applied sciences

MDPI

Article

ExtendAIST: Exploring the Space of AI-in-the-Loop System Testing

Tingting Wu, Yunwei Dong *, Yu Zhang and Aziz Singa

School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China; tingtingwu@mail.nwpu.edu.cn (T.W.); zhangyu@nwpu.edu.cn (Y.Z.); azsinga81@mail.nwpu.edu.cn (A.S.)

* Correspondence: yunweidong@nwpu.edu.cn; Tel.: +86-1599-161-8002

Received: 5 December 2019; Accepted: 7 January 2020; Published: 10 January 2020

Abstract: The AI-in-the-loop system (AIS) has been widely used in various autonomous decision and control systems, such as computing vision, autonomous vehicle, and collision avoidance systems. AIS generates and updates control strategies through learning algorithms, which make the control behaviors non-deterministic and bring about the test oracle problem in AIS testing procedure. The traditional system mainly concerns about properties of safety, reliability, and real-time, while AIS concerns more about the correctness, robustness, and stiffness of system. To perform an AIS testing with the existing testing techniques according to the testing requirements, this paper presents an extendable framework of AI-in-the-loop system testing by exploring the key steps involved in the testing procedure, named ExtendAIST, which contributes to define the execution steps of ExtendAIST and design space of testing techniques. Furthermore, the ExtendAIST framework provides three concerns for AIS testing, which include: (a) the extension points; (b) sub-extension points; and (c) existing techniques commonly present in each point. Therefore, testers can obtain the testing strategy using existing techniques directly for corresponding testing requirements or extend more techniques based on these extension points.

Keywords: AI-in-the-loop system; machine learning; AIS testing; AIS testing strategy

1. Introduction

With the broad applications of artificial intelligence on wearable devices, autonomous cars, and smart city, the traditional embedded system with predefined control strategies, including sensor, controller, and actuator, has evolved into the AI-in-the-loop system (AIS). AI-in-the-loop system is an intelligent system with capabilities of self-learning and autonomous decision because of embedded Artificial Intelligence (AI) components/modules or implementation of AI solutions in an embedded system. For instance, autonomous cars [1] perceive the surrounding driving environment by sensors such as radar and camera. Based on the perception data, autonomous cars perform tasks such as obstacles detection and tracking, traffic signals detection and recognition, and route planning through the AI modules implemented under machine learning algorithm. Autonomous cars then make decisions for these tasks and drive automatically on road via the autonomous interactions with driving environment.

The safety-critical system with AI module embedded is potentially unsafe. For example, the adaptive cruise control system is an intelligent system to make cars drive automatically by interacting with the driving environment and learning the corresponding driving behaviors, such as speed up, slow down, and brake. However, the traffic accident of Tesla Model S, which crashes because of the misclassification between the sky and white truck, indicates that the AI module may be vulnerable to the perturbation and make AI-in-the-loop system unsafe. Once some faults in AI module occur without any safety control behavior from system or person, there will be either error decision

or fatal consequence. Therefore, it is crucial to detect the erroneous behaviors in AIS and assure its properties (e.g., correctness and robustness) through certain testing techniques.

Some researchers have reviewed the progress of machine learning testing from different perspectives. For example, Hains et al. [2] investigated the existing verification and simulation techniques for the safety of machine learning. Masuda et al. [3] reviewed the error detection of machine learning and the applications of conventional software testing on machine learning. Braiek et al. [4] studied the datasets, and black- and white-box based testing techniques for machine learning testing. Ma et al. [5] discussed the safety challenge in machine learning system. Huang et al. [6] introduced the testing and verification techniques for the safety of machine learning. Zhang et al. [7] presented a comprehensive survey on machine learning testing.

However, the above-mentioned studies mainly focus on the testing or verification techniques without a systematic testing procedure from test data generation to evaluation except the review in [7], and few studies present the differences between testing conventional system and AI-in-the-loop system. Even though Zhang et al. [7] presented a thorough introduction of machine learning testing, including the techniques of detecting data, learning program, framework, open-source tool, and testing objects, they did not mention how to develop a new technique and the further work researchers can conduct based on each testing step. Inspired by the above enquiries, we follow a similar research method as Astor [8] to propose an Extendable framework of AIS Testing, namely ExtendAIST. The ExtendAIST takes each involved testing step as extension point and implemented approaches as sub-extension points to organize the design space of AIS testing techniques practically. Additionally, this framework provides the existing techniques commonly present in individual sub-extension point. Therefore, debuggers can reuse these techniques, extend new techniques over the extension and sub-extension points, or implement new points in the space of AIS testing.

The primary contributions of our paper are summarized as follows.

- The AI-in-the-loop system is proposed to discuss the differences between testing ordinary system and AI-in-the-loop system in terms of testing coverage, test data, testing property, and testing technique based on the distinct nature of AI modules.
- The testing workflow of AI-in-the-loop system is presented to show the individual step including dataset generation, training and testing for AI-in-the-loop system.
- To the best of our knowledge, this is the first time researchers explore the design space of AI-in-the-loop system testing and present an extendable framework ExtendAIST. This framework provides five extension points, 19 sub-extension points, and existing techniques that researchers can use directly or extend further new techniques for corresponding testing requirements.

The remainder of this paper is organized as follows. Section 2 introduces the motivation of ExtendAIST including the architecture of AI-in-the-loop system, the differences between testing ordinary system and AI-in-the-loop system, and the AIS testing workflow. Section 3 describes the architecture, design, and illustration example of ExtendAIST. Section 4 presents the extension points, sub-extension points, and existing techniques in ExtendAIST. Discussions and conclusions are given in Sections 5 and 6.

2. Motivation

2.1. AI-in-the-Loop System

The conventional embedded system takes as input the data from sensors and computes the control strategies according to the program with specific task. The controllers output the corresponding decision based on the control strategies and transform this decision into a command to actuators. Then, actuators consequently take some operations to control the entire system. Thus, the conventional embedded system is logic deterministic with the certain program and control behaviors.

Different from the conventional embedded system, the control strategies of AI-in-the-loop system are obtained by learning from training data under the learning algorithm, which will lead to less precise and non-deterministic strategies. As shown in Figure 1, the AIS learns the online or offline strategies by online learning or offline learning from the sensor data or offline data under the learning algorithm for specific application. Then, it makes a decision according to the learned strategies and forwards this decision to actuators in the format of command. In this case, the system with AI module embedded can perform the autonomous learning, decision making, and controlling with machine learning program. The testing of conventional system follows a general testing procedure: selecting test coverage, generating test cases, executing test cases, and compare the actual outputs with the expected outputs. Nevertheless, because of the nature of machine learning program, the control strategies vary even for the same training data, which may generate different or uninterpretable control behaviors and weaken the correctness and robustness of AI-in-the-loop system. Therefore, there exists the oracle problem [9] in testing AI-in-the-loop system.

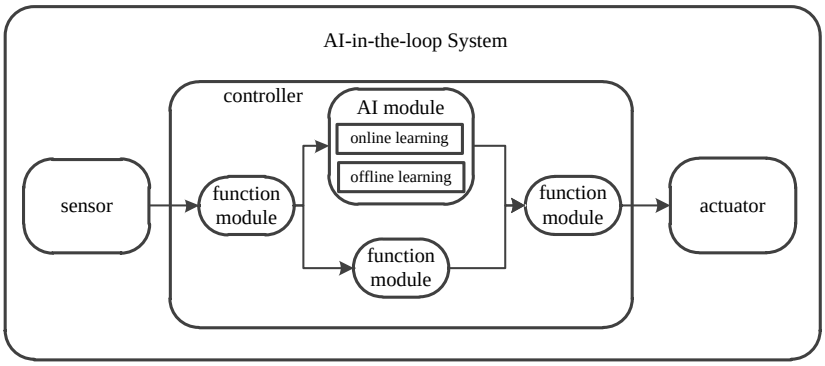


Figure 1. Architecture of the AI-in-the-loop system.

2.2. *AI-in-the-Loop System Testing*

The nature of AI module design and implementation, the complex requirements, large-scale network, non-deterministic and uninterpretable learning results bring great challenges in testing AI-in-the-loop system [10–12], which distinguishes from traditional system testing. The traditional system is always a logic deterministic program with the definite output against the system under testing (SUT) and the corresponding test case. The execution behavior can be tracked and interpretable with the absolutely deterministic control flow and data flow. However, the output of AI module is non-deterministic. For example, the machine learning based AI module is a data-driven program [7] which delivers a trained model by learning from the training dataset under the learning algorithm. The predicted label of the trained model may be distinct even for the same test data, because the trained model evolves with different training datasets so that different weights and biases are trained. The testing approaches commonly used in testing ordinary system and non-AI modules are not suitable for testing the AI modules. Therefore, more novel testing coverage metrics and test data are necessary when performing the unit testing of AI-in-the-loop system. More testing properties and testing techniques should be investigated when conducting the system testing of AI-in-the-loop system for the behavior consistency.

For the unit testing of traditional system and AI-in-the-loop system, the test adequacy of traditional system can be measured by control flow coverage (e.g., SC, DC, CC, and MC/DC) and data flow coverage (e.g., p-use and c-use). Except for the coverage metrics used above for non-AI modules, the test adequacy of AI modules should be measured by the structure coverage of neural network because of its non-deterministic logic, such as neuron level coverage [13–15], layer level coverage [15], and neuron pair level coverage [16]. Based on the above, all activated neurons leading to major function behaviors and corner case behaviors can be covered. The common dataset designed for different application areas sometimes consists of training and testing data. However, the common

dataset is not sufficient for each testing task. To meet the specific testing tasks of AI-in-the-loop system, new dataset including training dataset and testing dataset is generated by some test data generation strategies. The training dataset is used to train the learning program to get a trained model, and the testing dataset is used to evaluate the training effectiveness of trained model.

For the system testing of traditional system and AI-in-the-loop system, the traditional system testing mainly focuses on the following functional and non-functional properties, such as correctness, real-time, reliability and safety. Due to the non-deterministic behavior of AI module, the AI-in-the-loop system suffers from the oracle problem. Therefore, more properties beyond those of traditional systems, such as the robustness [7] and stiffness [17] of the AI-in-the-loop system, should be tested by checking the behavior consistency. The robustness measures the resilience of AI-in-the-loop system towards perturbations on data or learning program. For example, if the trained model misclassifies the test data with imperceptible perturbation, then AI-in-the-loop system is said to be of low robustness. The stiffness is proposed to measure the resistance of AI-in-the-loop system by the effect of a small gradient step on one test data upon the loss on another test data. That is, the predicted label of AI-in-the-loop system is convergent for the gradient alignment between different test data. To test unique properties of the AIS, more testing techniques such as adversarial attack [18] and generative adversarial network [19] are proposed. The adversarial attack is a technique generating adversarial perturbations on the original test data to determine whether the AIS is vulnerable to adversarial perturbations.

Testing AI-in-the-loop system is a process of detecting system erroneous behaviors, which aims to guarantee the properties of system, such as correctness and robustness. The AIS learns from the input training dataset and predicts the behavior with the trained model. Therefore, the correctness of the system can be determined by checking whether the output behaviors are consistent with the requirements. The robustness can be determined by checking whether the output behaviors are affected by the adversarial perturbation. As shown in Figure 2, given an AI-in-the-loop system, the testing activity is conducted as the following steps.

1. Generate the test sample, including training dataset and testing dataset, through some test data generation algorithms. The training dataset can be selected from the common benchmarks or designed for some special application areas to train the learning model. The testing dataset is generated with the requirement of test coverage to test the trained model.
2. When testing the ordinary system, the testing procedure follows the black arrowlines in Figure 2. The system executes the testing dataset and outputs the deterministic decision according to the predefined rules.
3. When testing the AI-in-the-loop system, the testing procedure follows the red and black arrowlines in Figure 2 for AI module and other function modules, respectively. The trained model is learnt from the training dataset under the training model. Then, the AIS executes the testing dataset and outputs the predicted non-deterministic decision.
4. After test execution procedure, the test report is generated to indicate the test results. The failed test demonstrates an erroneous behavior of system, and the passed test shows the behavior consistency.
5. If some errors are detected during the testing procedure, then conduct the regression testing after all errors are repaired to make sure that no new error is introduced.
6. If no error is detected, then terminate the testing procedure.

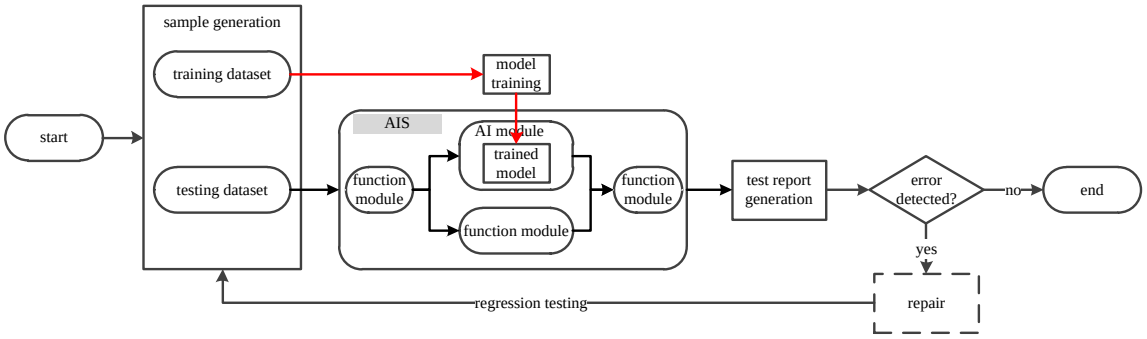
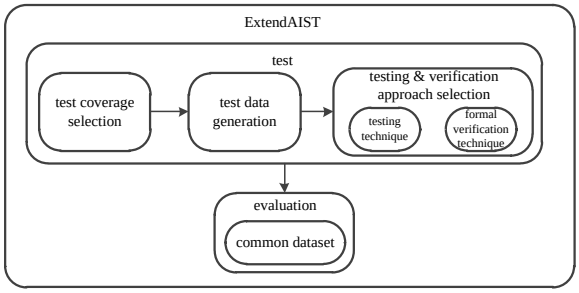


Figure 2. Workflow of the AI-in-the-loop system testing.

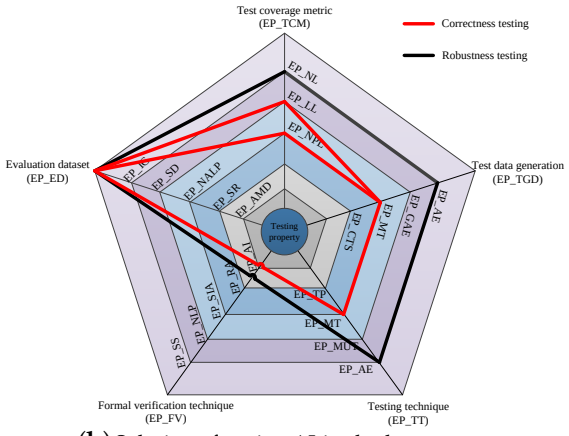
3. ExtendAIST

3.1. Architecture

ExtendAIST encodes the design space of AIS testing, which involves four key elements: test coverage selection, test data generation, testing and verification approach, and evaluation with common dataset. Figure 3a shows the architecture of the ExtendAIST framework, in which the four elements are four dimensions for designing new techniques. In other words, researchers can develop a new technique by choosing one or more elements. Our proposed framework ExtendAIST provides a solution for designing comprehensive testing techniques by connecting these elements from test to evaluation, so that the developer can not only develop new techniques based on the four dimensions but also evaluate the effectiveness of new techniques.



(**a**) Architecture of ExtendAIST



(**b**) Solution of testing AI-in-the-loop system

Figure 3. Design of ExtendAIST.

Since testing technique reveals the erroneous behaviors in AI-in-the-loop system and formal verification technique ensures the properties of the system, we divide the test and verification approaches into two kinds of elements including testing technique and formal verification technique.

Then, we have five elements, referred to as five extension points, to form the design space of AIS testing. Figure 3b indicates the potential solution of testing AI-in-the-loop system, in which the five nodes of the pentagon are the five extension points above, the edge from each node to the center implies the possible techniques used to meet the requirement of testing property, and all the possible techniques are regarded as the sub-extension points (see Section 4). For example, given a testing property, debuggers will select one or more suitable sub-extension points in each extension point to complete this test task. Therefore, the ExtendAIST is a framework providing extension points, sub-extension points, and existing techniques that allow testers to obtain a testing strategy based on existing techniques or extend novel techniques by choosing one or more extension points.

3.2. Design

According to the testing workflow defined in Section 2.2, we present the main steps executed in ExtendAIST, as shown in Algorithm 1, which outputs a recommendation of testing strategy for the input testing requirements of AI-in-the-loop system under test. ExtendAIST first investigates each testing requirement from the input testing requirements (Lines 2–20), such as the testing coverage requirement, testing property requirement, and application area requirement. According to the requirement, it inspects and selects the corresponding extension point available for the testing requirements (Lines 6–18). Then, the algorithm inspects and selects the appropriate sub-extension points for the testing requirements (Lines 11–17). All the existing techniques in the selected sub-extension points can be used (Line 16). It generates the testing strategy for the selected testing requirement (Line 19). Finally, it returns the recommended testing strategy for all of the input testing requirements (Line 21). The time complexity of Algorithm 1 is $O(n^2)$.

Algorithm 1 Execution steps of testing AI-in-the-loop system in ExtendAIST.

Require: Testing requirement set R of AI-in-the-loop system under test
Ensure: Testing strategy
1: $Strategy \leftarrow \varnothing$ // Initialize the testing strategy
2: **for** $i = 1 \rightarrow n$ **do** // There are n requirements in the set R
3: $ExtenP \leftarrow \varnothing$ // Initialize the set of selected extension points for the ith requirement
4: $SubExtenP \leftarrow \varnothing$ // Initialize the set of selected sub-extension points for the ith requirement
5: Investigate(R_i) // Investigate the ith testing requirement
6: **for** $j = 1 \rightarrow 5$ **do** // Five extension points
7: $f_{ep} \leftarrow$ Inspect(EP_j) // Inspect the jth extension point
8: **if** $f_{ep} ==$ true **then**
9: $ExtenP \leftarrow ExtenP \cup EP_j$ // The jth extension point is appropriate for the ith testing requirement
10: **end if**
11: **for** $k = 1 \rightarrow m_j$ **do** // m_j sub-extension points in the jth extension point
12: $f_{sep} \leftarrow$ Inspect(SEP_j^k) // Inspect the kth sub-extension point
13: **if** $f_{sep} ==$ true **then**
14: $SubExtenP \leftarrow SubExtenP \cup SEP_j^k$ // The kth sub-extension point in the jth extension point is appropriate for the ith testing requirement
15: **end if**
16: $Tech \leftarrow$ Select($Tech_j^k$) // Select the existing techniques in the kth sub-extension point of the jth extension point to test the AIS
17: **end for**
18: **end for**
19: $Strategy[i] \leftarrow (R_i, ExtenP, SubExtenP, Tech)$ // Generate the strategy for the ith testing requirement
20: **end for**
21: **return** $Strategy$ // Output the recommendation of testing strategy for the input requirement

3.3. Illustration

In this section, we take the security and stability intelligent control unit in power system terminal as an illustration example to show the detailed implementation steps of testing AIS following the design space of ExtendAIST and the integration of other available techniques in ExtendAIST.

The security and stability intelligent control unit in power system terminal is an AI-in-the-loop system which aims at predicting exceptions according to the learnt control strategies and taking relevant actions to avoid the potential failure for power transmission. As shown in Figure 4, the security and stability intelligent control unit in power system terminal obtains the transmission sections between the power supply center and electric load center, including the data of current, voltage, power, and frequency. This information is then transferred to the control strategies table to predict the decisions according to the learned strategies. For example, if the current value s_1 is greater than the threshold P_1, then decision D_1 is decided according to the learned control strategies and transferred to switch off the actuator. Error decisions lead to illegal actions on actuators which can cause incalculable loss. Thus, it is critical to test the security and stability intelligent control unit in power system terminal to reveal the erroneous behaviors and ensure its security and stability.

According to the implementation steps of ExtendAIST, we test the correctness of security and stability intelligent control unit in power system terminal during the process of power transmission. As shown in Figure 3b, the red thick line is one of the solutions of testing the correctness of security and stability intelligent control unit. Firstly, debuggers select the layer level coverage [15] and neuron pair level coverage [16] to measure the numbers of hyperactive neurons in each layer and the influence between layers and ensure the test adequacy of test data generated from metamorphic testing based strategy. Secondly, they take the experienced decisions when exceptions occur as original test data, because no common dataset exists for the security and stability intelligent control unit. Then, they generate new test data with the metamorphic testing based strategy [20,21] on the source test data. Thirdly, once generating the test data, debuggers can test the control strategies learned in AI module by metamorphic testing and determine test results by checking whether metamorphic relation is satisfied. If there are faults in control strategies, e.g., an illegal action of switching on instead of switching off actuators is decided when overloading, debuggers can repair this system by proper repair techniques, such as changing the parameters of training model. Finally, they evaluate the fixed system with the experienced dataset to show the repair quality.

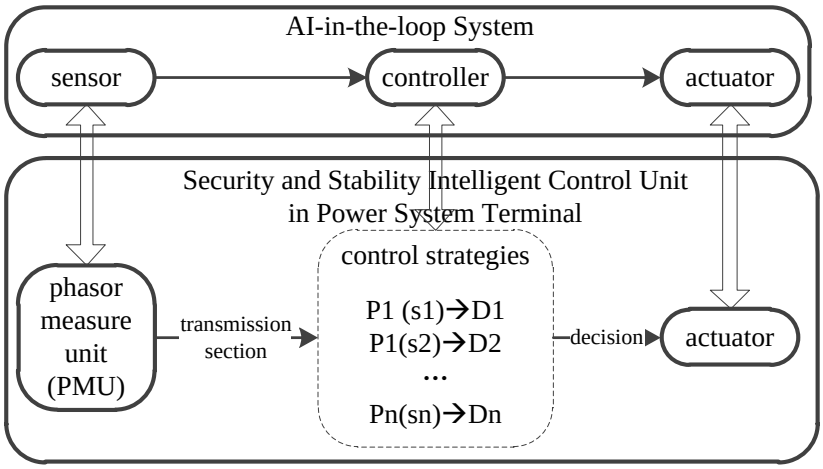


Figure 4. Security and stability intelligent control unit in power system terminal.

Since the correctness testing of security and stability intelligent control unit in power system terminal is similar to the traditional system testing, and the metamorphic testing can also be applied in traditional system testing, we further test the robustness of security and stability intelligent control unit to show the differences compared to the traditional system testing. As shown in Figure 3b,

the black thick line is one of the solutions of testing the robustness of security and stability intelligent control unit. Firstly, debuggers select the neuron level coverage [13–15] to measure the numbers of activated neurons so that the major function behaviors and corner-case function behaviors are covered. The adversarial attack [18] is proven to be a promising technique for testing robustness of machine learning system. Therefore, adversarial attack is chosen to generate adversarial examples and test whether the control strategies in AI module misclassify a small perturbation. For example, the decision of the original current value s_1 is D_1 if $s_1 < P_1$. The adversarial example $s_1^{'}$ is greater than s_1 and smaller than P_1, that is, $s_1 < s_1^{'} < P_1$, then the AI module should predict the same decision D_1. If a different decision is predicted, then the security and stability intelligent control unit is said to be of low robustness. Finally, debuggers repair this system and evaluate the fixed system with the experienced dataset to show the repair quality.

4. Extension Points and Sub-Extension Points

In this section, we explain the five extension points and 19 sub-extension points provided in the ExtendAIST framework for testers using the existing techniques directly or develop new techniques based on these points for the corresponding testing requirements. Each extension point is a key step involved in testing AIS, which is depicted with existing techniques in this framework. The five extension points, 19 sub-extension points, and existing techniques for individual point are summarized in Figure 3b and Table 1, respectively. As shown in Table 2, the recommendation of testing strategy including extension points and sub-extension points are selected for the corresponding testing requirement. "✓" implies that the sub-extension point is available for the related testing requirement. Take the "Correctness" row as example, all the coverage metrics in different granularity levels of extension point EP_TCM can be used to measure the coverage of network. All five kinds of evaluation common datasets are available for the correctness testing. In the extension point of EP_TDG, only sub-extension point EP_MT can be used to generate test data for testing correctness of AIS. When selecting the testing technique, sub-extension points EP_MUT, EP_MT, and EP_TP are appropriate to perform AIS correctness testing.

Table 1. Summary of extension points and existing techniques in ExtendAIST.

Extension Point	Sub-Extension Point	Description	Existing Technique
Test Coverage Metric (EP_TCM)	Neuron level (EP_NL)	The coverage of activated neurons for the major and corner-case behaviors	Activated neuron coverage [13,14], Major and corner-case coverage [15]
	Layer level (EP_LL)	The number of the most active k neurons in each layer	Top-k neuron coverage [15]
	Neuron pair level (EP_NPL)	The sign (or distance) change of neuron n_l^i (or layer l) affects the sign (or value) of neuron n_{l+1}^j	Sign/Distance–Sign/ Value cover [16]
Test Data Generation (EP_TDG)	Metamorphic testing based strategy (EP_MT)	Generate follow-up test data with the noise from the metamorphic relation	Classifier testing [20,21], DeepTest [14], noise effect [22]
	Adversarial examples (EP_AE)	Generate test data with imperceptible perturbations to the input	L-BFGS [18], FGSM [23], BIM [24], ILCM [25], JSMA [26]
	Generative adversarial examples (EP_GAE)	Generate test data using generative adversarial nets	Generative adversarial nets [19]
	Concolic testing based strategy (EP_CTS)	Generate test data with the smallest distance to the input by the concrete execution and symbolic execution	DeepConcolic [27]
Testing Technique (EP_TT)	Adversarial attack (EP_AE)	Reveal the defects in DNN by executing adversarial examples	Targeted, non-targeted, L_p-norm attack [28,29]
	Mutation testing (EP_MUT)	Evaluate the testing adequacy by mutating training data, training program and trained model	MuNN [30], DeepMutation [31]
	Metamorphic testing (EP_MT)	Determine the system correctness by checking whether the metamorphic relation is satisfied	DeepTest [14]
	Test prioritization (EP_TP)	Measure the correctness of classification by the purity of test data	DeepGini [32]
Formal Verification Technique (EP_FV)	Satisfiability Solver (EP_SS)	Transform the safety verification to satisfiability solver problem	Safety verification [33]
	Non-linear problem (EP_NLP)	Transform the safety verification to non-linear problem	Piecewise linear network verification [34]
	Symbolic interval analysis (EP_SIA)	Transform the safety verification by analyzing symbolic interval	ReluVal [35]
	Reachability analysis (EP_RA)	Transform the safety verification by analyzing the reachability problem	DeepGo [36]
	Abstract interpretation (EP_AI)	Transform the safety verification to abstract interpretation	AI^2 [37], Symbolic propagation [38]
Evaluation Dataset (EP_ED)	Image classification (EP_IC)	Dataset for image recognition	MNIST [39,40], EMNIST [41,42], Fashion-MNIST [43,44], ImageNet [45,46], CIFAR-10/CIFAR-100 [47]
	Self driving (EP_SD)	Dataset for training and testing autonomous car system	Udacity Challenge [48], MSCOCO 2015 [49,50], KITTI [51–53], Baidu Apollo [54]
	Natural language processing (EP_NALP)	Dataset for text processing	Enron Email Dataset [55], bAbI [56],Text Classification Datasets [57], SNLI [58,59]
	Speech recognition (EP_SR)	Dataset for speech recognition	Speech Commands [60,61], Free Spoken Digit Dataset [62], Million Song Dataset [63,64], LibriSpeech [65,66]
	Android malware detection (EP_AMD)	Dataset for Android malware detection	Drebin [67–69], Genome [70,71], VirusTotal [72],Contagio [73]

Table 2. Summary of testing strategy recommendation with extension points provided in ExtendAIST.

Testing Requirement	Test Coverage Metric (EP_TCM)			Test data generation (EP_TDG)				Testing Technique (EP_TT)				Formal Verification Technique (EP_FV)					Evaluation Dataset (EP_ED)				
	EP_NL	EP_LL	EP_NPL	EP_MT	EP_AE	EP_GAE	EP_CTS	EP_AE	EP_MUT	EP_MT	EP_TP	EP_SS	EP_NLP	EP_SIA	EP_RA	EP_AI	EP_IC	EP_SD	EP_NALP	EP_SR	EP_AMD
Neuron coverage	✓																				
Network layer coverage		✓																			
Layers influence coverage			✓																		
Correctness	✓	✓	✓	✓					✓	✓	✓						✓	✓	✓	✓	✓
Robustness	✓	✓	✓		✓	✓	✓	✓									✓	✓	✓	✓	✓
Safety	✓	✓	✓									✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Image processing																	✓	✓			
Text processing																			✓		
Audio processing																				✓	
Malware detection																					✓

4.1. Test Coverage Metric (EP_TCM)

4.1.1. Description

Test coverage metric selection is the first step of test data generation as test data must satisfy the requirement of coverage criterion. There already exist many test coverage metrics for traditional software testing adequacy: statement coverage, condition coverage, decision coverage, and Modified Condition/Decision Coverage (MC/DC). However, all the metrics above cannot be used directly to cover the control and data flow of AI-in-the-loop system due to the structure of learning program for AI module and the non-deterministic program logic. Researchers take neuron as the basic coverage unit to conduct both major function behavior coverage and corner-case behavior coverage of neural network [15]. We divide the neural network coverage into three categories from the perspective of granularity: *neuron level coverage*, *layer level coverage*, and *neuron pair level coverage*.

Therefore, ExtendAIST takes test coverage metric as an extension point EP_TCM, and neuron level coverage, layer level coverage, and neuron pair level coverage as sub-extension points named EP_NL, EP_LL, and EP_NPL, respectively. Then, testers can implement new coverage metric based on these extension points or reuse the existing techniques.

4.1.2. Neuron Level (EP_NL)

The neuron level coverage depends on the output value of neuron. There currently exist four kinds of neuron level coverage in this sub-extension point EP_NL: *activated neuron coverage*, *k-multisection neuron coverage*, *neuron boundary coverage*, and *strong neuron activation coverage*.

- Activated neuron coverage. A neuron is considered *activated* if the neuron output is greater than the neuron activation threshold and makes contribution to neural network's behaviors including major function and corner-case behaviors. As shown in Equation (1), the *activated neuron coverage* [13,14] Cov_{ANC} is the rate of the number of neurons activated and the number of neurons in the whole DNN.

$$Cov_{ANC} = \frac{n_{activated}}{N} \tag{1}$$

- *k*-multisection neuron coverage. Ma et al. [15] proposed using neuron output value range $[low_n, high_n]$ to distinguish the major function region and corner-case region. Then, we can measure the coverage of major function region $[low_n, high_n]$ by dividing this region into k equal subsections. As shown in Equation (2), *k-multisection neuron coverage* for a neuron n, $Cov_{k,n}$ is the rate of the number of subsections covered by T and the total number of subsections, in which x is a test input in dataset T, $\phi(x, n)$ is the output of neuron n with test input x, and S_i^n is the set of values in the ith subsection. The *k-multisection neuron coverage* for the neural network N, Cov_{KMN}, is based on the *k*-multisection neuron coverage of all neurons in network, which is defined in Equation (3).

$$Cov_{k,n} = \frac{|\{S_i^n | \exists x \in T : \phi(x, n) \in S_i^n\}|}{k} \tag{2}$$

$$Cov_{KMN} = \frac{\Sigma_{n \in N} |\{S_i^n | \exists x \in T : \phi(x, n) \in S_i^n\}|}{k \times |N|} \tag{3}$$

- Neuron boundary coverage. In some cases, neuron output $\phi(x, n) \notin [low_n, high_n]$. That is, $\phi(x, n)$ may locate $(-\infty, low_n) \cup (high_n, +\infty)$, which is referred to as the corner-case region of neuron n. Therefore, the *neuron boundary coverage* Cov_{NBC}, the rate of the number of neurons falling in corner-case region and the total number of corner cases as in Equation (4), is used to measure how many corner-case regions are covered by test dataset T. $N_{UPPER} = \{n \in N | \exists x \in T : \phi(x, n) \in (high_n, +\infty)\}$ is the set of neurons located in the upper corner-case region, and $N_{LOWER} = \{n \in N | \exists x \in T : \phi(x, n) \in (-\infty, low_n)\}$ is the set of neurons located in the lower corner-case region. The total number of corner cases of neuron boundary coverage is equal to

$2 \times |N|$, because $(-\infty, low_n)$ and $(high_n, +\infty)$ are mutually exclusive and neuron cannot fall in two regions at the same time.

$$Cov_{NBC} = \frac{|N_{UPPER}| + |N_{LOWER}|}{2 \times |N|} \tag{4}$$

- Strong neuron activation coverage. Because hyperactive corner-case neurons affect the training of DNN significantly, it is essential to measure the coverage of hyperactive corner-case neurons, namely *strong neuron activation coverage*. Similar to the neuron boundary coverage, strong neuron activation coverage Cov_{SNA} is the rate of the number of neurons falling in the upper corner-case region and the total number of corner cases as in Equation (5).

$$Cov_{SNA} = \frac{|N_{UPPER}|}{|N|} \tag{5}$$

4.1.3. Layer Level (EP_LL)

In the layer level sub-extension point EP_LL, the neuron coverage from the perspective of top hyperactive neurons in each layer has been further investigated [15]. An effective input test dataset should cover more and more hyperactive neurons in one layer. In this sub-extension point, the top-k neuron coverage has been implemented to cover the top-k neurons in the same layer.

- Top-k neuron coverage. Given test input x and neurons n_1 and n_2 in the same layer, if $\phi(x, n_1) > \phi(x, n_2)$, then n_1 is more active than n_2. Here, the *top-k neuron coverage* is designed to measure how many neurons are activated as the most k neurons in each layer by the input test dataset T. As shown in Equation (6), $top_k(x, i)$ is the set of first k neurons, which are ranked in descending order of their outputs.

$$Cov_{TKN} = \frac{|\cup_{x \in T} (\cup_{1 \le i \le l} top_k(x, i))|}{|N|} \tag{6}$$

4.1.4. Neuron Pair Level (EP_NPL)

The neuron pair level coverage focuses on the propagation of changes between neurons from adjacent layers. Inspired by the MC/DC criterion, Sun et al. [16] proposed the neuron pair level coverage by taking $\alpha = (n_l^i, n_{l+1}^j)$ as the neuron pair, in which n_l^i is a neuron regarded as a condition in the lth layer and n_{l+1}^j is a neuron regarded as a decision in the $(l+1)$th layer. Hence, neuron pair level coverage is presented to inspect the influence of neurons in the lth layer on neurons in the $(l+1)$th layer. The change of neuron n_l^k when given two test inputs x_1 and x_2 could be a *sign change* (denoted as $sc(n_l^k, x_1, x_2)$), a *value change* (denoted as $vc(g, n_l^k, x_1, x_2)$, where g is a value change function), and a *distance change* (denoted as $dc(h, l, x_1, x_2)$, where h is a distance change function) for neurons in the lth layer. Therefore, given the neuron pair $\alpha = (n_l^i, n_{l+1}^j)$ and two test inputs x_1 and x_2, there exit four implemented techniques in this sub-extension point of neuron pair level coverage.

- Sign–Sign cover. The sign change of condition neuron n_l^i and signs of other neurons in the lth layer not changing affect the sign of decision neuron n_{l+1}^j in the next layer. That is, if $sc(n_l^i, x_1, x_2) \wedge \neg sc(n_l^k, x_1, x_2) \wedge (k \neq i) \Rightarrow sc(n_{l+1}^j, x_1, x_2)$, we say that (n_l^i, n_{l+1}^j) is *sign–sign covered* by x_1 and x_2 which is denoted as $cov_{SS}(\alpha, x_1, x_2)$.
- Distance–Sign cover. The small distance change of neurons in the lth layer can cause the sign change of decision neuron n_{l+1}^j in the next layer. Namely, if $dc(h, l, x_1, x_2) \Rightarrow sc(n_{l+1}^j, x_1, x_2)$, we say that (n_l^i, n_{l+1}^j) is *distance–sign covered* by x_1 and x_2, denoted as $cov_{DS}^h(\alpha, x_1, x_2)$.
- Sign–Value cover. Similar to sign–sign cover, the sign change of condition neuron n_l^i and signs of other neurons in the lth layer not changing affect the value of decision neuron n_{l+1}^j in the

next layer. That is, if $sc(n_l^i, x_1, x_2) \wedge \neg sc(n_l^k, x_1, x_2) \wedge (k \neq i) \Rightarrow vc(g, n_{l+1}^j, x_1, x_2)$, we say that (n_l^i, n_{l+1}^j) is *sign–value covered* by x_1 and x_2, denoted as $cov_{SV}^g(\alpha, x_1, x_2)$.

- Distance–Value cover. Similar to distance–sign cover, the small distance change of neurons in the lth layer leads to the value change of decision neuron n_{l+1}^j in the next layer. Namely, if $dc(h, l, x_1, x_2) \Rightarrow vc(g, n_{l+1}^j, x_1, x_2)$, then (n_l^i, n_{l+1}^j) is *distance–value covered* by x_1 and x_2, denoted as $cov_{DV}^{h,g}(\alpha, x_1, x_2)$.

4.2. Test Data Generation (EP_TDG)

4.2.1. Description

During training procedure, some common datasets are used as the training sets to compare the training performance of different training models. During testing procedure, the testing data included in the common datasets are used to measure the training effectiveness and correctness of AI-in-the-loop system. Thus, both training and testing procedures are data-driven. Since the AIS may output different labels for inputs with high similarity, generating test data that can cover not only major function behaviors but also corner-case behaviors becomes an indispensable activity for AI-in-the-loop system testing.

Thus, test data generation is regarded as an extension point, namely EP_TDG, in ExtendAIST. Since various generation strategies and their variants have been investigated over the past decades, there are mainly four kinds of strategies present to generate test data: adversarial examples, generative adversarial examples, metamorphic testing based strategy, and concolic testing based strategy. These four kinds of common used strategies are also regarded as the sub-extension points, respectively: EP_AE, EP_GAE, EP_MT, and EP_CTS. All of these points allow designing the strategy of test data generation for ExtendAIST producing relevant test samples to train, test, and evaluate the AI-in-the-loop system.

4.2.2. Adversarial Examples (EP_AE)

Adversarial examples are test data generated with small, even imperceptible, perturbations on the original test inputs, which cause the network under test to misclassify them [18,28,74–76]. A neural network is vulnerable to adversarial perturbation, and adversarial examples are regarded as effective means to attack network. Therefore, debuggers can detect erroneous behaviors of AI-in-the-loop system by adversarial examples and enhance the robustness by re-training the AI-in-the-loop system against adversarial examples. The following techniques are implemented based on the sub-extension point EP_AE.

- L-BFGS. The *box-constrained L-BFGS* [18] is proposed to generate test data to solve the following minimization problem under the condition of $f(x^{'}) = t$. Equation (7) hopes to minimize the distance between x and $x^{'}$, $L_2 = \| x - x^{'} \|_2^2$, and the loss function $loss_{f,t}(x^{'})$ for the generated test data $x^{'}$ also labeled as t.

$$\text{Minimization } c \cdot \| x - x^{'} \|_2^2 + loss_{f,t}(x^{'}) \tag{7}$$

- FGSM. To generate adversarial examples in a quick way, Goodfellow et al. [23] presented the *Fast Gradient Sign Method* (FGSM) based on the norm L_∞. As shown in Equation (8), the adversarial example $x^{'}$ depends on the single-step ϵ and the gradient sign of loss function $loss_{f,t}(x)$, which determines the direction that increases the probability of the targeted class t.

$$x^{'} = x - \epsilon \cdot sign(\nabla loss_{f,t}(x)) \tag{8}$$

- BIM. To improve the accuracy of adversarial examples, Kurakin et al. [24] proposed the *Basic Iterative Method* (BIM) by replacing single-step ϵ in FGSM with multiple smaller steps α and minimizing the loss function for directing at the targeted label t. Equation (9) indicates that the adversarial example generated from the ith step depends on that from the last step iteratively.

$$x_i^{'} = x_{i-1}^{'} - clip_{\epsilon}(\alpha \cdot sign(\nabla loss_{f,t}(x_{i-1}^{'}))), x_0^{'} = 0 \quad (9)$$

- ILCM. Kurakin [25] further proposed the *Iterative Least-likely Class Method* (ILCM) by taking the target label with the least likelihood that is the most difficult to attack rather than the label with the most possibility in BIM.
- JSMA. Papernot et al. [26] proposed the *Jacobian-based Saliency Map Attack* (JSMA) to obtain a saliency map, which indicates the influence of each pixel on target label classification. Therefore, adversarial example can be generated with small perturbation by changing pixels with the greatest influence.
- One pixel attack. The one pixel attack [77] is presented extremely to generate adversarial image by changing only one pixel of the seeded image.
- C&W attack. To get an adversarial image with less distortion on the seeded image, Carlini and Wagner [78] proposed applying three distance metrics—L_0, L_2, and L_∞ norms—to have a target attack on neural network.
- Universal perturbation. The *universal perturbation* [79–81] is proposed for the misclassification of entire test dataset instead of misclassifying one specific test case. In other words, the adversarial examples are generated from the same perturbation and recognized as the same target label.

4.2.3. Generative Adversarial Examples (EP_GAE)

This sub-extension point EP_GAE is provided to allow researchers to generate test data with the generative adversarial nets, which aims at delivering data that cannot be distinguished from the training data or generator.

- Generative adversarial nets. The Generative Adversarial Nets (GAN) [19], including a generative model G and a discriminative model D, are proposed to generate samples by learning and try to fool the discriminative model. The purpose of generative model is producing samples by learning the training data distribution that causes the discriminative model making a mistake. The discriminative model must determine whether the input sample is from training data or generative model G.

4.2.4. Metamorphic Testing Based Strategy (EP_MT)

Metamorphic testing (MT) [82–84] is an approach to alleviate the test oracle problem by checking whether the certain property of program is satisfied instead of comparing the expected output with the actual output since some expected outputs are expensive to compute. Such a property is referred to as the *metamorphic relation* (MR) of the program function. MT is also used as a strategy of test case generation since it generates follow-up test cases according to source test cases and related MRs. Therefore, metamorphic testing based strategy is regarded as a sub-extension point in ExtendAIST, namely EP_MT, to generate test data for AIS. The existing applications of MT on AIS are as follows.

- Classifier testing. MT has already been applied to generate follow-up test cases for machine learning classifier testing by employing the MRs of permutation, addition, duplication, and removal of attributes, samples, and labels [20,21].
- DeepTest. To test DNN-based self-driving system, DeepTest [14] takes the property that the output steering angle should keep unchanged under different real-world driving environments as the metamorphic relation to generate new test cases and determine whether the self-driving system satisfies the MR.

- Effect of noise outside ROI. Zhou et al. [22] investigated the effect of noise outside of the drivable area on the obstacle perception inside of the region of interest (ROI). The MR of obstacle perception system is that the noise in the region outside ROI would not cause the obstacles inside ROI undetectable.

4.2.5. Concolic Testing Based Strategy (EP_CTS)

Concolic testing is an integration of concrete execution and symbolic execution [85]. The program is executed with some concrete input values, and then solves the symbolic constraints collected for each conditional statement to cover other execution paths which are uncovered by concrete inputs. New test case variants from concrete inputs are generated during the constraint solving procedure. Thus, concolic testing based strategy is regarded as one of the sub-extension points, namely EP_CTS, to generate test data by integrating concrete execution and symbolic execution.

- DeepConcolic. The DeepConcolic [27] leverages concolic testing to generate adversarial examples with high coverage. Given a DNN under test, a set of coverage requirements $\Re$, an unsatisfied requirement r, and the initial test suite T, in the phase of *concrete execution*, a test input $t \in T$ is identified to satisfy requirement r. Then, in the phase of *symbolic execution*, a new test input $t^{'}$ close to one of the test input t from T, is generated to satisfy requirement r and distance constraint $\| t - t^{'} \|_{\infty} \leqslant d$. Then $t^{'}$ is added to the test suite T. $t^{'}$ is repeatedly generated until all requirements are satisfied or no more requirements in $\Re$ can be satisfied.

4.3. Testing Technique (EP_TT)

4.3.1. Description

After generating test data, the ExtendAIST selects an effective testing technique to detect erroneous behaviors in the AI-in-the-loop system. The application of machine learning on embedded systems leads to challenges on testing the safety, correctness, and robustness of AI-in-the-loop systems [2–6,10–12,33,34]. Researchers have investigated various testing techniques to test the properties of AI-in-the-loop system, including adversarial attack, mutation testing, metamorphic testing, and test prioritization techniques.

The ExtendAIST provides the testing technique as an extension point EP_TT, and the four techniques mentioned above as sub-extension points named EP_AE, EP_MUT, EP_MT (same as the point EP_MT in Section 4.2.4), and EP_TP, respectively. Researchers consequently can apply the existing techniques to reveal error behaviors in system or extend new testing techniques.

4.3.2. Adversarial Attack (EP_AE)

A neural network is vulnerable to imperceptible perturbations [18]. In other words, the adversarial examples are misclassified with high confidence by a trained neural network. Therefore, adversarial attack can be regarded as a powerful method to detect defects in AI-in-the-loop systems [86]. The existing techniques of adversarial attack are described in Section 4.2.2, thus we introduce the categories of adversarial attack with regard to the output label and distance metric in this section.

$$L_p = (\Sigma_{i=1}^{n} |x_i - x_i^{'}|_p)^{\frac{1}{p}} \tag{10}$$

$$L_0 = \Sigma_{i=1}^{n} |x_i - x_i^{'}|^0 \tag{11}$$

$$L_2 = \sqrt{\Sigma_{i=1}^{n} |x_i - x_i^{'}|^2} \tag{12}$$

$$L_\infty = \max(|x_1 - x_1^{'}|, |x_2 - x_2^{'}|, \cdots, |x_n - x_n^{'}|) \tag{13}$$

- Targeted and Non-Targeted Adversarial Attack. According to the classification results for individual adversarial example, adversarial attack is divided into two categories: *targeted* and

non-targeted adversarial attack [28,29]. Targeted attack indicates that the input adversarial example is misclassified as a specific label, while non-targeted attack indicates that the input adversarial example is assigned as any label not equal to the correct one.

- L_0-norm attack. According to the definition of L_p-norm in Equation (10), L_0 (Equation (11)) indicates the number of changed features in the original input. As a result, L_0-norm attack is used to minimize the number of features perturbed and generate new feature data against the target label.
- L_2-norm attack. L_2 (Equation (12)) equals the Euclidean distance between original data and adversarial example. A lower L_2 indicates a smaller perturbation between the individual feature data and a higher similarity between original data and adversarial example. Hence, L_2 is also used to solve the problem of overfitting in adversarial example.
- L_∞-norm attack. L_∞ (Equation (13)) represents the maximum difference among all feature data, which is utilized to control the maximum perturbation between original data and adversarial example.

4.3.3. Mutation Testing (EP_MUT)

Mutation testing is a method to measure test adequacy of a test suite by injecting faults into the original program under test, namely mutants [87]. The ratio of the number of mutants detected and the total number of mutants is referred to as the mutation score. The higher the mutation score is, the stronger the fault detectability of the test suite is. The existing techniques based on this sub-extension point EP_MUT are as follows.

- MuNN. MuNN [30] proposes five kinds of mutation operators according to the structure of neural network, including deleting neurons in the input layer and hidden layers, changing the bias, weights, and activation functions. A mutant neural network is said to be killed once its output is distinct from the output of the original network. More mutant networks being killed indicates a powerful test suite for DNN testing.
- DeepMutation. Since a trained model is obtained from the training program and training data, Ma et al. [31] proposed a further study on mutation testing of AI module from two perspectives: (1) generating mutant trained models based on the *source-level* mutation operators on the training data or training program; and (2) generating mutant trained models directly based on the *model-level* mutation operators on the original trained model, similar to MuNN.

4.3.4. Metamorphic Testing (EP_MT)

Metamorphic testing determines the correctness of software under test by checking whether the related metamorphic relations are satisfied or not. Thus, it is a fundamental activity identifying diverse metamorphic relations to evaluate their capabilities of fault detection and the quality of software. Since Section 4.2.4 has introduced the existing techniques in point EP_MT, we briefly introduce the DeepTest [14] to show the mechanism of metamorphic testing on AI-in-the-loop system.

- DeepTest. Nine transformation based MRs have been proposed in DeepTest, namely changing brightness, contrast, translation, scaling, horizontal shearing, rotation, blurring, fog, and rain on the images, to test the robustness of autonomous vehicles. Take images from camera as the source images, and create the follow-up images by one or more transformation MRs on source images. The DNN under test takes source and follow-up images as inputs and outputs the source and follow-up steering angles under different real-world weather conditions, namely $\theta_s = \{\theta_s^1, \theta_s^2, \cdots, \theta_s^n\}$, $\theta_f = \{\theta_f^1, \theta_f^2, \cdots, \theta_f^n\}$. Strictly speaking, the steering angles should keep unchanged under these transformations. That is, $\theta_s = \theta_f$. However, a small variation of steering angles, $\hat{\theta} = \{\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_n\}$, in real-world driving environment would not affect the driving

behaviors. Thus, the variations within the error ranges are allowed, as shown in Equation (14), in which $MSE_{orig} = \frac{1}{n}\Sigma_{i=1}^{n}(\hat{\theta}_i - \theta_s^i)^2$.

$$(\hat{\theta}_i - \theta_f^i)^2 \leqslant \lambda MSE_{orig} \tag{14}$$

4.3.5. Test Prioritization (EP_TP)

The general procedure of testing DNN-based system is executing DNN-based system against a test dataset with manual labels and inspecting whether the learned label and manual label are identical, during which labeling each test case before testing is a labor-intensive activity. The sub-extension point of test prioritization EP_TP converts the problem of misclassification to the problem of impurity of test dataset, which can determine whether the input test case is misclassified by its impurity without the need of labels.

- DeepGini [32]. Take binary classification as example; given a test data t and the output feature vector $B = \{c_1, c_2\}$, execute DNN to compute the probability of each feature for t. If the probability classified as c_1 is $P_{c_1} = 100\%$, and c_2 is $P_{c_2} = 0$, then the feature vector B has the highest purity and t is more likely to be classified correctly. In contrast, if $P_{c_1} = 50\%$, $P_{c_2} = 50\%$, then B has the lowest purity, and t is more likely to be misclassified. $\xi(t)$ is defined as the metric of impurity and the likelihood of t being misclassified. As shown in Equation (15), $p_{t,i}$ is the probability of test case t being classified as class i. A lower $p_{t,i}^2$ indicates a higher impurity and a higher likelihood to misclassify t.

$$\xi(t) = 1 - \Sigma_{i=1}^{N} p_{t,i}^2 \tag{15}$$

4.4. Formal Verification Technique (EP_FV)

4.4.1. Description

Since formal verification is also an effective way to assure properties of AI-in-the-loop system, the formal verification is regarded as an extension point EP_FV in the ExtendAIST. There already exist various verification techniques for testing the robustness, testing adequacy, and decreasing test cost, in which the formal verification is always transformed into the following problems: satisfiability solver [33,88–91], non-linear problem [34,92–94], symbolic interval analysis [35], reachability analysis [36], and abstract interpretation [37,95]. Therefore, the ExtendAIST provides these five techniques as sub-extension points named EP_SS, EP_NLP, EP_SIA, EP_RA, and EP_AI, respectively. Based on the above, researchers can develop or extend new formal verification techniques and apply the existing techniques to assure the robustness and safety of AI-in-the-loop system.

4.4.2. Satisfiability Solver (EP_SS)

The safety verification of an image classifier can be reduced to the correct behavior satisfiability which can search for adversarial examples if misclassifications exist [33].

- Safety verification. Given a neural network N, an input image x, a region η around x with the same class, N is said to be safe for input x and η if the classification of images in η is invariant to x. That is, $N, \eta \vDash x$. In more depth, modifying the input image x with a family of manipulations Δ, N is said to be safe for input x, η, and manipulations Δ if the classification of region η keeps invariant to x under manipulations Δ, namely $N, \eta, \Delta \vDash x$. If $N, \eta, \Delta \nvDash x$, then the image classifier is vulnerable to these manipulations or adversarial perturbations.

4.4.3. Non-linear Problem (EP_NLP)

The formal verification of safety is also specified to prove the counterexample not exist for the set of variable constraints to make the property always true [34].

- Piecewise linear network verification. Since some activation functions are non-linear, the formal verification is transformed into a Mixed Integer Program (MIP) with the value of binary variables δ_a, where $\delta_a = \{0, 1\}$. The binary variable δ_a indicates the phase of activation function ReLU. If $\delta_a = 0$, the activation function is blocked and the output of related neuron will be 0; otherwise, the activation function is passed and the output of related neuron will be equal to its input value.

4.4.4. Symbolic Interval Analysis (EP_SIA)

Symbolic interval analysis [35] utilizes the symbolic interval arithmetic to obtain the accurate range of DNN's output according to the ranges of input variables.

- ReluVal. Given an input range X, subintervals of X and security property P. DNN is secure if no value in range X and its subintervals violate property P, that is, any value from range X satisfies P; otherwise DNN is insecure if there exists one adversarial example in X violating P, that is, there exists at least one subinterval containing an adversarial example to make property P unsatisfied.

4.4.5. Reachability Analysis (EP_RA)

To eliminate the limitation of network scale for formal verification, safety verification can also be transformed into reachability analysis [36].

- DeepGo. If all values in the output range, the lower and upper bounds $[l, u]$, correspond to an input in input subspace $X^{'} \subseteq [0, 1]^n$, then the network f is reachable and the reachability diameter is $D(X^{'}; f) = u(X^{'}) - l(X^{'})$. The network f is said to be safe for the input $x \in X^{'}$ if all inputs in $X^{'}$ have the same label to x, as shown in Equation (16).

$$\forall x^{'} \in X^{'} : \arg\max_j c_j(x^{'}) = \arg\max_j c_j(x) \tag{16}$$

4.4.6. Abstract Interpretation (EP_AI)

Since the scale of input data and neural network are tremendous, it is infeasible to verify whether individual input satisfies the safety properties precisely. To overcome this obstacle, an abstract domain is used to approximate the concrete domain and verify the safety properties against abstract domain directly, referred to as the abstract interpretation theory [96,97]. The following existing techniques in point EP_AI show the application of abstract interpretation on AIS testing.

- AI^2. AI^2 [37] proposes to employ the zonotope domain to represent the abstract elements in neural network and output the abstract elements in each layer. Finally, the safety of network is determined by verifying whether the label of abstract output in the output layer is consistent with that of the concrete output.
- Symbolic propagation. To improve the precision and efficiency of DNN safety verification, Yang et al. [38] proposed a symbolic propagation method based on the abstract interpretation by representing the values of neurons symbolically and propagating them from the input layer to output layer forwardly. A more precise range of output layer is computed by the interval abstract domain based on the symbolic representation of output layer.

4.5. Evaluation Dataset (EP_ED)

4.5.1. Description

Since the machine learning based AI-in-the-loop system is a data-driven system, it is essential to design datasets adaptive for diverse application areas to conduct the training, testing, and evaluation procedures. There mainly exist five kinds of datasets for different purposes: image classification, self driving, natural language processing, speech recognition, and Android malware detection.

Therefore, we take the common dataset as the super extension point EP_CD and the five kinds of datasets as the sub-extension points named EP_IC, EP_SD, EP_NALP, EP_SR, and EP_AMD. Based on the above, researchers can design more datasets for diverse scenarios and different learning, testing, and evaluation tasks. The existing techniques and common datasets in each sub-extension point are displayed as follows.

4.5.2. Image Classification (EP_IC)

The datasets for image classification are designed for object detection and classification in many application areas. We take the dataset of image classification as a sub-extension point, EP_IC, to identify more samples for training and testing models of image classification.

- MNIST. MNIST [39,40] is a dataset for recognizing handwritten digits (0–9) including 70,000 images originating from the NIST database [98]. The MNIST dataset is composed of a training dataset with 60,000 images and a test dataset with 10,000 images, each of which contains half clear digits written by government staff and half blurred digits written by students.
- EMNIST. EMNIST [41,42] is an extension dataset of MNIST for identifying handwritten digits (0–9) and letters (a–z and A–Z). Therefore, there are totally 62 classes in EMNIST, including 10 classes of digits and 52 classes of letters. However, some uppercases and lowercases cannot be distinguished easily (e.g., C and c and K and k), then the letters are merged into 37 classes.
- Fashion-MNIST. Fashion-MNIST [43,44] is a dataset with the extremely same format and size of MNIST for identifying 10 classes of fashion products, such as T-shirt, trouser, pullover, dress, coat, sandals, shirt, sneaker, bag, and ankle boots.
- ImageNet. ImageNet [45,46] is an image dataset describing the synsets in the WordNet hierarchy with on average 1000 images.
- CIFAR-10/CIFAR-100. CIFAR-10 and CIFAR-100 [47] are labeled image datasets consisting of 60,000 32 $\times$ 32 color images, 50,000 of which are training images and 10,000 are test images. CIFAR-10 is divided into 10 classes, and there are 5000 training images and 1000 test images for each class. CIFAR-100 is divided into 100 fine classes, and each class contains 500 training images and 100 test images.

4.5.3. Self Driving (EP_SD)

Self driving is a typical autonomous system inspecting the performance of AI-in-the-loop system. Hence, the dataset for self driving is also regarded as a sub-extension point, EP_SD, for tasks such as obstacle detection and tracking and traffic signal recognition.

- Udacity Challenge. Udacity Challenge dataset [48] is a set of images for training and testing the objects detection models in the Udacity Challenge competition to measure the performance of each participating model in terms of detection capability and classification precision. The Udacity Challenge dataset contains two parts according to its classification. One consists of 9420 images in three classes: car, truck, and pedestrian; and the other consists of 15,000 images in five classes: car, truck, pedestrian, traffic light, and bycicle.
- MSCOCO 2015. MSCOCO 2015 [49,50] is a dataset gathered from the daily scenes of common objects in the real-world environment, which aims at object recognition and localization.

This dataset contains 165,482 training images, 81,208 verification images and 81,434 testing images in 91 classes, such as animal, vegetable, and human.

- KITTI. KITTI dataset [51–53] contains the realistic images captured from the real-world driving environments such as mid-size city, rural area, and highway. All these images are divided into five classes: road, city, residence, campus, and human. Evaluation tasks include stereo, optical flow, visual odometry, object detection, and tracking.
- Baidu Apollo. The Baidu Apollo [54] open platform provides the massive annotation data and simulation for training and testing autonomous driving tasks, such as obstacle detection and classification, traffic light detection, road hackers, obstacle trajectory prediction, and scene analysis under different street views and vehicle movement images.

4.5.4. Natural Language Processing (EP_NALP)

Automatic natural language processing with machine learning releases researchers from the labor-intensive work of text processing. It is essential to design datasets for training and testing systems with the ability of natural language processing, which guarantees the precision of these systems. Therefore, we take the dataset for natural language processing as a sub-extension point, EP_NALP, to design more processing method for various processing tasks.

- Enron. Enron [55] is an email dataset collected from 150 senior managers of Enron, which contains about 500,000 real email messages. All emails from one manager are organized into a folder, which contains the information of message-ID, date, sender, recipient, and messages.
- bAbI. bAbI [56] is a dataset from Facebook AI Research with 20 toy question-answering tasks. Each task contains 1000 training examples and 1000 test examples, which aims at reasoning or answering questions after training.
- Common Crawl. The Common Crawl corpus [57] contains petabytes of data including raw web page data, metadata extracts, and text extracts by web crawling. This corpus provides web-scaled data for developing or evaluating natural language processing algorithms empirically.
- SNLI. The SNLI corpus [58,59] contains 570,000 handwritten English sentence pairs with the manual labels entailment, contradiction, and neutral. This dataset is used to evaluate models of natural language inference.

4.5.5. Speech Recognition (EP_SR)

This sub-extension point EP_SR allows designing datasets for learning speech recognition, with which the information omitted by human can be detected and represented by machine.

- Speech Commands. Speech Commands [60,61] is an audio dataset for detecting single keyword from a set of spoken words. This dataset focuses on the audio of on-device trigger phrases and contains 105,829 utterances of 35 words in format of WAVE files. All these utterance were recorded from 2618 speakers, and each utterance is one second or less.
- Free Spoken Digit Dataset. FSDD [62] is a dataset of 2000 audio recordings of spoken English digits in format of WAVE files. These speeches were recorded by four speakers with 50 recordings of each digit per speaker.
- Million Song Dataset. This dataset [63,64] provides the feature analysis and metadata of one million popular songs, and the audio can be fetched by the code in this dataset.
- LibriSpeech. LibriSpeech [65,66] is a collection of about 1000 hours of read English speech with the labels of sentence level. Therefore, this dataset is suitable for full speech recognition.

4.5.6. Android Malware Detection (EP_AMD)

The sub-extension point EP_AMD allows researchers to design more malware datasets or enlarge the scales of the following existing datasets, detect the Android malware, and ensure the security of smart phones with the rapid development.

- Drebin. The Drebin dataset [67–69] contains 5560 Apps from 179 different malware families, which is a lightweight dataset to identify Android malware.
- Genome. The Genome dataset [70,71] has collected more than 1200 malwares in different aspects, such as the installation methods, activation mechanism, and the nature of carried malicious payloads.
- VirusTotal. VirusTotal [72] is a dataset of mobile Apps samples for analyzing suspicious files and URLs to detect types of malware including viruses, worms, and trojans.
- Contagio. The Contagio [73] is a platform for collecting the most recent malware, thus it can be used to analyze and help detect unknown malware.

5. Discussion

Although we have investigated the possible extension, sub-extension points, and existing techniques of AI-in-the-loop system testing, the AI techniques update significantly with the increasing development of application areas and computation ability, which lead to the crucial fact that even the points provided in ExtendAIST are not adequate to develop new techniques. This case will be the main threat to the validity of our proposed ExtendAIST framework.

6. Conclusions

The AI-in-the-loop system is defined in this paper for discussing the main differences between testing ordinary system and AI-in-the-loop system in terms of testing coverage, testing data, testing property, and testing approach for unit testing and system testing, respectively. In this case, we find that it is essential to design or extend new testing techniques based the existing techniques of AI-in-the-loop system testing. Therefore, we propose an extendable framework ExtendAIST to explore the design space of testing AI-in-the-loop system. The extendable framework ExtendAIST proposed in our paper provides five extension points, 19 sub-extension points and corresponding testing requirements. Each extension point involves one step in AIS testing procedure, and the sub-extension points for individual extension point are the implemented approaches that could be extended further. Additionally, ExtendAIST also provides some existing techniques in each sub-extension point which can be reused directly or extended new techniques. According to the description of each point, we find that there exist some opportunities for AI-in-the-loop system testing in the following extension points.

- EP_TCM. Training and testing procedure should be enhanced to adapt to different systems with more complex networks. Thus, the extension point EP_TCM provides the first opportunity to increase the testing coverage on different scaled AI-in-the-loop systems with regard to the traditional software testing coverage metrics.
- EP_ED. Since enormous test data involve powerful computing ability and expensive computing cost, the extension point EP_ED provides the second opportunity to design a dataset with relatively smaller size for various application areas.
- EP_TT and EP_FV. AI-in-the-loop system testing techniques currently focus on assuring the system correctness, safety, and robustness. This is the third opportunity provided by EP_TT and EP_FV to design new testing approach to test more properties for AI-in-the-loop system including efficiency, interpretability, and stiffness.

Author Contributions: Conceptualization, T.W. and Y.D.; Methodology, T.W.; Project administration, Y.D. and Y.Z.; Supervision, Y.D.; Validation, T.W., Y.D., Y.Z., and A.S.; Visualization, T.W.; Writing—original draft, T.W.; and

Writing—review and editing, T.W., Y.D., Y.Z., and A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the State Grid Technology Major Project of China under grant No. 2019GW-12 (Security Protection Technology of Embedded Components and Control Units in Power System Terminal).

Acknowledgments: Special thanks to my colleague Qianwen Gou for her help in the background of security and stability intelligent control unit in power system terminal.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Badue, C.; Guidolini, R.; Carneiro, R.V.; Azevedo, P.; Cardoso, V.B.; Forechi, A.; Jesus, L.F.R.; Berriel, R.F.; Paixão, T.M.; Mutz, F.; et al. Self-driving cars: A survey. *arXiv* **2019**, arXiv:1901.04407.
2. Hains, G.; Jakobsson, A.; Khmelevsky, Y. Towards formal methods and software engineering for deep learning: security, safety and productivity for dl systems development. In Proceedings of the 2018 Annual IEEE International Systems Conference (SysCon), Vancouver, BC, Canada, 23–26 April 2018; pp. 1–5.
3. Masuda, S.; Ono, K.; Yasue, T.; Hosokawa, N. A Survey of Software Quality for Machine Learning Applications. In Proceedings of the 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Xi'an, China, 9–13 April 2018; pp. 279–284.
4. Braiek, H.B.; Khomh, F. On testing machine learning programs. *arXiv* **2018**, arXiv:1812.02257.
5. Ma, L.; Juefei-Xu, F.; Xue, M.; Hu, Q.; Chen, S.; Li, B.; Liu, Y.; Zhao, J.; Yin, J.; See, S. Secure Deep Learning Engineering: A Software Quality Assurance Perspective. *arXiv* **2018**, arXiv:1810.04538.
6. Huang, X.; Kroening, D.; Kwiatkowska, M.; Ruan, W.; Sun, Y.; Thamo, E.; Wu, M.; Yi, X. Safety and Trustworthiness of Deep Neural Networks: A Survey. *arXiv* **2018**, arXiv:1812.08342.
7. Zhang, J.M.; Harman, M.; Ma, L.; Liu, Y. Machine Learning Testing: Survey, Landscapes and Horizons. *arXiv* **2019**, arXiv:1906.10742.
8. Martinez, M.; Monperrus, M. Astor: Exploring the design space of generate-and-validate program repair beyond GenProg. *J. Syst. Softw.* **2019**, *151*, 65–80. [CrossRef]
9. Barr, E.T.; Harman, M.; Mcminn, P.; Shahbaz, M.; Yoo, S. The Oracle Problem in Software Testing: A Survey. *IEEE Trans. Softw. Eng.* **2015**, *41*, 507–525. [CrossRef]
10. Koopman, P.; Wagner, M. Challenges in autonomous vehicle testing and validation. *SAE Int. J. Transp. Saf.* **2016**, *4*, 15–24. [CrossRef]
11. Goodfellow, I.; Papernot, N. *The Challenge of Verification and Testing of Machine Learning*. Available online: http://www.cleverhans.io/security/privacy/ml/2017/06/14/verification.html (accessed on 15 May 2019).
12. Koopman, P.; Wagner, M. Autonomous vehicle safety: An interdisciplinary challenge. *IEEE Intell. Transp. Syst. Mag.* **2017**, *9*, 90–96. [CrossRef]
13. Pei, K.; Cao, Y.; Yang, J.; Jana, S. Deepxplore: Automated whitebox testing of deep learning systems. In Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, 28–31 October 2017; pp. 1–18.
14. Tian, Y.; Pei, K.; Jana, S.; Ray, B. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 27 May–3 June 2018; pp. 303–314.
15. Ma, L.; Juefei-Xu, F.; Zhang, F.; Sun, J.; Xue, M.; Li, B.; Chen, C.; Su, T.; Li, L.; Liu, Y.; et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018; pp. 120–131.
16. Sun, Y.; Huang, X.; Kroening, D. Testing deep neural networks. *arXiv* **2018**, arXiv:1803.04792.
17. Fort, S.; Nowak, P.K.; Narayanan, S. Stiffness: A new perspective on generalization in neural networks. *arXiv* **2019**, arXiv:1901.09491.
18. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. *arXiv* **2013**, arXiv:1312.6199.

19. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. In Proceedings of the Annual Conference on Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 2672–2680.
20. Xie, X.; Ho, J.; Murphy, C.; Kaiser, G.; Xu, B.; Chen, T.Y. Application of metamorphic testing to supervised classifiers. In Proceedings of the 2009 Ninth International Conference on Quality Software, Jeju, Korea, 24–25 August 2009; pp. 135–144.
21. Xie, X.; Ho, J.W.; Murphy, C.; Kaiser, G.; Xu, B.; Chen, T.Y. Testing and validating machine learning classifiers by metamorphic testing. *J. Syst. Softw.* **2011**, *84*, 544–558. [CrossRef] [PubMed]
22. Zhou, Z.Q.; Sun, L. Metamorphic testing of driverless cars. *Commun. ACM* **2019**, *62*, 61–67. [CrossRef]
23. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. *arXiv* **2014**, arXiv:1412.6572.
24. Kurakin, A.; Goodfellow, I.; Bengio, S. Adversarial examples in the physical world. *arXiv* **2016**, arXiv:1607.02533.
25. Kurakin, A.; Goodfellow, I.; Bengio, S. Adversarial machine learning at scale. *arXiv* **2016**, arXiv:1611.01236.
26. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The limitations of deep learning in adversarial settings. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbrucken, Germany, 21–24 March 2016; pp. 372–387.
27. Sun, Y.; Wu, M.; Ruan, W.; Huang, X.; Kwiatkowska, M.; Kroening, D. Concolic testing for deep neural networks. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018; pp. 109–119.
28. Akhtar, N.; Mian, A. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access* **2018**, *6*, 14410–14430. [CrossRef]
29. Poursaeed, O.; Katsman, I.; Gao, B.; Belongie, S. Generative adversarial perturbations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 16–20 June 2018; pp. 4422–4431.
30. Shen, W.; Wan, J.; Chen, Z. MuNN: Mutation Analysis of Neural Networks. In Proceedings of the 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Lisbon, Portugal, 16–20 July 2018; pp. 108–115.
31. Ma, L.; Zhang, F.; Sun, J.; Xue, M.; Li, B.; Juefei-Xu, F.; Xie, C.; Li, L.; Liu, Y.; Zhao, J.; et al. Deepmutation: Mutation testing of deep learning systems. In Proceedings of the 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), Memphis, TN, USA, 15–18 October 2018; pp. 100–111.
32. Shi, Q.; Wan, J.; Feng, Y.; Fang, C.; Chen, Z. DeepGini: Prioritizing Massive Tests to Reduce Labeling Cost. *arXiv* **2019**, arXiv:1903.00661.
33. Huang, X.; Kwiatkowska, M.; Wang, S.; Wu, M. Safety verification of deep neural networks. In Proceedings of the International Conference on Computer Aided Verification, Heidelberg, Germany, 24–28 July 2017; pp. 3–29.
34. Bunel, R.; Turkaslan, I.; Torr, P.H.; Kohli, P.; Kumar, M.P. A Unified View of Piecewise Linear Neural Network Verification. *arXiv* **2017**, arXiv:1711.00455.
35. Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; Jana, S. Formal security analysis of neural networks using symbolic intervals. In Proceedings of the 27th {USENIX} Security Symposium ({USENIX} Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1599–1614.
36. Ruan, W.; Huang, X.; Kwiatkowska, M. Reachability analysis of deep neural networks with provable guarantees. *arXiv* **2018**, arXiv:1805.02242.
37. Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; Vechev, M. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–23 May 2018; pp. 3–18.
38. Yang, P.; Liu, J.; Li, J.; Chen, L.; Huang, X. Analyzing Deep Neural Networks with Symbolic Propagation: Towards Higher Precision and Faster Verification. *arXiv* **2019**, arXiv:1902.09866.
39. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
40. LeCun, Y.; Cortes, C.; Burges, C.J. The MNIST Database of Handwritten Digits. 1998. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 15 May 2019).

41. Cohen, G.; Afshar, S.; Tapson, J.; van Schaik, A. EMNIST: An extension of MNIST to handwritten letters. *arXiv* **2017**, arXiv:1702.05373.
42. EMNIST. 2017. Available online: https://www.westernsydney.edu.au/bens/home/reproducible_research/emnist (accessed on 15 May 2019).
43. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.
44. Fashion-MNIST. 2017. Available online: https://github.com/zalandoresearch/fashion-mnist (accessed on 15 May 2019).
45. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255.
46. ImageNet. 2009. Available online: http://www.image-net.org/ (accessed on 15 May 2019).
47. CIFAR. 2014. Available online: http://www.cs.toronto.edu/~kriz/cifar.html (accessed on 15 May 2019).
48. Udacity-Challenge 2016. Using Deep Learning to Predict Steering Angles. 2016. Available online: https://github.com/udacity/self-driving-car (accessed on 15 May 2019).
49. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; pp. 740–755.
50. MSCOCO. 2015. Available online: http://cocodataset.org/ (accessed on 15 May 2019).
51. KITTI. 2015. Available online: http://www.cvlibs.net/datasets/kitti/index.php (accessed on 15 May 2019).
52. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets robotics: The KITTI dataset. *Int. J. Robot. Res.* **2013**, *32*, 1231–1237. [CrossRef]
53. Geiger, A.; Lenz, P.; Urtasun, R. Are we ready for autonomous driving? The kitti vision benchmark suite. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 16–21 June 2012; pp. 3354–3361.
54. Baidu Apollo. 2017. Available online: http://apolloscape.auto/ (accessed on 15 May 2019).
55. Enron. 2015. Available online: https://www.cs.cmu.edu/~./enron/ (accessed on 15 May 2019).
56. The bAbI Dataset. Available online: https://research.facebook.com/research/babi/ (accessed on 15 May 2019).
57. Common Crawl. Available online: http://commoncrawl.org/the-data/ (accessed on 15 May 2019).
58. Bowman, S.R.; Angeli, G.; Potts, C.; Manning, C.D. A large annotated corpus for learning natural language inference. *arXiv* **2015**, arXiv:1508.05326.
59. Stanford Natural Language Inference. Available online: https://nlp.stanford.edu/projects/snli/ (accessed on 15 May 2019).
60. Warden, P. Speech Commands: A dataset for Limited-Vocabulary speech recognition. *arXiv* **2018**, arXiv:1804.03209.
61. Speech Commands. 2017. Available online: https://download.tensorflow.org/data/speech_commands_v0.01.tar.gz (accessed on 15 May 2019).
62. Free Spoken Digit Dataset. Available online: https://github.com/Jakobovski/free-spoken-digit-dataset (accessed on 15 May 2019).
63. Million Song Dataset. Available online: http://millionsongdataset.com/ (accessed on 15 May 2019).
64. Bertin-Mahieux, T.; Ellis, D.P.W.; Whitman, B.; Lamere, P. The Million Song Dataset. In Proceedings of the International Society for Music Information Retrieval Conference, Porto, Portugal, 8–12 October 2012.
65. Panayotov, V.; Chen, G.; Povey, D.; Khudanpur, S. Librispeech: An ASR corpus based on public domain audio books. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, Queensland, Australia, 19–24 April 2015.
66. LibriSpeech. Available online: http://www.openslr.org/12/ (accessed on 15 May 2019).
67. Drebin. Available online: https://www.sec.cs.tu-bs.de/~danarp/drebin/index.html (accessed on 15 May 2019).
68. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. *Ndss* **2014**, *14*, 23–26.

69. Spreitzenbarth, M.; Freiling, F.; Echtler, F.; Schreck, T.; Hoffmann, J. Mobile-sandbox: Having a deeper look into android applications. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, Coimbra, Portugal, 18–22 March 2013; pp. 1808–1815.
70. Zhou, Y.; Jiang, X. Dissecting Android Malware: Characterization and Evolution. In Proceedings of the 33rd IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 20–23 May 2012.
71. Android Malware Genome Project. Available online: http://www.malgenomeproject.org/ (accessed on 15 May 2019).
72. VirusTotal. Available online: https://www.virustotal.com/ (accessed on 15 May 2019).
73. Contagio Malware Dump. Available online: http://contagiodump.blogspot.com/ (accessed on 15 May 2019).
74. Yuan, X.; He, P.; Zhu, Q.; Li, X. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 2805–2824. [CrossRef]
75. Biggio, B.; Roli, F. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognit.* **2018**, *84*, 317–331. [CrossRef]
76. Zhang, S.; Zuo, X.; Liu, J. The problem of the adversarial examples in deep learning. *J. Comput.* **2018**, 41, 1–21. [CrossRef]
77. Su, J.; Vargas, D.V.; Sakurai, K. One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.* **2019**, *23*, 828–841. [CrossRef]
78. Carlini, N.; Wagner, D. Towards evaluating the robustness of neural networks. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–24 May 2017; pp. 39–57.
79. Moosavi-Dezfooli, S.M.; Fawzi, A.; Fawzi, O.; Frossard, P. Universal adversarial perturbations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1765–1773.
80. Sarkar, S.; Bansal, A.; Mahbub, U.; Chellappa, R. UPSET and ANGRI: Breaking high performance image classifiers. *arXiv* **2017**, arXiv:1707.01159.
81. Mopuri, K.R.; Garg, U.; Venkatesh, B.R. Fast Feature Fool: A data independent approach to universal adversarial perturbations. *arXiv* **2017**, arXiv:1707.05572.
82. Chen, T.Y.; Cheung, S.C.; Yiu, S.M. *Metamorphic Testing: A New Approach for Generating Next Test Cases*; Technical Report; Technical Report HKUST-CS98-01; Department of Computer Science: Hong Kong, China, 1998.
83. Segura, S.; Fraser, G.; Sanchez, A.B.; Ruiz-Cortés, A. A survey on metamorphic testing. *IEEE Trans. Softw. Eng.* **2016**, *42*, 805–824. [CrossRef]
84. Chen, T.Y.; Kuo, F.C.; Liu, H.; Poon, P.L.; Towey, D.; Tse, T.; Zhou, Z.Q. Metamorphic testing: A review of challenges and opportunities. *ACM Comput. Surv. (CSUR)* **2018**, *51*, 4. [CrossRef]
85. Cadar, C.; Sen, K. Symbolic execution for software testing: three decades later. *Commun. ACM* **2013**, *56*, 82–90. [CrossRef]
86. Yi, P.; Wang, K.; Huang, C.; Gu, S.; Zou, F.; Li, J. Adversarial attacks in artificial intelligence: A survey. *J. Shanghai Jiao Tong Univ.* **2018**, 52, 1298–1306.
87. Jia, Y.; Harman, M. An analysis and survey of the development of mutation testing. *IEEE Trans. Softw. Eng.* **2010**, *37*, 649–678. [CrossRef]
88. Katz, G.; Barrett, C.; Dill, D.L.; Julian, K.; Kochenderfer, M.J. Reluplex: An efficient SMT solver for verifying deep neural networks. In Proceedings of the International Conference on Computer Aided Verification, Heidelberg, Germany, 24–28 July 2017; pp. 97–117.
89. Pulina, L.; Tacchella, A. An abstraction-refinement approach to verification of artificial neural networks. In Proceedings of the International Conference on Computer Aided Verification, Edinburgh, UK, 15–19 July 2010; pp. 243–257.
90. Narodytska, N.; Kasiviswanathan, S.; Ryzhyk, L.; Sagiv, M.; Walsh, T. Verifying properties of binarized deep neural networks. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
91. Cheng, C.H.; Nührenberg, G.; Huang, C.H.; Ruess, H. Verification of Binarized Neural Networks via Inter-Neuron Factoring. *arXiv* **2017**, arXiv:1710.03107.
92. Lomuscio, A.; Maganti, L. An approach to reachability analysis for feed-forward relu neural networks. *arXiv* **2017**, arXiv:1706.07351.

93. Cheng, C.H.; Nührenberg, G.; Ruess, H. Maximum resilience of artificial neural networks. In Proceedings of the International Symposium on Automated Technology for Verification and Analysis, Pune, India, 3–6 October 2017; pp. 251–268.
94. Xiang, W.; Tran, H.D.; Johnson, T.T. Output reachable set estimation and verification for multilayer neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 5777–5783. [CrossRef] [PubMed]
95. Mirman, M.; Gehr, T.; Vechev, M. Differentiable abstract interpretation for provably robust neural networks. In Proceedings of the International Conference on Machine Learning, Jinan, China, 26–28 May 2018; pp. 3575–3583.
96. Cousot, P.; Cousot, R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles Of Programming Languages, Los Angeles, CA, USA, 17–19 January 1977; pp. 238–252.
97. Cousot, P.; Cousot, R. Abstract interpretation frameworks. *J. Logic Comput.* **1992**, *2*, 511–547. [CrossRef]
98. Grother, P.J. *NIST Special Database 19 Handprinted Forms and Characters Database*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 1995.