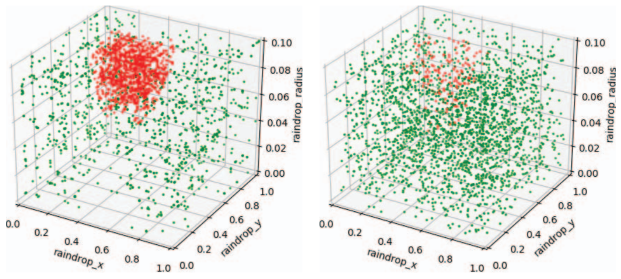
Step 2: *Context Evaluation*

For each given context, a subset of the default training data is transformed to simulate the environment described by the context. Then the transformed data is processed by the DNN to determine the resulting *confusion matrix*,[6] which can be used to derive the DNN's precision and recall behavior metrics.

The scatter plot in Fig. 4 shows how ENLIL's archived contexts compare to Monte Carlo (i.e., random-generated) contexts. For this example, ENLIL introduced a variety of raindrops onto the lens of a camera for an LES. Operating characteristics were defined by parameters such as the raindrop's image position and radius. Behavior categories were defined to include a category for raindrops that have no significant impact on the LES's ability to detect objects (green) and a category for raindrops that cause the LES to consistently fail to detect objects (red). ENLIL was able to generate an equal number of contexts for each behavior category, where Monte Carlo generation produced a severely imbalanced set of contexts, favoring raindrops that had no significant impact on the LES. Since ENLIL uses these generated contexts as training cases for predictive behavior models, a balanced distribution is preferred to reduce bias towards a specific behavior category. To reach a comparable number of failures cases, the Monte Carlo method would need to generate significantly more contexts (e.g., over 10X, for the examples shown in Fig. 4). This form of *between-class imbalance* is a known problem for learning algorithms [49]. ENLIL addresses this problem using novelty search and parallel evolution to produce archives of equal sizes for each behavior category.

[6]A *confusion matrix* for an object detector tallies the number of correct and incorrect detections for each type of object [47].

Comparison of ENLIL vs. Monte Carlo Generated Contexts



(a) ENLIL Contexts **(b)** Monte Carlo Contexts

Fig. 4: Scatter plot comparison of ENLIL-generated (a) versus Monte Carlo-generated (b) raindrop contexts (where a raindrop occludes the view). Points correspond to raindrops with varying size/position. Colors show whether a raindrop resulted in failure for the LES (red) or had no significant impact (green). ENLIL contexts show distinct clusters of equal quantity. Monte Carlo contexts contain few problem-causing raindrops.

Step 3: *Behavior Modeling*

To predict an LES's behavior under a condition of uncertainty (e.g., the presence of rainfall), the ENLIL framework constructs and trains a separate meta-level DNN from synthetically-generated data to act as a *behavior oracle* for the LES. Data for training the behavior oracle is derived from a combination of unaltered training data for the LES and the contexts archived by ENLIL in ***Step 1***. Each entry in the ENLIL's training dataset consists of the following values: synthetic LES sensor data (x), the context parameter values (t_0) used to generate the synthetic data, and a label (t_1) for the observed behavior category for the LES when exposed to the synthetic data. ENLIL trains the behavior oracle DNN (ENLIL-ORACLE) by learning to model the relationship between x and the target variables t_0 and t_1. Once trained, the resulting ENLIL-ORACLE DNN may be used as a function to preemptively determine how the LES will react to new examples of sensor data and thus enable adaptation of the LES at run time to mitigate failure.

A high-level topology for the ENLIL-ORACLE DNN is illustrated in Fig. 5. ENLIL-ORACLE takes sensor input data for the LES (x) and outputs an estimation of the corresponding operating context parameters (y_0) and a prediction for the LES's behavior category (y_1). ENLIL-ORACLE contains four sub-networks: an *interference isolation* sub-network, a *feature extraction* sub-network, an operating *context regression* sub-network, and a *behavior classification* sub-network.

The interference isolation and feature extraction sub-networks are responsible for reducing sensor input (x) into a set of learned features to enable behavior classification. The interference isolation sub-network learns to isolate any adverse noise introduced into the sensor input. The feature extraction sub-network learns to reduce the isolated noise into a set of latent features. The specific architectures for the interference isolation and feature extraction sub-networks are dependent on the type of sensor data used by the LES. For this study, sensor data is limited to static images, and the feature extraction sub-network has been implemented with a ResNet [11] architecture (a common method for image processing tasks).

The extracted features are provided as input to the context regression sub-network, which is responsible for estimating an operating context (y_0) to describe the given sensor input (i.e., a *perceived context*). For this study, the context regression sub-network comprises a fully-connected (FC) [4] layer with Rectified Linear Unit (ReLU) [50] activation and a second FC layer of n units with linear activation (where n is the number of context parameters).

Output from the context regression sub-network is relayed to the behavior classification sub-network to classify an expected behavior category (y_1) for the perceived context (i.e., an *inferred behavior*). The behavior classification sub-network for this study comprises an FC layer with ReLU activation and a second FC layer of m units with softmax[7] activation (where m is the number of behavior categories). Thus, ENLIL-ORACLE receives sensor data, perceives an operating context for the given data, and infers a behavior category for the LES in response to the context.

A *gradient descent* training method (Algorithm 2) minimizes the error of the ENLIL-ORACLE DNN by adjusting its weight values via *back propagation* [4]. The training procedure is implemented in two stages. First, the behavior classification sub-network is trained in isolation to minimize the *categorical cross entropy* between t_1 and y_1 for synthetic examples of each of the archived contexts. Second, the behavior classification sub-network is frozen, and the whole network is trained to minimize the *mean-squared-error* of t_0 and y_0 for synthetic examples of each of the archived contexts. The two-stage approach prevents the output of the behavior classification sub-network from directly influencing the types of features extracted by the feature extraction sub-network. Instead, features are extracted to have a greater relevance to the network's perceived context.

Algorithm 2 ENLIL: Behavior Modeling (***Step 3***)

```
1:  function TRAIN-BEHAVIOR-MODEL($archive$, $train\_data$, $n\_epochs$)
2:    $dnn \leftarrow$ INITIALIZE-DNN()
3:    $subnet \leftarrow$ GET-BEHAVIOR-SUBNET($dnn$)
4:    for 0 to $n\_epochs$ do
5:      for ($context, behavior$) in $archive$ do
6:        $x \leftarrow$ SELECT-RANDOM($train\_data$)
7:        $x \leftarrow$ TRANSFORM($context, x$)
8:        $t_0, t_1 \leftarrow context, behavior$
9:        $subnet \leftarrow$ FIT-WEIGHTS($subnet, x, (t_0, t_1)$)
10:   $dnn \leftarrow$ FREEZE-BEHAVIOR-SUBNET($dnn, subnet$)
11:   for 0 to $n\_epochs$ do
12:     for ($context, behavior$) in $archive$ do
13:       $x \leftarrow$ SELECT-RANDOM($train\_data$)
14:       $x \leftarrow$ TRANSFORM($context, x$)
15:       $t_0, t_1 \leftarrow context, behavior$
16:       $dnn \leftarrow$ FIT-WEIGHTS($dnn, x, (t_0, t_1)$)
17:   return $dnn$
```

[7]The *softmax* function estimates a classification probability distribution [4].

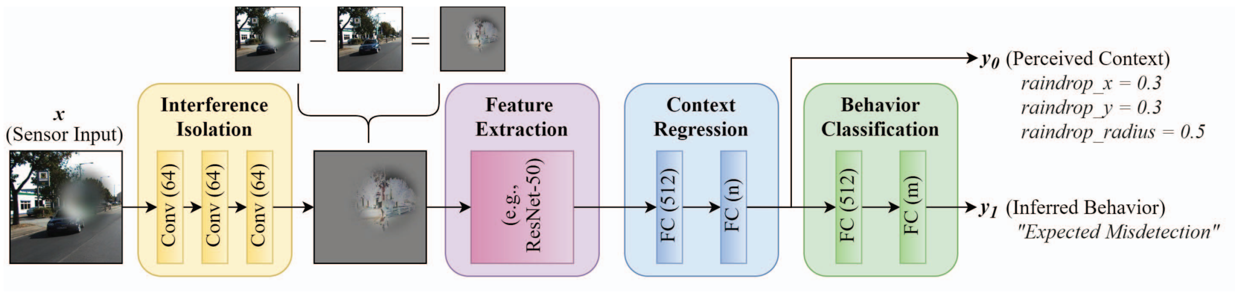


Fig. 5: High-level illustration of the ENLIL-ORACLE DNN architecture. The architecture comprises four sub-networks. First, the suspected adverse noise is separated from the sensor input by the *interference isolation* sub-network. Second, a set of "latent" features are distilled from the isolated noise via a *feature extraction* sub-network. Next, a *context regression* sub-network converts these latent features into a "perceived" context for the current environment. Finally, a *behavior classification* sub-network predicts a behavior category for the LES.

IV. Evaluation

This section presents results from an empirical evaluation of the ENLIL framework. Experiments have been performed on an LES that uses an image-processing DNN for object detection. Multiple trials have been conducted to evaluate the impact of specific weather conditions on the LES. Synthetic examples of each weather condition are generated using both ENLIL and a Monte Carlo method. Next, predictive behavior oracles are trained using examples generated by each method for comparison. We answer the following research questions:

RQ1: Can ENLIL generate a balanced distribution of operating contexts with respect to system behavior more efficiently than Monte Carlo sampling?

RQ2: Can operating contexts from ENLIL train a more accurate behavior oracle when compared to an oracle trained by Monte Carlo contexts?

A. Datasets

Training and validation images have been acquired from a forward-facing camera mounted on an automobile, sourced from the KITTI Vision Benchmark Suite [46] and the Waymo Open Dataset [51] for autonomous driving. These datasets have been chosen, because they are publicly available, well-documented, and contain high-quality images taken from real-world driving scenarios. A majority of images from each dataset depict clear-weather scenes, and therefore, it is unknown how a DNN trained only by default data will react under introduced adverse weather conditions. Validating our approach on two independently collected datasets helps to illustrate that our techniques are not dataset dependent.

B. Experimental Setup

Object detection DNNs for this study have been implemented using Tensorflow [52] machine learning libraries and a RetinaNet [53] [54] architecture. When trained and evaluated with unaltered KITTI images, the object detector achieved a mean recall of 88% overall (94% for vehicles and 70% for pedestrians/cyclists). For Waymo images, the object detector achieved a mean recall of 95% overall (95% for vehicles and 91% for pedestrians/cyclists).

This study considers an assortment of weather conditions. Specific conditions include *fog* conditions, *lens flare* on the camera lens, *raindrops* placed on the camera lens, and *rainfall* occluding portions of the image. These weather conditions are simulated by performing parameterized image processing techniques on real-world images taken from the original KITTI and Waymo datasets. Examples of each weather condition can be seen in Fig. 6. An unaltered image from the KITTI dataset is shown in Fig. 6(a), followed by example contexts of fog, lens flares, raindrops, and rainfall in Figs. 6(b), 6(c), 6(d), and 6(e), respectively. Table I lists parameters for each weather condition with respective value ranges. Each set of sample values drawn for these parameters is considered a unique operating context for the corresponding weather condition.

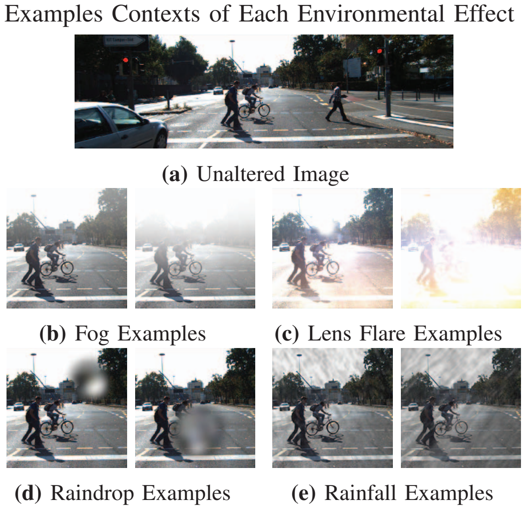


Fig. 6: Examples of each environmental effect: (a) shows an unaltered image, (b) shows examples of fog applied to the scene, (c) shows examples of lens flare applied to the camera, (d) shows examples of raindrops on the camera lens, and (e) shows examples of rainfall applied to the scene.

The behavior categories referenced in these experiments are the same as those specified in Sec. III. Thresholds have been set such that CAT 1 ("expected object misdetection") is assigned when an introduced phenomenon causes recall to decrease more than 5% over a set of validation images, CAT 2 ("expected pedestrian misdetection") is assigned when there is greater than a 1% decrease in recall for pedestrians over a set of validation images, and CAT 3 ("expected successful detection") is assigned for all other cases. Categories are assigned to a given operating context after executing the evaluation procedure in ***Step 2*** (Sec. III) to compare the recall for a set of validation images with and without the newly-introduced phenomenon.

TABLE I: Parameters for Environmental Effects

PARAMETER	PERMISSIBLE VALUES
$fog\_density$	0 to 100%
$fog\_intensity$	0 to 100%
$lensflare\_blur\_radius$	3 to 20% image width
$lensflare\_chromatic\_distortion$	0 to 100%
$lensflare\_ghost\_spacing$	0 to 100% image radius
$lensflare\_halo\_width$	0 to 100% image radius
$lensflare\_light\_x$	0 to 100% image width
$lensflare\_light\_y$	0 to 50% image height
$lensflare\_light\_radius$	0 to 10% image width
$raindrop\_x$	0 to 100% image width
$raindrop\_y$	0 to 100% image height
$raindrop\_radius$	0 to 33% image width
$raindrop\_blur$	3 to 6% image width
$rainfall\_angle$	$\pi/4$ to $3\pi/4$ radians
$rainfall\_density$	0 to 100%
$rainfall\_intensity$	0 to 100%
$rainfall\_seed$	0 to 100

C. *Evaluation of Context Generation (**RQ1**)*

To evaluate the distribution of operating contexts generated by ENLIL, multiple archives have been created for each environmental condition, with contexts divided into the specified behavior categories (CAT 1, CAT 2, and CAT 3). Separate trials were executed with alternative random seeds and varying archive sizes, ranging from 300 to 1,200 contexts. The distribution of contexts in each archive are compared to a Monte Carlo selection of contexts (i.e., parameters in Table I have been randomly sampled).

Tables II and III show the distribution of contexts generated by ENLIL and Monte Carlo sampling when applied to the KITTI and Waymo datasets, respectively. All archives produced by ENLIL were found to contain an even distribution of contexts belonging to each behavior category. In contrast, a Monte Carlo sampling method was found to consistently produce an uneven distribution of contexts with respect to each behavior category. Tables II and III also show the number of samples each method would need to generate in order to achieve an even distribution at each respective archive size. For ENLIL, this is the number of candidate contexts produced by its EA. For the Monte Carlo method, this number is a projection based on the observed distribution. A key benefit for using ENLIL over a Monte Carlo method is its ability to efficiently uncover an equal proportion of contexts in cases where one behavior category is significantly less likely to occur (i.e., corner cases). To train a model that can effectively classify contexts into each behavior category, it is necessary to have relatively equal representation for each data category.

TABLE II: Context Generation Results for KITTI Data

		ENLIL		MONTE CARLO	
EFFECT	ARCHIVE SIZE	RATIO CAT 1:2:3	SAMPLES TO BALANCE	RATIO CAT 1:2:3	SAMPLES TO BALANCE
FOG	300	1:1:1	574	12:1:4	1,250
	600	1:1:1	910	25:1:7	4,500
	1,200	1:1:1	1,691	19:1:6	15,285
LENS FLARE	300	1:1:1	500	1:1:15	1,810
	600	1:1:1	817	1:1:15	3,384
	1,200	1:1:1	1,431	1:1:13	6,286
RAIN DROP	300	1:1:1	438	1:1:4	703
	600	1:1:1	810	1:1:4	1,312
	1,200	1:1:1	1,428	1:1:4	2,712
RAIN FALL	300	1:1:1	610	22:1:1	2,457
	600	1:1:1	883	19:1:1	4,279
	1,200	1:1:1	1,581	19:1:1	8,581

TABLE III: Context Generation Results for Waymo Data

		ENLIL		MONTE CARLO	
EFFECT	ARCHIVE SIZE	RATIO CAT 1:2:3	SAMPLES TO BALANCE	RATIO CAT 1:2:3	SAMPLES TO BALANCE
FOG	300	1:1:1	528	4:2:1	711
	600	1:1:1	789	5:2:1	1,670
	1,200	1:1:1	1,437	6:2:1	3,650
LENS FLARE	300	1:1:1	482	1:7:55	6,667
	600	1:1:1	797	1:7:51	11,966
	1,200	1:1:1	1,409	1:8:50	24,233
RAIN DROP	300	1:1:1	481	1:4:17	2,566
	600	1:1:1	826	1:4:18	4,690
	1,200	1:1:1	1,421	1:3:16	8,103
RAIN FALL	300	1:1:1	533	8:1:1	1,087
	600	1:1:1	826	8:1:1	2,094
	1,200	1:1:1	1,473	9:1:1	4,362

D. *Evaluation of Predictive Behavior Models (**RQ2**)*

To compare the usefulness of ENLIL-generated operating contexts to Monte Carlo context, separate behavior oracles have been trained with each archive generated by each method using the procedure described in ***Step 3*** (Sec. III). The accuracy of each behavior oracle has been measured for an independent set of test images with random environmental contexts applied, with mean results shown in Tables IV and V for the KITTI and Waymo datasets, respectively.

For each archive generated, the accuracy of behavior oracles trained with ENLIL-generated contexts were either comparable to or better than those trained with Monte Carlo contexts. For smaller archive sizes, ENLIL behavior oracles were significantly more accurate. Since the distribution of Monte Carlo contexts are heavily biased, more examples are required to adequately represent all behavior categories, making behavior classification difficult for small sample sizes. These results show that ENLIL is more likely to produce an accurate behavior oracle when compared to an oracle trained only with randomly-sampled contexts, independent of archive size.

TABLE IV: Accuracy of KITTI Behavior Oracles

		Enlil		Monte Carlo	
	Archive	Accuracy (%)		Accuracy (%)	
Effect	Size	Cat 1 / 2 / 3	**Mean**	Cat 1 / 2 / 3	**Mean**
Fog	300	97.9 / 34.2 / 87.3	**73.1**	95.8 / 0.0 / 93.1	**63.0**
	600	98.7 / 89.0 / 73.5	**87.1**	96.6 / 0.0 / 93.1	**63.2**
	1,200	98.5 / 90.1 / 76.4	**88.3**	96.9 / 0.0 / 93.2	**63.4**
Lens Flare	300	84.6 / 53.9 / 82.3	**73.6**	32.4 / 0.0 / 95.6	**42.7**
	600	85.4 / 67.7 / 83.0	**78.7**	50.5 / 0.0 / 92.6	**47.7**
	1,200	85.0 / 75.3 / 84.5	**81.6**	75.0 / 40.2 / 94.8	**70.0**
Rain drop	300	87.9 / 67.9 / 92.0	**82.6**	84.3 / 26.9 / 96.5	**69.2**
	600	91.8 / 76.1 / 85.4	**84.4**	83.0 / 57.1 / 94.5	**78.2**
	1,200	89.3 / 79.2 / 88.3	**85.6**	93.2 / 60.1 / 79.7	**77.5**
Rain fall	300	87.9 / 75.1 / 26.5	**63.1**	99.6 / 1.6 / 41.6	**47.6**
	600	88.2 / 74.5 / 26.6	**63.1**	99.3 / 3.3 / 45.7	**49.4**
	1,200	89.7 / 73.3 / 27.3	**63.4**	96.6 / 4.7 / 71.5	**57.6**

TABLE V: Accuracy of Waymo Behavior Oracles

		Enlil		Monte Carlo	
	Archive	Accuracy (%)		Accuracy (%)	
Effect	Size	Cat 1 / 2 / 3	**Mean**	Cat 1 / 2 / 3	**Mean**
Fog	300	95.5 / 78.3 / 72.4	**82.1**	98.7 / 75.5 / 72.2	**82.1**
	600	97.2 / 83.9 / 69.0	**83.4**	98.7 / 84.9 / 62.0	**81.9**
	1,200	97.8 / 81.1 / 76.4	**85.1**	98.0 / 88.2 / 53.3	**79.8**
Lens Flare	300	64.8 / 21.1 / 89.7	**58.5**	10.6 / 0.9 / 99.9	**37.1**
	600	80.3 / 38.4 / 75.8	**64.8**	21.8 / 2.5 / 99.4	**41.2**
	1,200	80.4 / 40.3 / 80.3	**67.0**	51.2 / 3.8 / 99.2	**51.4**
Rain drop	300	69.8 / 44.5 / 65.7	**60.0**	0.0 / 0.0 / 99.9	**33.3**
	600	75.3 / 53.6 / 86.3	**71.7**	13.8 / 6.3 / 98.3	**39.5**
	1,200	77.0 / 52.8 / 88.4	**72.8**	56.1 / 37.1 / 96.7	**63.3**
Rain fall	300	82.5 / 58.2 / 24.3	**55.0**	97.9 / 12.7 / 24.4	**45.0**
	600	81.5 / 69.6 / 25.1	**58.8**	97.9 / 10.3 / 32.6	**46.9**
	1,200	82.1 / 60.7 / 36.0	**59.6**	97.7 / 12.7 / 37.9	**49.4**

E. Threats to Validity

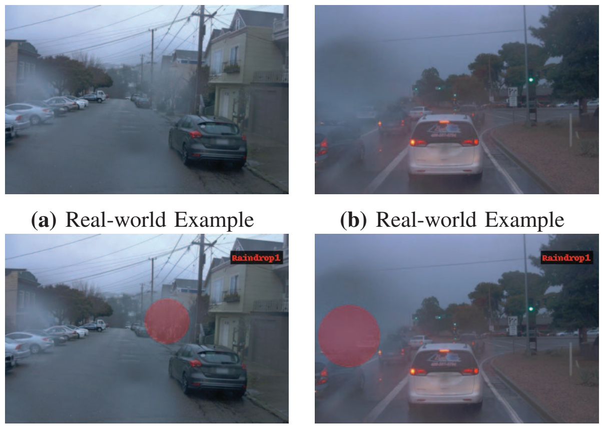
Since these experiments use synthetically generated data, results may not directly apply to the real world. Use of ENLIL is recommended when the "reality gap" [55] is insignificant. The simulated phenomena have been implemented to visually match real-world analogues and may help guide follow-on studies under real weather conditions to alleviate concerns about the reality gap. Fig. 7 has been included to demonstrate an ENLIL-ORACLE applied to images of real raindrops. In these examples, ENLIL-ORACLE was able to correctly perceive the presence of a raindrop and predict that it could possibly interfere with the performance of the object detector.

The use of stochastic functions may influence the results of this study. Since the DNNs in this study have been trained by a stochastic gradient descent method and since Monte Carlo sampling is also stochastic, some variability in the results presented may be expected. For variation, all results reported have been averaged over multiple runs.

V. Related Work

This section overviews related work in model testing, automated testing for DNNs, and predictive behavior modeling.

ENLIL-ORACLE Demonstration on Real Raindrops



(a) Real-world Example **(b)** Real-world Example

(c) "Expected misdetection" **(d)** "Expected misdetection"

Fig. 7: Proof-of-concept validation of ENLIL-ORACLE on real-world examples. (a) and (b) show images with real raindrops from the Waymo dataset [51]. (c) and (d) show that ENLIL-ORACLE was able to perceive the possibly disruptive raindrop in each image (highlighted in red).

A. Model Testing

Automated techniques are essential when evaluating the numerous possible interactions between a cyber-physical system and its physical environment. *Model testing* has been advocated to shift testing away from the physical implementation of a cyber-physical system to a more abstract representation [56]. The removal of hardware constraints facilitates a more efficient search of the system's interactions within a simulated environment. Related tools such as Veritas [2] and Proteus [1] have been introduced to perform adaptive run-time testing of a system by monitoring *utility functions* [48] [57] that measure a degree of system *requirements satisfaction* [58]. ENLIL's approach can be considered a form of *statistical model checking* [59], where multiple runs of a system are evaluated under varying operating conditions and statistical evidence is gathered to support claims about the system's behavior. For example, when evaluating a DNN, statistical behavior metrics may include accuracy or a *confusion matrix* based on *validation data*. By model testing with a widely diverse set of contexts, ENLIL can efficiently gather unique *counterexamples* [60] that can infer a more generalized description of how the system interacts with its environment.

B. Automated Testing for Deep Learning

Alternative methods have been proposed to address the problem of automatic test generation for DNNs, such as DeepXplore [61], DeepTest [62], DeepGauge [63], DeepRoad [64], and DeepConcolic [65]. These techniques generate test inputs by maximizing coverage over a DNN's topology. DeepXplore and DeepTest measure neuron coverage over an entire DNN for a given test input. However, preliminary results show that test inputs created to maximize neuron coverage are not able to account for all possible behaviors that a DNN may exhibit and that neuron coverage alone can be insufficient for finding corner cases [66]. Finer variants of neuron coverage have also been considered with DeepGauge, where test cases are generated to maximize coverage over subsections or specific activation patterns within a DNN at the layer-level. DeepConcolic supports neuron coverage but also modified condition/decision condition (MC/DC) variants [67]. Further studies have shown that even techniques that attempt to maximize these more fine-grained structural coverage metrics can be misleading, since selected subsets of data with significantly different error rates can produce similar amounts of coverage [68]. In contrast to these related techniques, ENLIL enables a user to select specific observable behavior metrics (e.g., neuron coverage, activation patterns, precision, recall, F-scores, etc.) and target those most relevant for the task at hand, when generating example operating contexts for the DNN under evaluation.

DNN testing methods generally rely on a generative procedure to construct synthetic test data. Methods like DeepTest [62] and DeepXplore [61] depend on a manually crafted procedure to transform real data into a manipulated, synthetic form (e.g., an image processing procedure to add fog to an existing image). Using an algorithmic transformation procedure gives developers direct control over how synthetic data is generated, but it also relies on the assumption that a transformation procedure can be developed to accurately reproduce the desired phenomenon.

Alternatively, methods like DeepRoad [64] use GANs [7] that perform Image-to-Image translation [8] to transform the appearance of any input data to match real-world examples of the desired phenomenon (e.g., transform a clear-weather image of a road into a snowy version). Image-to-Image translation does not require a formal understanding or model of the introduced environmental condition, but it does require an adequate collection of real-world examples (i.e., a sufficiently large collection of snowy images). Use of GANs in this manner gives developers less control over how the resulting synthetic data will appear. A GAN's output is entirely dependent on the quality and distribution of its given training images, and furthermore, whatever features a GAN learns to focus on during its training phase are subject to the same issues of robustness and interpretability suffered by all DNNs. ENLIL is intended for use when insufficient data has been collected to train a GAN for an environmental condition of interest but a parameterized transformation procedure can be constructed to synthetically reproduce the environmental phenomenon.

C. Predictive Behavior Modeling

ENLIL follows prior research in the use of machine learning to infer behavior models for software testing. Walkinshaw *et al.* [69] [70] described a framework that uses model inference to emulate software components. In their work, the inferred behavior model is trained on functional inputs and outputs of a tested software component. This paper describes a comparable conceptual framework but differs in a few key aspects. First, ENLIL interacts with the external interface of a cyber-physical LES, rather than modeling the direct input and output of a single unit software function or component. ENLIL trains a model that emulates a cyber-physical system's sensor data and its expected behavior patterns. Second, based on the hypothesis that a diverse training set will enable accurate behavior models to be trained more efficiently, ENLIL uses a diversity-driven method to generate training examples for its behavior models, rather than a greedy optimization method. Finally, a major motivation for ENLIL is to assess the robustness of DNNs and model the impact of uncertain conditions on DNNs, which has not been addressed in the work of Walkinshaw *et al.*

VI. Conclusion

This paper describes how the ENLIL framework enables the creation of a behavior oracle (ENLIL-ORACLE) that can predict how an LES will behave under conditions of uncertainty. ENLIL has been demonstrated to uncover a wide variety of training contexts that result in behaviors that would be underrepresented by Monte Carlo sampling. Furthermore, ENLIL can train behavior models more efficiently (i.e., requiring fewer training examples) to achieve equivalent or greater accuracy than Monte Carlo methods for introducing adversity. By using novelty search to achieve a more diverse sampling, ENLIL requires additional overhead in comparison to Monte Carlo sampling. The number of parameters in the operating specification, the archive size, and the complexity of the given evaluation procedure are major factors to consider when using ENLIL. For this study, each trial took approximately one additional hour for context generation with ENLIL versus Monte Carlo sampling on a single workstation. Since these procedures are intended to be executed at design time, the additional overhead is considered tolerable.

ENLIL-ORACLE predictive behavior models can potentially inform developers about training gaps and system limitations that require additional development to enable proactive mitigation strategies. Self-adaptive systems can use an ENLIL-ORACLE to determine how the managed system will react to conditions for which it has not explicitly been trained. For example, an adaptation manager can harness an ENLIL-ORACLE to anticipate failure of a learning component and trigger preventative actions at run time. Future work will explore how to fully integrate ENLIL into a MAPE-K [45] loop for self-adaptation. Additionally, future empirical studies will explore alternative types of adverse phenomena (e.g., weather effects in combination and/or non-weather oriented forms of adversity).

Acknowledgments

This work has been supported in part by grants from the National Science Foundation (NSF) (DBI-0939454). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF or other sponsors.

References

[1] E. M. Fredericks and B. H. Cheng, "Automated Generation of Adaptive Test Plans for Self-adaptive Systems," in *Proc. 10th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*. ACM, 2014, p. 157–168.

[2] E. M. Fredericks, B. DeVries, and B. H. Cheng, "Towards Run-time Adaptation of Test Cases for Self-adaptive Systems in the Face of Uncertainty," in *Proc. 9th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*. ACM, 2014, p. 17–26.

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson Education, 2010.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT, 2016.

[5] M. Borg, C. Englund, K. Wnuk, B. Duran, C. Levandowski, S. Gao, Y. Tan, H. Kaijser, H. Lönn, and J. Törnqvist, "Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry," *Journal of Automotive Software Engineering*, vol. 1, pp. 1–19, 2019.

[6] G. Calikli and A. Bener, "Empirical Analyses of the Factors Affecting Confirmation Bias and the Effects of Confirmation Bias on Software Developer/Tester Performance," in *Proc. 6th Int. Conf. on Predictive Models in Software Engineering (PROMISE 2010)*. ACM, 2010.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*. Curran Associates, 2014, vol. 27, pp. 2672–2680.

[8] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," in *Proc. IEEE Int. Conf. on Computer Vision (ICCV 2017)*. IEEE, 2017, pp. 2242–2251.

[9] S. N. Kramer, *The Sumerians: Their History, Culture, and Character*. The University of Chicago Press, 1963.

[10] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The Oracle Problem in Software Testing: A Survey," *Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2016)*. IEEE, Jun 2016, pp. 770–778.

[12] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object Detection With Deep Learning: A Review," *Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[13] S. Ghosh, N. Das, I. Das, and U. Maulik, "Understanding Deep Learning Techniques for Image Segmentation," *ACM Comput. Surv.*, vol. 52, no. 4, Aug 2019.

[14] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent Trends in Deep Learning Based Natural Language Processing [Review Article]," *Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[15] A. F. Del Carpio and L. B. Angarita, "Trends in Software Engineering Processes using Deep Learning: A Systematic Literature Review," in *Proc. 46th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA 2020)*, 2020.

[16] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation," in *Proc. 40th IEEE/ACM Int. Conf on Software Engineering (ICSE 2018)*, 2018, pp. 665–676.

[17] G. Zhao and J. Huang, "DeepSim: Deep Learning Code Functional Similarity," in *Proc. 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, 2018, p. 141–151.

[18] R. S. Malik, J. Patra, and M. Pradel, "NL2Type: Inferring JavaScript Function Types from Natural Language Information," in *Proc. 41st Int. Conf. on Software Engineering (ICSE 2019)*. IEEE, 2019, p. 304–315.

[19] G. Lin, J. Zhang, W. Luo, L. Pan, Y. Xiang, O. De Vel, and P. Montague, "Cross-Project Transfer Representation Learning for Vulnerable Function Discovery," *Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3289–3297, 2018.

[20] M. Abuhamad, J.-s. Rhim, T. AbuHmed, S. Ullah, S. Kang, and D. Nyang, "Code Authorship Identification Using Convolutional Neural Networks," *Future Generation Computer Systems*, vol. 95, pp. 104–115, 2019.

[21] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A Survey of Android Malware Detection with Deep Neural Models," *ACM Comput. Surv.*, vol. 53, no. 6, Dec 2020.

[22] Y. Xiao, J. Keung, K. E. Bennin, and Q. Mi, "Improving Bug localization with Word Embedding and Enhanced Convolutional Neural Networks," *Information and Software Technology*, vol. 105, pp. 17–29, 2019.

[23] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and Y. Gao, "Improving Code Readability Classification using Convolutional Neural Networks," *Information and Software Technology*, vol. 104, pp. 60–71, 2018.

[24] X. Huo, Y. Yang, M. Li, and D. Zhan, "Learning Semantic Features for Software Defect Prediction by Code Comments Embedding," in *Proc. IEEE Int. Conf on Data Mining (ICDM 2018)*. IEEE, 2018, pp. 1049–1054.

[25] S. Wang, T. Liu, and L. Tan, "Automatically Learning Semantic Features for Defect Prediction," in *Proc. 38th Int. Conf. on Software Engineering (ICSE 2016)*. ACM, 2016, p. 297–308.

[26] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2018)*. IEEE, 2018, pp. 1625–1634.

[27] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *CoRR*, vol. abs/1412.6572, 2014, appeared in ICLR 2015.

[28] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," *CoRR*, vol. abs/1611.01236, 2016, appeared in ECAI 2016.

[29] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," *CoRR*, vol. abs/1312.6199, 2013, appeared in ICLR 2014.

[30] A. Nguyen, J. Yosinski, and J. Clune, "Deep Neural Networks Are Easily Fooled: High Confidence Predictions For Unrecognizable Images," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2015)*. IEEE, 2015, pp. 427–436.

[31] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges Toward Responsible AI," *Information Fusion*, vol. 58, pp. 82–115, Jun 2020.

[32] J. Jo and Y. Bengio, "Measuring the Tendency of CNNs to Learn Surface Statistical Regularities," *CoRR*, vol. abs/1711.11561, 2017.

[33] F. Yu, Z. Qin, C. Liu, L. Zhao, Y. Wang, and X. Chen, "Interpreting and Evaluating Neural Network Robustness," in *Proc. 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*. ijcai.org, 2019, pp. 4199–4205.

[34] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-Based Software Engineering: Trends, Techniques and Applications," *ACM Comput. Surv.*, vol. 45, no. 1, Dec 2012.

[35] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Springer-Verlag, 2015.

[36] P. McMinn, "Search-Based Software Testing: Past, Present and Future," in *Proc. 4th Int. Conf. on Software Testing, Verification and Validation Workshops (ICST 2011)*. IEEE, 2011, pp. 153–163.

[37] G. Fraser and A. Arcuri, "EvoSuite: Automatic Test Suite Generation for Object-oriented Software," in *Proc. 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011)*. ACM, 2011, p. 416–419.

[38] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for android applications," in *Proc. 25th Int. Symposium on Software Testing and Analysis (ISSTA 2016)*. ACM, 2016, p. 94–105.

[39] J. Lehman, "Evolution Through the Search for Novelty," Ph.D. dissertation, University of Central Florida, 2012.

[40] J. Lehman and K. Stanley, "Novelty Search and the Problem with Objectives," in *Genetic Programming Theory and Practice IX. Genetic and Evolutionary Computation*. Springer, Oct 2011, pp. 37–56.

[41] M. Boussaa, O. Barais, G. Sunyé, and B. Baudry, "A Novelty Search Approach for Automatic Test Data Generation," in *Proc. 8th Int. Workshop on Search-Based Software Testing (SBST 2015)*. IEEE, 2015, pp. 40–43.

[42] A. J. Ramirez, A. C. Jensen, B. H. Cheng, and D. B. Knoester, "Automatically Exploring How Uncertainty Impacts Behavior of Dynamically Adaptive Systems," in *Proc. 26th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2011)*. IEEE, 2011, p. 568–571.

[43] M. A. Langford and B. H. Cheng, "Enhancing Learning-Enabled Software Systems to Address Environmental Uncertainty," in *Proc. 16th IEEE Int. Conf on Autonomic Computing (ICAC 2019)*. IEEE, 2019, pp. 115–124.

[44] M. A. Langford, G. A. Simon, P. K. McKinley, and B. H. Cheng, "Applying Evolution and Novelty Search to Enhance the Resilience of Autonomous Systems," in *Proc. 14th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2019)*. ACM, 2019, pp. 63–69.

[45] J. O. Kephart and D. M. Chess, "The Vision Of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[46] A. Geiger, P. Lenz, and R. Urtasun, "Are We Ready For Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2012)*. IEEE, 2012, pp. 3354–3361.

[47] M. Sokolova and G. Lapalme, "A Systematic Analysis of Performance Measures for Classification Tasks," *Information Processing & Management*, vol. 45, pp. 427–437, Jul 2009.

[48] S.-W. Cheng, "Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation," Ph.D. dissertation, Carnegie Mellon University, 2008.

[49] H. He and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sept 2009.

[50] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proc. 27th Int. Conf. on Machine Learning (ICML 2010)*. Omnipress, 2010, p. 807–814.

[51] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in Perception for Autonomous Driving: Waymo Open Dataset," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2020)*. IEEE, 2020, pp. 2446–2454.

[52] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *Proc. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2016)*. USENIX, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[53] H. Gaiser, M. de Vries, V. Lacatusu, vcarpani, A. Williamson, E. Liscio, Andr/'as, Y. Henon, jjiun, C. Gratie, M. Morariu, C. Ye, M. Zlocha, B. Weinstein, R. M. de Andrade, P. Conceição, A. Pacha, hannesedvartsen, D. Shahrokhian, W. Fang, M. Clark, meagerYak, I. Jordal, M. Van Sande, Jin, Etienne-Meunier, A. Grigorev, G. Erhard, E. Ramos, and D. Dowling, "fizyr/keras-retinanet 0.5.1," Jun 2019.

[54] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *Proc. IEEE Int. Conf on Computer Vision (ICCV 2017)*. IEEE, 2017, pp. 2999–3007.

[55] N. Jacobi, P. Husbands, and I. Harvey, "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics," in *Proc. 3rd European Conf. on Advances in Artificial Life*. Springer-Verlag, 1995, p. 704–720.

[56] L. Briand, S. Nejati, M. Sabetzadeh, and D. Bianculli, "Testing the Untestable: Model Testing of Complex Software-Intensive Systems," in *Proc. 38th Int. Conf. on Software Engineering Companion (ICSE 2016)*. ACM, 2016, p. 789–792.

[57] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility Functions in Autonomic Systems," in *Proc. Int. Conf. on Autonomic Computing (ICAC 2004)*. IEEE, 2004, pp. 70–77.

[58] P. deGrandis and G. Valetto, "Elicitation and Utilization of Application-Level Utility Functions," in *Proc. 6th Int. Conf on Autonomic Computing (ICAC 2009)*. ACM, 2009, p. 107–116.

[59] A. Legay, B. Delahaye, and S. Bensalem, "Statistical Model Checking: An Overview," in *Runtime Verification. Lecture Notes in Computer Science (RV 2010)*. Springer, 2010, vol. 6418, pp. 122–135.

[60] M. Chechik and A. Gurfinkel, "A Framework for Counterexample Generation and Exploration," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, no. 5–6, pp. 429–445, 2007.

[61] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated Whitebox Testing of Deep Learning Systems," in *Proc. 26th Symposium on Operating Systems Principles (SOSP 2017)*. ACM, 2017, p. 1–18.

[62] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars," in *Proc. 40th Int. Conf. on Software Engineering (ICSE 2018)*. ACM, 2018, p. 303–314.

[63] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, and Y. Liu, "DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems," in *Proc. 33rd ACM/IEEE Int. Conf. on Automated Software Engineering (ASE 2018)*. ACM, 2018, p. 120–131.

[64] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems," in *Proc. 33rd ACM/IEEE Int. Conf. on Automated Software Engineering (ASE 2018)*. ACM, 2018, p. 132–142.

[65] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "DeepConcolic: Testing and Debugging Deep Neural Networks," in *Proc. 41st Int. Conf. on Software Engineering (ICSE 2019)*. IEEE, 2019, p. 111–114.

[66] J. Sekhon and C. Fleming, "Towards Improved Testing for Deep Learning," in *Proc. 41st Int. Conf. on Software Engineering (ICSE-NIER 2019)*. IEEE, 2019, p. 85–88.

[67] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic Testing for Deep Neural Networks," in *Proc. 33rd ACM/IEEE Int. Conf. on Automated Software Engineering (ASE 2018)*. ACM, 2018, p. 109–119.

[68] Z. Li, X. Ma, C. Xu, and C. Cao, "Structural Coverage Criteria for Neural Networks Could Be Misleading," in *Proc. 41st Int. Conf on Software Engineering (ICSE-NIER 2019)*. IEEE, 2019, p. 89–92.

[69] G. Fraser and N. Walkinshaw, "Assessing and Generating Test Sets in Terms of Behavioural Adequacy," *Softw. Test. Verif. Reliab.*, vol. 25, no. 8, p. 749–780, Dec 2015.

[70] P. Papadopoulos and N. Walkinshaw, "Black-Box Test Generation from Inferred Models," in *Proc. 4th Int. Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2015)*. IEEE, 2015, p. 19–24.