Themis: Automatic and Efficient Deep Learning System Testing with Strong Fault Detection Capability

Anonymous Author(s)

Abstract

Deep Learning Systems (DLSs) have been widely applied in safety-critical tasks such as autopilot. However, when a perturbed input (e.g., by raindrop effects) is fed into a DLS for inference, the DLS often has incorrect outputs (i.e., faults). DLS testing techniques (e.g., DeepXplore) detect such faults by generating perturbed inputs to explore data flows that induce faults. Since a DLS often has infinitely many data flows, existing techniques require developers to manually specify a set of activation values in a DLS's neurons for exploring fault-inducing data flows. Unfortunately, recent studies show that such manual effort is tedious and can detect only a tiny proportion of fault-inducing data flows.

We present Themis, the first automatic DLS testing system, which attains strong fault detection capability by ensuring a full coverage of fault-inducing data flows at a high probability. Themis carries a new workflow for automatically and systematically revealing data flows whose internal neurons' outputs vary substantially when the inputs are slightly perturbed, as these data flows are likely fault-inducing. We evaluated Themis on ten different DLSs and found that on average the number of faults detected by Themis was 3.78X more than four notable DLS testing techniques. By retraining all evaluated DLSs with the detected faults, Themis also increased (regained) these DLSs' accuracies on average 14.7X higher than all baselines. Themis's source code and all evaluation results are available on github.com/ase648/ase648.

1 Introduction

Deep Learning Systems (DLSs) have been widely applied in safety-critical tasks such as autopilot and smart cities [7, 11, 21, 28, 36]. However, when a DLS (e.g., an autopilot system) is deployed in a real-world environment (e.g., a crowded city), the DLS's input (e.g., a road condition image of the crowded city) is often perturbed by environmental noise such as raindrops and fog effects, causing incorrect values and disaster [11, 21, 28, 36]. The incorrect output value of a DLS caused by perturbation on the DLS's input is defined as the DLS's fault [16].

Since a DLS's faults greatly undermine the DLS's reliability, a DLS must be systematically tested in order to detect as many faults as possible [11, 21, 28, 36]. These faults are essential for further developing a DLS (i.e., improving the DLS's accuracy by retraining the DLS with perturbed inputs which lead to faults).

The rationale of existing DLS testing is inspired by conventional software testing. In conventional software testing [25, 31, 42], a testing technique generates a set of inputs (known as the *test set*) to explore diverse data flows of a software (e.g., statements or branches), especially the data flows which likely lead to the software's faults such as exceptions or crashes (these data flows are called "fault-inducing data flows"). When all data flows of the software have been explored (i.e., the test coverage metric adopted by the testing technique, such as code coverage, reaches 100%), the testing process is considered completed, and the testing technique stops.

Similarly, existing DLS testing techniques (e.g., DeepXplore [36], DeepGauge [28], DeepImportance [11], and Surprise Adequacy for Deep Learning System [21]) generate a test set (a set of perturbed inputs) to explore a DLS's data flows, where the DLS's data flow is defined as a set of numerical values: each of the numerical value is one of the DLS's neurons' output value (a DLS's neuron is a "function", and the output value of a neuron is called "activation value" [28, 36]) corresponding to a DLS's input. A DLS testing instance generates noise test set by perturbing test set with the same type of environmental noise (e.g., raindrop effect) of various levels of noise strength (e.g., various raindrop densities).

However, since the activation values of a DLS's neurons are discrete numbers from negative infinity to positive infinity, generating a test set to make a DLS's neurons output all possible sets of activation values is prohibitively inefficient. To mitigate such inefficiency in DLS testing, for each type of environmental effects (noise), existing techniques (DeepXplore [36], DeepGauge [28], DeepImportance [11], and Surprise Adequacy for Deep Learning System [21]) require a DLS's developer to manually specify likely sets of fault-inducing activation values in all neurons.

For example, DeepXplore [36] requires developers to manually partition a neuron's outputs into two segments (i.e., outputs larger than a threshold and outputs smaller than a threshold, where the threshold has to be manually specified by the developers). Then, DeepXplore [36] generates test inputs to cover each of the two segments for all neurons. Nevertheless, recent work [16] showed that manually specifying the threshold is error-prone, and such manual effort is often overly coarse-grained, identifying a DLS's all activation values that may induce faults for each type of environmental effects is fundamentally difficult, and not all values will induce faults on all neurons. Worse, existing DLS testing techniques [11, 36] often can't attain strong fault detection capability (strong fault detection capability means the correlation between the error rate of a DLS and the number of the DLS's faults detected is larger than 0.7).

We believe that the root cause behind the limitations of the existing techniques is that the triggering condition and the total number of a DLS's fault-inducing data flows are unknown [11, 16, 21, 26, 28, 36, 45]. Hence, existing techniques inevitably require manual expert effort in inferring a set of fault-inducing data flows, in order to guide the exploration of a DLS's data flows and the computation of test coverage (i.e., the proportion of the fault-inducing data flows being explored from the DLS during testing). Unfortunately, despite these advancements [11, 16, 21, 26, 28, 36, 45], in principle, manual effort is far away from guaranteeing that these techniques can explore a full coverage of fault-inducing data flows from a DLS, while the full coverage is an essential and sufficient condition for a testing technique to attain strong fault-detection capability [16, 26, 45]. Overall, an automated DLS testing technique that can theoretically explore a full coverage of fault-inducing data flows is highly desirable but vacant.

The main observation of the paper is that, a fault-inducing data flow in a DLS is often *sensitive* to perturbation on a DLS's input. We denote the data flow with respect to a DLS M and an input I as $DF(M, I)$. For a DLS (M), a clean input (I) and its perturbed input (I'), we consider a data flow $DF(M, I')$ *sensitive* to perturbation if

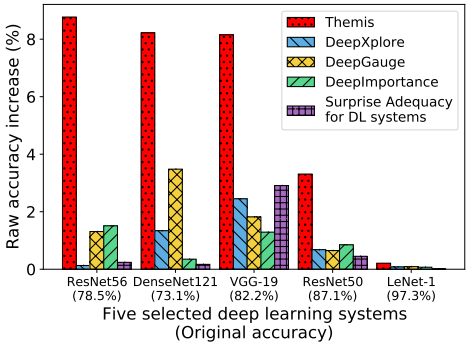


Figure 1: The increases in DLS's accuracy brought by Themis were on average 14.7X more than the increases in DLS's accuracy brought by baselines.

the difference of $DF(M, I')$ and $DF(M, I)$ (denoted as $DF(M, I') - DF(M, I)$) is large, given that I' and I just have slight difference (e.g., I' and I differ by light raindrops). Intuitively, sensitive $DF(M, I')$ inevitably causes $M(I')$ and $M(I)$ to be different, confirmed in our evaluation (Figure 5b).

With this observation, we present Themis, a DLS testing technique that systematically explores fault-inducing data flows guided by data flows' sensitivity: the difference between $DF(M, I')$ and $DF(M, I)$ with respect to the difference between I' and I. Themis leverages math optimization techniques (e.g., Gradient Descent) to adjust the intensity of the perturbation (e.g., raindrops' densities) on I, in order to generate a new test set of I', which maximizes $DF(M, I') - DF(M, I)$. Hence, Themis maximizes the likelihood that $DF(M, I')$ will lead to a fault. With this new workflow, Themis explores faults from a DLS without the necessity of either manual effort or exploring all data flows of a DLS.

One major challenge for Themis to achieve strong fault detection capability is how Themis infers the coverage of fault-inducing data flows being explored by Themis's generated test set during testing (i.e., how Themis computes the test coverage metric). This is because the actual number of a DLS's fault-inducing data flows on all inputs is unknown.

To tackle this challenge, we summarize existing AI theories [7, 27, 39, 46] to show that, a DLS's $DF(M, I') - DF(M, I)$ on all inputs often follows normal distribution: for all clean inputs, most noise added to these inputs will lead to similar influence on M's final outputs, and only a tiny portion of noise will lead to outlying influence. Hence, once Themis's workflow infers that $DF(M, I') - DF(M, I)$ converges to a normal distribution driven by Themis's generated test set, according to these theories, Themis has explored a statistically full coverage of fault-inducing data flows (i.e., only a tiny portion of fault-inducing noise and their data flows were missed by Themis), and Themis's testing instance can complete. Our paper derives a proof (§4.2) to show that Themis's workflow can achieve statistically full coverage (i.e., achieving the "fault detection capability with high probability" in this paper).

We implemented Themis on PyTorch [35] and compared Themis against four recent and notable DLS testing techniques (Deepexplore [36], DeepGauge[28], DeepImportance [11] and Surprise Adequacy for Deep Learning System [21]). We evaluated Themis and these baselines on ten popular Deep Learning models (e.g., LeNet [24], VGG [41], ResNet [17]) trained on six public datasets (Cifar10 [2], ImageNet [9], Driving [44], Contagio/VirusTotal [8], Drebin [4] and MNIST [23]), which cover a complete set of datasets evaluated by the baselines [11, 21, 28, 36]. Evaluation shows that:

- Themis consistently achieved strong fault detection capability. Themis achieved a high correlation [16] (0.70 to 0.95) for all the DL models, while the baselines' correlation varied from -0.89 to 0.59 (Table 2).
- Themis detected 3.78X more faults than the baselines, when Themis's and the baselines' test coverage reach 100% (i.e., Themis and the baselines considered no more faults can be further detected from a DLS).
- By retraining the DL models with faults detected in testing, Themis increased (regained) the DL models' accuracy by 0.21% to 8.77%, while baselines increased the DL models' accuracy by 0.01% to 3.48%. Overall, Themis increased the DL models' accuracy on average 14.7X higher than the baselines (Figure 1).
- A Themis testing instance's time cost was comparable with baselines'.

Our main contribution is Themis, the first automatic DLS testing technique which can empirically (probabilistically) attain strong fault detection capability for perturbed inputs. The key novelty is Themis's new workflow for systematically exploring a DLS's fault-inducing data flows and computing its test coverage metric without manual effort, by leveraging our observation that most fault-inducing data flows are sensitive. Our theoretical analysis (§4.2) shows that Themis can explore a full coverage of fault-inducing data flows at high probability (95%), so Themis empirically attained strong fault detection capability for all the evaluated DLSs (Table 2).

Themis can promote mature DL models to be widely used by third parties with diverse environment effects, and greatly regain these models' accuracy in diverse real-world environments (e.g., specific weather conditions and lighting changes). Themis's source code and all evaluation results are available on github.com/ase648/ase648.

2 Background and Related Work

2.1 Existing DLS testing techniques are unautomated

DLS testing techniques [36, 43] (e.g., Deepexplore [36], DeepGauge[28], DeepImportance [11], and Surprise Adequacy for Deep Learning System [21]) are proposed to detect a DLS's faults. A testing instance includes a pretrained DLS M (e.g., an autopilot system), an arbitrary input I fed to M for inference (e.g., a road condition image), and the same type of environmental noise with arbitrary noise strength $E(\theta)$ (e.g., raindrop effect E, with arbitrary raindrop size θ, where θ is within a given range such as the real-world raindrop's sizes) which may be present on I. For any $M(I + E(\theta))$ which is different from $M(I)$, the $M(I + E(\theta))$ is considered a DSL's fault.

Note that an incorrect $M(I+E(\theta))$ is considered as a fault instead of a failure because existing DLS testing techniques aim to test deep learning models of a DLS rather than the entire DLS (i.e., both deep learning models and the software code of a DLS). Since in existing DLSs (e.g., industrial autonomous systems such as Autoware [20]

and Apollo [1]), deep learning models are intermediate components, the incorrect output of these deep learning models are regarded as the "fault" of a DLS [16].

These techniques generate perturbed inputs to make a DLS's neurons output diverse activation values, in order to explore the DLS's data flows (§1). Specifically, denote $N_i, i = 1, .., n$ as the ith neuron of a DLS, and $N_i(I + E(\theta)) \in \mathbb{R}$ as the activation values of the ith neuron corresponding to input $I + E(\theta)$. A DLS's data flow is defined as a set of numerical values, where each numerical value is each neuron's activation of the DLS (i.e., a data flow is defined as $\{N_i(I + E(\theta))\}$). Existing work generates diverse $I + E(\theta)$, in order to trigger diverse data flows.

Since activation values are discrete numbers from negative infinity to positive infinity, existing techniques proposed various heuristic rules for a DLS's developers to manually specify fault-inducing data flows.

For instance, DeepXplore [36] requires a DLS's developer to divide the activation values of neurons into two segments (e.g., values larger than zero and values smaller than zero), such that DeepXplore only needs to generate at minimal two perturbed inputs (we denote these inputs as $I_1 + E_1(\theta_1)$ and $I_2 + E_2(\theta_2)$), to cover the segments (e.g., $N_i(I_1 + E_1(\theta_1)) \geq 0\ \forall i$ and $N_i(I_2 + E_2(\theta_2)) > 0\ \forall i$). Other DLS testing techniques also proposed similar heuristic rules for developers to specify fault-inducing data flows. DeepGauge [28] requires the user to divide activation values into several segments, where the segments that contain activation values towards zero have smaller intervals (because most inputs of a DLS make the neurons' output activation values close to zero). DeepImportance [11] requires a DLS developer to specify fault-inducing activation values for "important" neurons only (the "important" neurons are identified via a popular technique in DL called "Layer-wise Relevance Propagation" [34]). Surprise Adequacy for Deep Learning System [21] requires a DLS's developer to specify fault-inducing activation values for neurons in softmax layers only.

2.2 Fault-inducing data flows have to be automatically identified

Overall, these techniques allow a DLS to be tested within a reasonable time (e.g., several minutes). However, substantial studies [16, 26, 45] showed that manual effort in specifying fault-inducing data flows often causes DLS testing to have unsatisfactory fault detection capability because fault-inducing data flows are often unknown (see §1).

To automatically identify fault-inducing data flows, our observation is that for any I and $E(\theta)$, the $N_i(I + E(\theta)), 1 \leq i \leq n$ which induces a fault usually has a large difference with $N_i(I), 1 \leq i \leq n$ (i.e., the value of $sum_{i=1}^{n}|N_i(I + E(\theta)) - N_i(I)|$ is large, we call this value as the *sensitivity* of a DLS's data flow on perturbed inputs). It is because large sensitivity often results in a large difference between $M(I+E(\theta))$ and $M(I)$ (i.e., $M(I+E(\theta))$ is likely to be a fault). This observation is also confirmed in our evaluation (§6.2). Based on this observation, we propose an input generation technique guided by *sensitivity* (Sensitivity Maximizing Fuzzer in §3).

2.3 Computing test coverage automatically

Test coverage metric measures the proportion of fault-inducing data flows explored. Ideally, when test coverage metric reaches 100%, all fault-inducing data flows of a DLS are explored [16, 26, 45]. However, since the actual number of fault-inducing data flows in a DLS is unknown, proposing an accurate test coverage metric is an open challenge [16, 26, 45]. Existing testing techniques [7, 11, 21, 28, 36] mitigate this challenge by requiring developers to specify a set of fault-inducing data flows. However, recent studies show that such manual effort is tedious and error-prone (§2.1).

To solve this challenge, Themis leverages the sensitivity of a DLS's data flow to infer the test coverage. We summarize the existing theories to show that for any DLS, $N_i(I+E(\theta))-N_i(I)$ follows a normal distribution. First, existing theories show that DNN is an ordinary differential equation [7, 27], and DNN's neurons are differential operators of an ordinary differential equation. Second, when random noise is present on an ordinary differential equation's input, the variation on the output values of the ordinary differential equation's differential operators often follows a normal distribution [39, 46].

By combining the first and the second theories, we can derive that when random noise is present on a DLS's input, the variation of the DLS's activation values (i.e., $N_i(I + E(\theta)) - N_i(I)$ defined in §2.1) often follows a normal distribution (we denote ND_i as the normal distribution, and we call $\hat{ND_i}$ as "sensitivity distribution" in Figure 2). We also confirmed these observations in our evaluation (Figure 4b).

Since $\hat{ND_i}$ consists of the frequency of all $N_i(I+E(\theta))-N_i(I)$ values including the fault-inducing data flows $\{N_i(I + E(\theta))\}$, the condition that $\hat{ND_i}$ being identified as ND_i implies all fault-inducing data flows are also identified (we carry a more detailed analysis in §4.2). Based on this observation, we proposed a test coverage metric based on the distribution of ND_i (Sensitivity Convergence Coverage in §3).

3 Overview

This section presents the architecture and workflow of Themis. Figure 2 shows Themis's major components (namely Sensitivity Calculator, Sensitivity Convergence Coverage and Sensitivity Maximizing Fuzzer) and these components' workflow. To ease discussion, this section uses the notations defined in §2.1: given a DLS (M), a set of inputs (I) for the DLS to perform inference and environmental noise to be applied on I ($E(\theta)$), Themis aims to detect the $M(I + E(\theta))$ which is not equal to $M(I)$ (such an $M(I + E(\theta))$ is a fault).

To do so, Themis first perturbs each input of I with environmental noise $E(\theta)$ (for each input in I, θ is a randomly chosen value within a given range). Then, Themis feeds both I and $I + E(\theta)$ to **Sensitivity Calculator (Phase 1)**, to compute the difference between the outputs of each M's neurons with respect to these inputs (i.e., $N_i(I+E(\theta))-N_i(I)$ defined in §2.1, which are the ith neuron's outputs with respect to I and $I + E(\theta)$ respectively). Sensitivity Calculator also computes $M(I)$ and $M(I + E(\theta))$, and reports faults to DLS developers (i.e., $M(I + E(\theta))$ which is not equal to $M(I)$).

Note that Themis needs to compute both $N_i(I)$ and $N_i(I + E(\theta))$, while existing DLS testing techniques only compute $N_i(I + E(\theta))$, even though all these techniques aim to trigger fault-inducing $N_i(I)$. It is because Themis leverages the theoretical implication behind $N_i(I+E(\theta)) - N_i(I)$ to guide the detection of fault-inducing $N_i(I + E(\theta))$, in order to automatically identify fault-inducing data flows

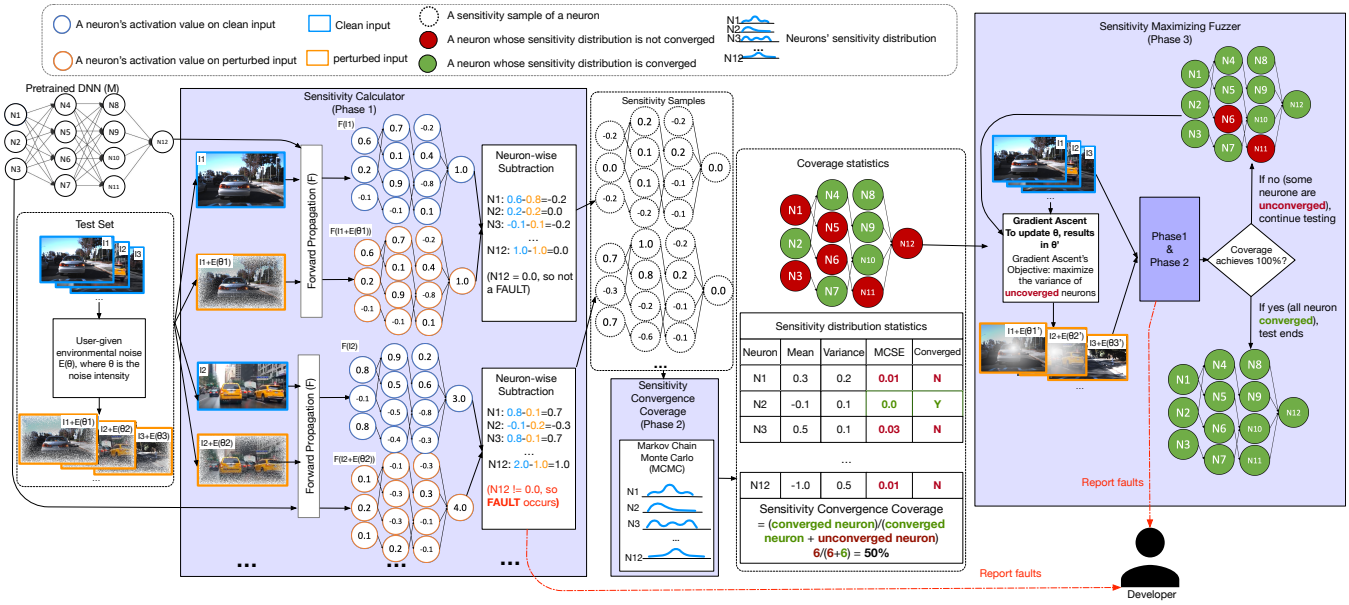


Figure 2: Themis's architecture. Themis's components are shaded in purple. The DNN's neurons (e.g., N2) are colored in green if their MCSE value (as shown in the "Sensitivity distribution statistics") equals 0.0 (i.e., sensitivity distribution of the neurons converged).

(see §2.2). On the other hand, existing techniques require fault-inducing $N_i(I+E(\theta))$ to be specified by DLS developers, which has been proven tedious and error-prone (§2.3).

Sensitivity Calculator then passes $N_i(I+E(\theta)) - N_i(I)$ values to **Sensitivity Convergence Coverage (Phase 2)**, which infers a distribution from the values (i.e., $\hat{ND}_i$ defined in §2.3) for each neuron (i.e., each $i \in n$). Sensitivity Convergence Coverage then determines whether $\hat{ND}_i$ converges to a normal distribution and returns a test coverage value accordingly. Specifically, for each neuron in a DLS, Sensitivity Convergence Coverage feeds the $N_i(I+E(\theta)) - N_i(I)$ values to Monte Carlo Markov Chain [12] (MCMC, a popular statistical technique for inferring probability distributions from a set of numerical values).

MCMC then outputs both $\hat{ND}_i$ (the distribution inferred by MCMC) and the Monte Carlo Standard Error (MCSE, a metric to measure the potential error between the inferred distribution and the ground-truth normal distribution) value corresponding to $\hat{ND}_i$. Sensitivity Convergence Coverage thus determines whether $\hat{ND}_i$ converges to a normal distribution based on the MCSE value: If MCSE equals zero within a confidence interval α (Themis's default value of α is 95%, a standard value of setting a confidence interval [3, 18]), then Sensitivity Convergence Coverage considers $\hat{ND}_i$ converges to a normal distribution at a high probability (i.e., 95%). Otherwise, $\hat{ND}_i$ is considered not converged.

Sensitivity Convergence Coverage then computes the test coverage metric as the proportion of neurons whose $\hat{ND}_i$ converges. For instance, in Figure 2, after Sensitivity Convergence Coverage computes $\hat{ND}_i$ for the twelve neurons, six neurons among the twelve neurons (i.e.,N2, N4, N7, N8, N9 and N10 which are colored in green) have MCSE as 0.0 (i.e., their $\hat{ND}_i$ are converged), while the rest of the six neurons of the DLS have MCSE large than 0.0 (i.e., their $\hat{ND}_i$ are not converged). Hence, Sensitivity Convergence Coverage computes the test coverage as 50%.

Sensitivity Convergence Coverage addresses an open challenge of DLS testing: accurately inferring the proportion of faults undetected from a DLS (i.e., computing a test coverage metric), even the total number of the DLS's faults is unknown. Specifically, Sensitivity Convergence Coverage does so by inspecting $\hat{ND}_i$'s convergence condition, which is correlated with the proportion of the DLS's faults being detected (§2.3). Our theoretical analysis showed that when Sensitivity Convergence Coverage reaches 100%, all fault-inducing data flows are identified (i.e., explored) at high probability (95%). In contrast, existing works mitigate this challenge by requiring DLS developers to specify a set of fault inducing data flows, such that the test coverage metric is computed as the specified fault inducing data flows being explored.

After that, Sensitivity Convergence Coverage passes the neurons whose $\hat{ND}_i$ is converged to Themis's **Sensitivity Maximizing Fuzzer (Phase 3)**. Sensitivity Maximizing Fuzzer then leverages math optimization techniques (e.g., Gradient Ascent) to generate a new test set, to explore unexplored fault-inducing $N_i(I+E(\theta))$. Specifically, Sensitivity Maximizing Fuzzer iteratively adjusts θ of E (we denote the adjusted θ as θ'), such that $I+E(\theta')$ maximizes the sum of $N_i(I+E(\theta)) - N_i(I))$ for all neurons whose $\hat{ND}_i$ are not converged (i.e., N2, N4, N7, N8, N9 and N10 in Figure 2). Sensitivity Maximizing Fuzzer does so to ensure that the generated test set can explore new fault-inducing data flows (i.e., $N_i(I+E(\theta))$ which has a large difference with $N_i(I)$)). Sensitivity Maximizing Fuzzer then feeds the generated $I+E(\theta')$ to Sensitivity Calculator for computing $N_i(I+E(\theta')) - N_i(I))$, and thus Themis enters another iteration of the testing.

To understand Themis's workflow from the perspective of fuzzing,

since Themis 's coverage metric is computed based on the neuron's divergence, Themis performs fuzzing to maximize the coverage metric (i.e., neuron's divergence), same as the conventional data-flow-oriented and coverage-guided fuzzing techniques [25, 31, 32].

Themis does not choose the divergence of a DLS's output as the coverage metric, because the divergence of a DLS's output does not have the important internal features of neurons' divergence. In contrast, given a data flow DF produced by a raw image I and a perturbed data flow DF' produced by a corresponding perturbed input (I'), neuron's divergence can reveal at which internal data flow points I and I' diverge, then Themis automatically generates subtle perturbed noise for further aggravating the divergence (**Phase 2**), leading to many more detected faults than baselines (§6.2) and attaining strong fault detection capability.

In sum, at a conceptual level, neuron's outputs consist of "all data flow branches" of a DLS (§2.1), while DLS's outputs consist of "the last data flow branches" of the DLS, because DLS's outputs can be viewed as mainly the outputs of neurons in the DLS's last layer (i.e., DLS' outputs only represent a minor subset of all data flows of the DLS). Hence, in order to explore all fault-inducing data flows at high probability (a prerequisite to attain strong fault detection capability), maximizing neuron's divergence (i.e., exploring all data flow branches of a DLS) is required by Themis, rather than maximizing the divergence of a DLS's outputs (i.e., exploring a minor subset of all data flow branches of the DLS).

4 Themis's runtime

4.1 The challenge in designing Themis's workflow

One major novelty of Themis is a new workflow to compute test coverage metric, by computing $N\hat{D}_i$ for all of a DLS's neurons (§2.1 and §3). However, realising such a workflow is challenging, because $N\hat{D}_i$ for all of a DLS's neurons may cause Themis unscalable to a DLS's size.

Computing $N\hat{D}_i$ for just one neuron of a DLS is already computationally expensive, because it requires Themis's Sensitivity Convergence Coverage to run MCMC on thousands of sensitivity samples (§3). Hence, for DLSs of large sizes (e.g., ResNet56 which consists of more than five hundred thousand neurons, see Table 1), the strawman approach of brutal force computing would require more than hours to conduct the testing, inefficient compared to the existing techniques [11, 21, 28, 36], which only takes several minutes to complete testing.

On the other hand, computing $N\hat{D}_i$ for only a subset of a DLS's neurons may cause Themis to miss substantial faults. Because $N\hat{D}_i$ of different neurons have different convergence rate ($N\hat{D}_i$'s convergence rate of a DLS neuron is essential for Themis to compute test coverage, see §3), computing $N\hat{D}_i$ for a subset of a DLS's neurons may cause Themis to compute test coverage incorrectly (e.g., stop testing too soon or too late).

To tackle this challenge, we present Sensitivity Sampler, a sampling technique used by Themis's Sensitivity Convergence Coverage to compute $N\hat{D}_i$ for a subset of a DLS's neurons, in order to precisely approximate the $N\hat{D}_i$'s convergence rate of all the DLS's neurons. Our observation behind Sensitivity Sampler is that $N\hat{D}_i$'s convergence rate is inversely proportional to $N\hat{D}_i$'s variance (the larger the variance, the slower the convergence rate, as pointed out by a classic statistic theory called Chebyshev's inequality [33]). Hence, for a DLS of any size, Themis samples a constant number (by default one thousand, see §4.2) of neurons for computing $N\hat{D}_i$ based on the variance of $N_i(I+E(\theta))-N_i(I))$ values of each neuron.

Algorithm 1: Themis's entire workflow

Input: $\{I\}$: raw inputs, M: DNN to be tested, n: the number of neurons of M, $E(\theta)$: User-specified perturbation, t: threshold value for MCMC to determine convergence, k: sample size for Sensitivity Sampler, c: desired sensitivity coverage

1 **Function** Themis ($\{I\}$,$E(\theta)$,M,n):
2 Initialise List $sensitivitySample$; Initialise List $Faults$
3 **while** True **do**
   /* Sensitivity Calculator (Phase 1) begins */
4 **for** each $I \in \{I\}$ **do**
5 $\{N_i(I)\}_{i=1}^{n} \leftarrow M(\{I\})$'s activtion values
6 $\{N_i(I+E(\theta))\}_{i=1}^{n} \leftarrow M(I+E(\theta))$'s activtion values
7 Initialise Array $\{Diff_i\}_{i=1}^{n}$
8 **for** each $j \in$ [1,...,n] **do**
9 $Diff_j = |N_j(I+E(\theta)) - N_j(I)|$
10 $sensitivitySample$.insert($Diff$)
11 **if** $M(I) \neq M(I+E(\theta))$ **then**
12 $Faults$.insert(I,$E(\theta)$)
   /* Sensitivity Convergence Coverage (Phase 2) begins */
13 $\{Diff_i\}_{i=1}^{k} \leftarrow$ sensitivitySampler($sensitivitySample$,k)
14 $\{N\hat{D}_i\}_{i=1}^{k} \leftarrow$ MCMC($\{Diff_i\}_{i=1}^{k}$)
15 sensitivityCoverage $\leftarrow$ proportion of $\{N\hat{D}_i\}$ that has MCSE == 0.0
   /* Sensitivity Maximizing Fuzzer (Phase 3) begins */
16 **if** sensitivityCoverage $< c$ **then**
17 $UncovNeuron \leftarrow$ unconverged neuron
18 obj = maximizer($UncovNeuron$)
19 $I + E(\theta') \leftarrow$ gradientAscent(obj,$I + E(\theta)$)
20 **else**
21 break
22 **return**
23 **Function** sensitivitySampler($sensitivitySample$,k):
24 Initialise Array $\{variance_i\}_{i=1}^{n}$
25 Initialise Array $\{\text{CoE}_i\}_{i=1}^{n}$
26 **for** each $j \in$ [1,...,n]) **do**
27 $Variance_j$ = compute_variance($sensitivitySample[j]$)
28 D_{sort} = sort $\{Variance_i\}_{i=1}^{n}$ based on variance
29 **for** each $j \in$ [1,...,k]) **do**
30 $selected_j = variance_{j*\frac{|D_i|}{k}}$
31 **return** $\{selected_i\}_{i=1}^{k}$

4.2 Themis's algorithm

Algorithm 1 shows the algorithm of Themis, including Themis's three components (§3): Sensitivity Calculator (Phase 1), Sensitivity Convergence Coverage (Phase 2) and Sensitivity Maximizing Fuzzer (Phase 3), as well as Sensitivity Sampler (§4.1).

When M, $\{I\}$, and $E(\theta)$ are fed into Themis, Themis starts testing process. Specifically, for I in $\{I\}$, Themis's Sensitivity Calculator (line 2-12) first computes $N_i(I)$ and $N_i(I+E(\theta))$ (line 5-6), which are defined in §2.2. Sensitivity Calculator then computes the difference

Dataset **(I)**	Description	DNN **(M)**	Number of neurons	Accuracy of pretrained model**M(I)** (%)
MNIST	Hand-written digit classification	LeNet-1 [10, 24]	7206	98.3
		LeNet-4 [10, 24]	69362	98.5
		LeNet-5 [10, 24]	107786	98.9
Contagio/virustotal	Malware classification in PDF Files	<200,200> [36]	35410	96.1
Drebin	Malware classification in Android apps	<200,10> [14, 36]	15230	98.6
ImageNet	General Image classification	VGG-19 [41]	14888	92.6
		ResNet-50 [17]	16168	96.4
Udacity	Road condition classification	DAVE-2 [47]	1560	99.1
Cifar10	General Image classification	ResNet56 [17]	532490	90.7
		DenseNet121 [19]	563210	85.6

Table 1: Datasets and DNNs for evaluating Themis, which covers the complete set of datasets evaluated by baselines.

(denoted as D_i) between $N_i(I)$ and $N_i(I + E(\theta))$ (line 8- 10). D_i are *sensitivity samples* (§3). Meanwhile, Sensitivity Calculator reports faults to the developer for any $M(I)$ and $M(I+E(\theta))$ with different DLS's outputs.

Then, Themis's Sensitivity Convergence Coverage infers $N\hat{D}_i$ of the DLS's neurons based on D_i (line 2- 15). Specifically, Sensitivity Convergence Coverage uses Sensitivity Sampler (line 23- line 31) to sample n (by default one thousand) neurons from all the DLS's neurons as CoE (Sensitivity Sampler's rationale is explained in §4.1).

Specifically, Sensitivity Sampler computes the variance of each item in D_i, and sort these items according to their variance. Then Sensitivity Sampler selects k (by default one thousand) items from all the items in an evenly-spaced manner (i.e., select the item for every $\frac{|D_i|}{k}$) items). The default value of k (i.e., one thousand) is selected based on Mann-Witney statistic [5], which points out that one thousand samples from a sorted list is sufficient to accurately approximate all the remaining values in the sorted list.

Hence, with the k samples selected by Sensitivity Sampler, Sensitivity Convergence Coverage is able to approximate the $N\hat{D}_i$'s convergence rate of all neurons (§3), based on the convergence rate of the k sampled neurons (§4.1). Our evaluation (Figure 6a) also shows that setting k as one thousand allows Sensitivity Convergence Coverage to precisely approximate $N\hat{D}_i$'s convergence rate of all neurons.

After Sensitivity Sampler computes $N\hat{D}_i$ for each sampled neurons (line 14), Sensitivity Sampler computes *sensitivity convergence* (the test coverage of Themis) as the number of converged $N\hat{D}_i$ over k (line 15). Specifically, the convergence criterion is whether Monte Carlo Standard Error (MCSE) equals 0.0, a standard criterion which indicates the probability of convergence identified by MCMC [12, 38]. By having MCSE equal 0.0, each batch of samples drawn by MCMC have almost identical probability distributions. We also evaluated the choice of MCSE value and our evaluation result (Figure 6b) shows that MCSE equal 0.0 allowed Sensitivity Sampler to precisely infer $N\hat{D}_i$'s convergence rate of all neurons. After Sensitivity Sampler computed *sensitivity convergence*, Themis's Sensitivity Maximizing Fuzzer (line 18- line 19) performs fuzzing in order to explore the unconverged $N\hat{D}_i$ (i.e., coverage-guided fuzzing).

Analysis of Themis's fault detection capability. Themis has strong fault detection capability because Themis theoretically can explore all fault-inducing data flows at high probability (95%). We first explain how we derive this theoretical property of Themis. As discussed in §2.3, we summarize two existing theories [7, 27, 39, 46] and conclude that when random noise is present on a DLS's input, the variation of a DLS's activation values (i.e., $N_i(I+E) - N_i(I)$ defined in §2.1) often follows a normal distribution (i.e., ND_i defined in §2.1). Hence, Themis examines whether the generated test set explores a full coverage of the fault-inducing data flows (i.e., $N_i(I+E)$ which has large difference with $N_i(I)$) by examining whether a normal distribution can be inferred from the $N_i(I+E) - N_i(I)$ values associated to the test set (i.e., whether the test set explores a full coverage of $N_i(I+E) - N_i(I)$ values).

Note that even when distribution inferred from the $N_i(I+E) - N_i(I)$ values (i.e., $N\hat{D}_i(I+E)$) converges to a normal distribution, Themis is not 100% guaranteed to identify all fault-inducing data flows. It is because the $N_i(I+E) - N_i(I)$ values can coincidentally form a normal distribution different from ND_i. Hence, Themis probabilistically (rather than deterministically 100%) identifies a full coverage of fault-inducing data flows, and the corresponding probability depends on the confidence interval (by default 95%) adopted by Themis when inferring ND_i (§3).

After we justify the reason why Themis theoretically can explore all fault-inducing data flows at high probability (95%), we show that this theoretical property of Themis allows Themis to have strong fault detection capability. As discussed in §4.2, for any DLS having more number of faults (i.e., has lower accuracy), the DLS's $N\hat{D}_i$ often has greater variance. This implies that these DLSs require Themis to generate more $I+E(\theta)$ to identify the DLS's $N\hat{D}_i$ (any statistical distribution with greater variance requires more samples to identify the distribution, according to Chebyshev's inequality [33]). Since with more $I+E(\theta)$, Themis can detect more faults (the number of $I+E(\theta)$ generated by Themis is the upper limit of the number of faults can be detected by Themis). Hence, Themis detects more faults from a DLS which has lower accuracy (i.e., Themis has strong fault detection capability).

5 Implementation

We implemented Themis using PyTorch 1.8.1 [35]. Themis's implementation consists of around 9,791 lines of Python code. We adopted PYMC3 [40], a popular Python package which realises MCMC, to perform MCMC in Themis. Specifically, Themis's Sensitivity Calculator (§3) calls PYMC3.DATA to load the samples of sensitivity, and constructs the prior distributions by calling PYMC3.NORMAL. Then, Themis's Sensitivity Convergence Coverage calls PYMC3.SAMPLE to perform MCMC. Sensitivity Convergence Coverage examines the convergence rate (i.e., MCSE, §4) by ARVIZ.SUMMARY [22], a popular Python package for analysis of Bayesian models. Finally, Themis's Sensitivity Maximizing Fuzzer is realised with PyTorch's AUTOGRAD [29] library, which efficiently performs gradient ascent on $\hat{ND}_i$(§4).

6 Evaluation

6.1 Experiment Setup

Our evaluation was done on a computer equipped with twenty CPU cores and four Nvidia RTX2080TI graphic cards. We compared Themis with state-of-the-art DLS testing techniques: DeepeXplore[36], DeepGauge[28], DeepImportance[11], and Surprise Adequacy[21]. For the baselines which are merely adequacy criteria but not a test generator (i.e., Deepguage, Surprise Adequacy, and DeepImportance), we applied the test generation method from notable existing studies (e.g., DeepXplore [36] and DeepTest [43]) to these baselines. Specifically, we integrated a backpropagation technique into these baselines, so that these baselines can generate test inputs by perturbing raw inputs with gradient descent and the generated inputs can maximize the test criteria proposed by these baselines.

Table 1 shows our evaluated datasets and models, which covers a complete set of six datasets evaluated by four baselines. To be fair and comprehensive, our evaluated models covered **all** models evaluated by four notable DL testing works [11, 21, 28, 36] published in the last three years. The models we evaluated include VGG [41], ResNet [17], DenseNet [19], etc. We believe the architecture of these models covers all basic building blocks (such as convolution layers and residual connections) of most modern real-world deployed AI applications. These DNN models all achieve high precision. Hence, finding faults (i.e., a DNN's incorrect outputs) from these well-studied and well-tested DNN will be valuable.

We choose the correlation metric [16, 26] (also defined in §1) as the main evaluation metric to compare the fault detection capability between Themis and the baselines: among the N faults reported by a testing technique, whether each fault reported can trigger an actual fault for the tested DLS (if so, correlation is added by $1/N$; otherwise, it is decreased by $1/N$). The value of the correlation metric is between -1.0 and 1.0 and a latest study [16] points out that a high correlation (e.g., 0.7) implies strong fault detection capability. We also evaluated the increased accuracy of DLS after retraining the DLS with the detected faults, to evaluate the quality of faults detected by a DLS testing technique. Finally, we evaluated the testing time cost for Themis and baselines to attain 100% test coverage.

To study Themis and the baselines' fault detection capability, we applied four well-studied perturbations (CW [6], FGSM [13], PGD [30], and Gaussian noise) to each dataset of the evaluated DNN models in Table 1. These perturbations are evaluated by Themis's baselines [11, 28] because these perturbations are common in real world DLS applications. Hence, faults detected from a DLS under these perturbations greatly promote the reliability of a DLS.

The values of the magnitude of these perturbations in our evaluation are listed in Table 3. We follow Themis's baselines to set these values [11, 21, 28, 36].

The evaluation questions are as follows:

§6.2: How is Themis's fault detection capability compared to the baselines?
§6.3: How is the improvement on a DLS's accuracy attained by Themis?
§6.4: How is Themis's efficiency compared to baselines?
§6.5: What are the factors that affect Themis's fault detection capability?
§6.6: What are Themis's limitations?

6.2 Themis's Fault Detection Capability Result

We first investigate the fault detection capability of Themis and baselines. Specifically, we perturbed raw inputs (e.g., road condition images) with various noise intensities as shown in Table 3. Then, for each set of inputs perturbed by the same noise intensity (e.g., PGD with epsilon as 0.1), we fed these perturbed inputs into a DLS and computed the DLS's error rate (i.e., the proportion of the perturbed inputs being wrongly classified by the DLS) and the number of faults detected by Themis until Themis 's coverage metric reached 100%. Finally, the fault detection capability is computed as the correlation between all pairs of a DLS's error rate and the number of faults detected (e.g., the pairs of DLS's error rate and the number of faults detected corresponding to PGD's epsilon as 0.1,0.2,...0.5 as shown in Table 3). Then, the inference accuracy of these DNNs and the number of faults detected by Themis or the baselines are used to compute the fault detection capability of Themis and the baselines.

Table 2 shows the fault detection capability of Themis and the baselines on all datasets. Themis's correlation was larger than 0.7 for all the evaluated datasets, while baselines rarely achieved 0.7 correlation or even had negative correlation (i.e., detected fewer faults for DNN which has lower accuracy). It is because the baselines tested a DLS deterministically, while Themis tested a DLS probabilistically (§2.2).

We further inspect why Themis achieved strong fault detection capability in Figure 4b. As discussed in §4.2, the condition for Themis to achieve strong fault detection capability is that Themis precisely computes $\hat{ND}_i$, which guides Themis to attain full fault coverage. Hence, we inspect whether Themis precisely computes $\hat{ND}_i$, by inspecting the consistency of $\hat{ND}_i$'s means computed by Themis: Themis is supposed to compute the same $\hat{ND}_i$'s mean for the same DLS on different sets of test inputs because theoretically, there is only one $\hat{ND}_i$ for each DLS's neuron for the same set of test inputs (§2.2).

Figure 4b shows that Themis computed $\hat{ND}_i$'s mean with less than 10% deviation, so according to conventional standards in statistics [15], Themis successfully computed $\hat{ND}_i$. Themis can do this because theoretically, $\hat{ND}_i$ could be identified by Bayesian analysis, and Themis's MCMC component could identify it. Figure 5b also confirms our observation that greater sensitivity ($N_i(I + E(\theta)) - N_i(I)$) values resulted in higher number of faults.

Dataset(DNN)	CW					FGSM				
	Themis	Deep-Xplore	Deep-Gauge	Deep-Import.	Surprise Adequacy	Themis	Deep-Xplore	Deep-Gauge	Deep-Import.	Surprise Adequacy
MNIST (LeNet-1)	0.79	-0.58	0.25	0.08	-0.21	0.77	0.21	0.35	-0.35	0.33
MNIST (LeNet-4)	0.72	-0.88	0.01	-0.62	-0.5	0.74	0.26	0.34	0.24	-0.18
MNIST (LeNet-5)	0.71	0.35	-0.41	-0.17	0.74	0.83	-0.02	-0.02	0.18	-0.05
Contagio (<200,200>)	0.71	0.33	-0.07	0.25	-0.87	0.77	0.71	-0.04	0.1	0.38
Drebin (<200, 10>)	0.70	0.11	0.18	0.78	-0.37	0.81	-0.06	0.08	-0.19	0.2
ImageNet (VGG-19)	0.79	-0.77	-0.07	0.12	-0.29	0.86	0.3	-0.62	-0.01	0.74
ImageNet (ResNet-50)	0.71	0.4	0.33	-0.06	-0.09	0.74	0.3	0.1	-0.05	0.1
Udacity (DAVE-2)	0.72	-0.16	0.31	0.76	0.21	0.84	0.77	0.4	0	-0.08
Cifar10 (ResNet56)	0.86	0.35	-0.36	0.4	0.21	0.85	0.37	0.01	-0.63	0.08
Cifar10 (DenseNet121)	0.88	0.25	-0.7	0.05	-0.01	0.78	-0.51	0.72	0.33	0.39
Dataset(DNN)	PGD					GAUSSIAN				
	Themis	Deep-Xplore	Deep-Gauge	Deep-Import.	Surprise Adequacy	Themis	Deep-Xplore	Deep-Gauge	Deep-Import.	Surprise Adequacy
MNIST (LeNet-1)	0.72	-0.89	0.01	0.39	0.23	0.7	0.2	-0.21	-0.06	0.71
MNIST (LeNet-4)	0.77	-0.03	-0.54	0.37	0.19	0.72	0.18	-0.07	-0.6	-0.68
MNIST (LeNet-5)	0.71	0.01	-0.04	-0.23	0.76	0.89	-0.17	-0.09	0.75	0
Contagio (<200,200>)	0.78	-0.13	0.3	0.23	-0.09	0.77	-0.63	0.18	0.37	0.28
Drebin (<200, 10>)	0.71	0.4	-0.02	-0.1	-0.57	0.82	0.26	-0.32	0.36	0.11
ImageNet (VGG-19)	0.75	0.09	0.1	-0.13	-0.19	0.84	0.79	0.28	0.1	-0.03
ImageNet (ResNet-50)	0.78	-0.09	-0.56	0.02	-0.05	0.86	-0.16	-0.17	-0.09	-0.17
Udacity (DAVE-2)	0.71	0.31	0.07	-0.71	-0.74	0.8	0.4	0.34	-0.54	0.08
Cifar10 (ResNet56)	0.8	-0.1	0.09	-0.08	0.36	0.95	-0.57	0.11	0.06	0.35
Cifar10 (DenseNet121)	0.86	-0.04	0.25	-0.56	-0.73	0.91	0.22	-0.14	0.31	0.4

Table 2: Correlation between the number of faults identified by DLS testing and the error rate of a DLS. Correlation larger than 0.7 (colored in green) is considered strong [16]. Correlation less than 0.0 is colored in red."Deep-Import." means "Deep-Importance".

Variables (Perturbations)	Magnitude Value				
c Confidence (CW [6])	10	20	30	40	50
ϵ (FGSM [13])	0.1	0.2	0.3	0.4	0.5
ϵ (PGD [30])	0.1	0.2	0.3	0.4	0.5
σ (Guassian noise)	0.01	0.02	0.03	0.04	0.05

Table 3: Perturbations adopted in our evaluation.

6.3 Increase in DNNs' accuracy obtained by Themis

One main purpose of DLS testing is to retrain a DLS with detected faults to increase the DLS's accuracy. Figure 3b shows the increase in DLS's accuracy brought by Themis and the baselines. Overall, Themis increased the DLS's accuracy higher than baselines for all datasets. The main reason is that Themis identified more faults than the baselines (Figure 4a). Specifically, in our evaluation, we let both Themis and the baselines keep generating inputs until their coverage metric reached 100%. By doing so, Themis on average generated 21399 test inputs, DeepXplore [36] generated 3774 test inputs, DeepGuage [28] generated 18501 test inputs, DeepImportance [11] generated 20052 test inputs, and Surprise Adequacy [21] generated 26757 test inputs. Overall, the test inputs generated by Themis consisted of much more faults than the baselines.

We found that Themis increased the accuracy of Cifar10 (densenet121) under PGD perturbation the most (i.e., 11.56%). It is because densenet121 is vulnerable to PGD perturbation [37], so it suffered from the greatest accuracy loss under this perturbation. Since Themis detected more faults from DLS which had lower accuracy (Table 2), Themis detected more faults for this DNN than other DNNs, which allowed Themis to increase the DNN's accuracy more than Themis did on the other DNNs. This also implies Themis is valuable to real-world DLSs, which are often trained with limited datasets and have moderate accuracy (especially on safety-critical tasks, see §1).

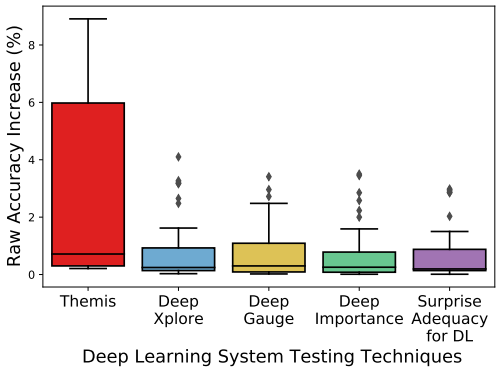
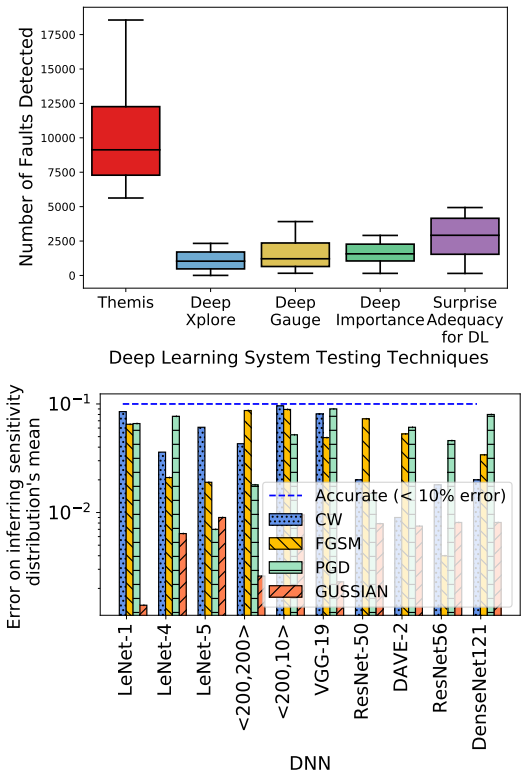


Figure 3: Increase in a DLS's accuracy by retraining the DLS with 100 faults detected by Themis and the baselines.

Dataset(DNN)	with Sensitivity Sampler				without Sensitivity Sampler			
	Total Test Time	Sensitivity Calculator	Sensitivity Coverage	Sensitivity Fuzzer (iterations)	Total Test Time	Sensitivity Calculator	Calculator Estimator	Sensitivity Fuzzer (iterations)
MNIST (LeNet-1)	160	27	24	109 (5)	11171	24	2006	9141 (5)
MNIST (LeNet-4)	143	22	11	110 (10)	2434	23	217	2194 (11)
MNIST (LeNet-5)	208	59	34	115 (4)	1704	56	379	1269 (4)
Contagio (<200,200>)	201	49	9	143 (15)	14320	50	856	13414 (16)
Drebin (<200,10>)	211	27	17	167 (10)	12690	27	1139	11524 (12)
ImageNet (VGG-19)	202	49	23	130 (6)	10273	50	1533	8690 (6)
ImageNet (ResNet-50)	131	57	2	72 (32)	5470	51	162	5257 (32)
Udacity (DAVE-2)	202	33	7	162 (24)	2055	33	80	1942 (25)
Cifar10 (ResNet56)	241	39	24	178 (7)	18036	39	2159	15838 (7)
Cifar10 (DenseNet121)	155	22	23	110 (5)	1613	21	270	1322 (6)

Table 4: Themis's testing time breakdown (all the numbers are in the unit of seconds). Those total testing time costs comparable to the baselines are colored in green.



(b)

Figure 4: (a) Number of faults detected by Themis and the baselines when their test coverage metric reaches 100%. (b) The mean of $N\hat{D}_i$ inferred by Themis for each evaluated DNN. Themis is considered accurate in inferring the mean if the variation of the mean is below 10% [15] .

6.4 Themis's testing time cost

We then study the efficiency of Themis. Figure 5a shows the testing time cost of Themis and the baselines. Themis on average had 27.1% more testing time cost than baselines. It is because Themis was probabilistic and hence required more computation than deterministic approaches (see 4.1). Nevertheless, Themis identified 3.78X more faults than baselines, so the testing time cost is worthwhile.

To identify the source of Themis's time cost, we break down Themis's time cost, which is comprised of three main components: Sensitivity Calculator, Sensitivity Convergence Coverage, and Sensitivity Maximizing Fuzzer (3). Table 4 shows Themis's time cost in Sensitivity Calculator, Sensitivity Convergence Coverage and Sensitivity Maximizing Fuzzer with or without Themis's Sensitivity Sampler. From the table, we can see that Themis's performance overhead was mainly from Sensitivity Maximizing Fuzzer, which iteratively performs MCMC (known to be time-consuming). Without Themis's sampler component, Themis's time on Sensitivity Maximizing Fuzzer was enormous (more than one hour), because Themis had to perform MCMC on all DNN's neurons. Fortunately, with Themis's sampler, Themis could accurately compute the coverage metric based on the results of one thousand DNN neurons (§4.1).

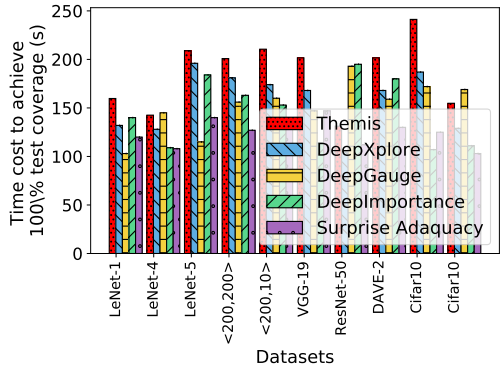
6.5 Sensitivity studies on Themis's parameters

We study the sensitivity of Themis's parameters (i.e., threshold t and Sensitivity Sampler's sampler size). Figure 6 shows Themis's coverage variation on these values. Specifically, Figure 6a shows that Themis's coverage was the same as the ground truth when the sample size equaled one thousand, which was coherent with Mann-Witney Test theory [5] (§4.2).

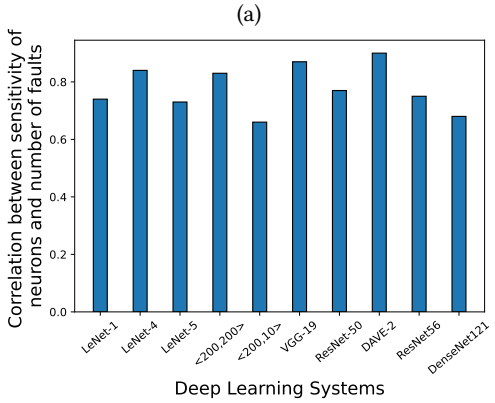
We could also observe that when the sample size increased, Themis's coverage rate was closer to the ground-truth. It is because statistically, with more samples, we could approximate the ground-truth distribution better. Nevertheless, the increase in precision is diminishing when the sample size grows larger. Hence, setting the sample size as one thousand is desirable, as justified theoretically (§4.2) and empirically (Figure 6a). Figure 6b shows the variation of Themis's coverage against a threshold value (t). The figure shows that setting t as zero (i.e., the default value) allowed Themis to achieve the ground-truth coverage value.

6.6 Limitation and future work

Themis has four main limitations. First, Themis requires extra testing time (20%) than the baselines, because Themis probabilistically tests a DNN (§6.4). Nevertheless, even though Themis has the additional testing time, Themis still completes the testing within several minutes, which is comparable with related work [11, 21, 28, 36]. Besides, Themis detected 3.78X more faults and enhanced the accu-



(a)



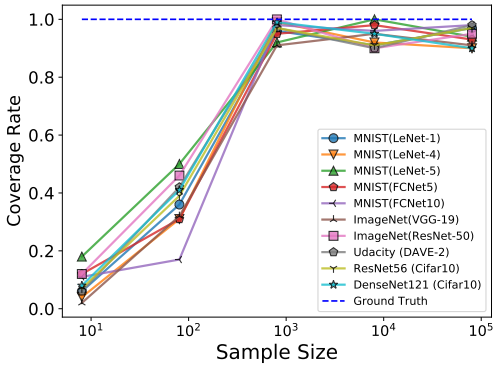
(b)

Figure 5: (a) Average time taken by DLS testing to complete testing (i.e., achieve 100% test coverage), (b) the correlation between sensitivity of neurons and the error rate of DLSs.

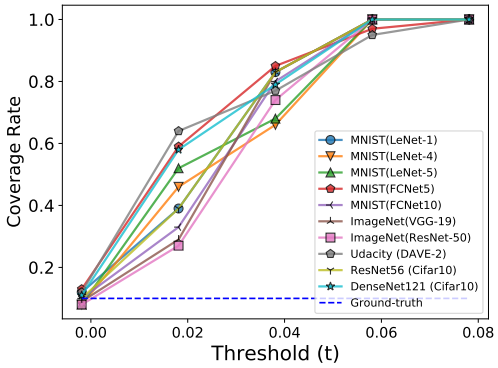
racy of DNN on average 14.7X times more than baselines. Hence, Themis's additional testing time cost is worthwhile.

The second limitation is that although Themis increased a DLS's accuracy by retraining the DLS with faults detected by Themis, Themis does not guarantee these faults would be eliminated from the DLS after retraining. Due to the randomness nature of sampling techniques, there is a significant probability that sensitivity samples converge to a normal distribution while a tiny portion of fault-inducing data flows is not covered (§4.2): the sensitivity samples can coincidentally converge to a normal distribution different from the ground-truth; hence, Themis probabilistically (95%, rather than deterministically 100%) covers the fault-inducing data flows in a DLS. Indeed, how a DLS's faults can be eliminated with a guarantee is still an open challenge for all DLS testing techniques [16].

The third limitation is that the noise produced by Themis cannot be transferred to other images. Same as all the baselines [11, 21, 28, 36], each type of environmental effect (e.g., fog) tested by Themis is specific to the tested images, because intuitively other non-tested images may require largely different strength of the same type of effects to induce faults. After all, given a combination of each set of images to be tested and each type of environmental effects, Themis is fully automatic and just runs the combination on a DLS, while existing DLS techniques may require intense manual effort on inferring a new set of activation values before each run.



(a)



(b)

Figure 6: (a) Themis's coverage values for different sample sizes. (b) Themis's coverage values for different threshold values.

The fourth limitation is that Themis can't test large models with hundreds of millions of parameters (e.g., Swin Transformer, V-MoE). Because themis samples the 1000 neurons with the largest perturbation value from the model for testing. When the model's parameters reach hundreds of millions, 1000 neurons can't reflect all the features learned by the model. (Different neurons in the DNN correspond to different features, such as colors, stripes, etc.)

Nevertheless, Themis is the first DLS testing technique that leverages the theoretical properties of a DLS's faults (i.e., $\hat{ND}_i$) to guide the detection of these faults. While empirically detecting faults based on $\hat{ND}_i$ allows Themis to have full fault coverage with high probability, we will leave its theoretical analysis as future work.

7 Conclusion

This paper presents Themis, the first automated and efficient DLS testing technique which can explore a full coverage of fault-inducing data flows, and thus the fault detection capability is empirically strong (higher than 0.7). Our evaluation results show that Themis has great potential to detect substantial faults from real-world DLS, and greatly increase these DLS's accuracy.

References

[1] Baidu Apollo team (2017), Apollo: Open Source Autonomous Driving, howpublished = https://github.com/apolloauto/apollo, note = Accessed: 2019-02-11.
[2] Cifar10 model in keras.

[3] Douglas G Altman and J Martin Bland. How to obtain the confidence interval from a p value. Bmj, 343, 2011.
[4] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In Ndss, volume 14, pages 23–26, 2014.
[5] ZW Birnbaum. On a use of the mann-whitney statistic. In Volume 1 Contribution to the Theory of Statistics, pages 13–18. University of California Press, 2020.
[6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp), pages 39–57. IEEE, 2017.
[7] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018.
[8] Contagio. Contagio, pdf malware dump. 2010.
[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.
[10] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine, 29(6):141–142, 2012.
[11] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. Importance-driven deep learning system testing. In 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pages 702–713. IEEE, 2020.
[12] Charles J Geyer. Practical markov chain monte carlo. Statistical science, pages 473–483, 1992.
[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014.
[14] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435, 2016.
[15] Bradley Harding, Christophe Tremblay, and Denis Cousineau. Standard errors: A review and evaluation of standard error estimators using monte carlo simulations. The Quantitative Methods for Psychology, 10(2):107–123, 2014.
[16] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. Is neuron coverage a meaningful measure for testing deep neural networks? In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 851–862, 2020.
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
[18] David W Hosmer and Stanley Lemeshow. Confidence interval estimation of interaction. Epidemiology, pages 452–456, 1992.
[19] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.
[20] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), pages 287–296. IEEE, 2018.
[21] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pages 1039–1049. IEEE, 2019.
[22] Ravin Kumar, Colin Carroll, Ari Hartikainen, and Osvaldo Martin. Arviz a unified library for exploratory analysis of bayesian models in python. Journal of Open Source Software, 4(33):1143, 2019.
[23] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In Advances in neural information processing systems, pages 396–404, 1990.
[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.
[25] Jun Li, Bodong Zhao, and Chao Zhang. Fuzzing: a survey. Cybersecurity, 1(1):1–13, 2018.
[26] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. Structural coverage criteria for neural networks could be misleading. In 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pages 89–92. IEEE, 2019.
[27] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In International Conference on Machine Learning, pages 3208–3216. PMLR, 2018.
[28] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pages 120–131, 2018.
[29] Dougal Maclaurin. Modeling, inference and optimization with composable differentiable procedures. PhD thesis, 2016.
[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083, 2017.
[31] Valentin Jean Marie Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J Schwartz, and Maverick Woo. The art, science, and engineering of fuzzing: A survey. IEEE Transactions on Software Engineering, 2019.
[32] Ravi Mangal, Kartik Sarangmath, Aditya V Nori, and Alessandro Orso. Probabilistic lipschitz analysis of neural networks. In International Static Analysis Symposium, pages 274–309. Springer, 2020.
[33] Albert W Marshall and Ingram Olkin. Multivariate chebyshev inequalities. The Annals of Mathematical Statistics, pages 1001–1014, 1960.
[34] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: an overview. Explainable AI: interpreting, explaining and visualizing deep learning, pages 193–209, 2019.
[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703, 2019.
[36] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In proceedings of the 26th Symposium on Operating Systems Principles, pages 1–18, 2017.
[37] Arash Rahnama, Andre T Nguyen, and Edward Raff. Robust design of deep neural networks against adversarial attacks based on lyapunov theory. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8178–8187, 2020.
[38] Samik Raychaudhuri. Introduction to monte carlo simulation. In 2008 Winter simulation conference, pages 91–100. IEEE, 2008.
[39] Christian P Robert, Peter J Bickel, P Diggle, S Fienberg, K Krickeberg, I Olkin, N Wermuth, and S Zeger. Discretization and MCMC convergence assessment, volume 135. Springer Science & Business Media, 1998.
[40] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using PyMC3. PeerJ Computer Science, 2:e55, apr 2016.
[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
[42] Michael Sutton, Adam Greene, and Pedram Amini. Fuzzing: brute force vulnerability discovery. Pearson Education, 2007.
[43] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In Proceedings of the 40th international conference on software engineering, pages 303–314, 2018.
[44] Udacity-challenge. Using deep learning to predict steering angles. 2016.
[45] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. Robot: Robustness-oriented testing for deep learning systems. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pages 300–311. IEEE, 2021.
[46] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In Proceedings of the ieee conference on computer vision and pattern recognition, pages 4480–4488, 2016.
[47] Yi Zhou, Li Liu, Ling Shao, and Matt Mellor. Dave: A unified framework for fast vehicle detection and annotation. In European Conference on Computer Vision, pages 278–293. Springer, 2016.