What is AI Software Testing? and Why

Jerry Gao③, Chuanqi Tao①②, Dou Jie ④, Shenqiang Lu④
① College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, P.R. China
② State Key Laboratory for Novel Software Technology, Nanjing University, P.R. China
③ San Jose State University, San Jose, USA
④ Taiyuan University of Technology, Taiyuan, P.R. China
Correspondence to: jerry.gao@sjsu.edu

Abstract— With the fast advance of artificial intelligence technology and data-driven machine learning techniques, building high-quality AI-based software in different application domains is becoming a very hot research topic in both academic and industry communities. Today, many machine learning models and artificial technologies have been developed to build smart application systems based on multimedia inputs to achieve intelligent functional features, such as recommendation, object detection, classification, and prediction, natural language processing and translation, and so on. This brings strong demand in quality validation and assurance for AI software systems. Current research work seldom discusses AI software testing questions, challenges, and validation approaches with clear quality requirements and criteria. This paper focuses on AI software quality validation, including validation focuses, features, and process, and potential testing approaches. Moreover, it presents a test process and a classification-based test modeling for AI classification function testing. Finally, it discusses the challenges, issues, and needs in AI software testing.

Keywords— AI Testing, Testing AI software, AI software quality validation.

1. INTRODUCTION

According to the report [1], the automation testing market size is expected to grow from USD 8.52 Billion in 2018 to USD 19.27 Billion by 2023, at a Compound Annual Growth Rate (CAGR) of 17.7% during the forecast period (2018–2023). With the fast advance of big data analytics and AI technologies, numerous AI software and applications have been widely accepted and used in people's daily life.

The current AI-based software and applications are developed based on state-of-the-art machine learning models and techniques through large-scale data training to implement diverse artificial intelligent features and capabilities. Current AI-based system functions and features can be classified into the following categories: a) natural language processing (NLP) capability with language understanding and translation; b) detection and recognition function, for example, human face identification, voice recognition and object detection; c) recommendation functional features in e-commerce and advertising; d) unman-controlled vehicles, robots, and UAVs; e) question and answer functions to assist users in messaging, phone calling, search, and smart home appliance control; f) object identification and classification; g) prediction and business decision-making, and so on.

In the past two years, we used our software quality testing class students to conduct testing projects for numerous mobile apps powered with machine learning capabilities using conventional software testing and tools. We have encountered many questions during software testing. The major ones are listed below.

- What is AI software testing?
- How to test AI functions in a mobile app?
- How to identify and establish well-defined test requirements for AI functions in a mobile app?
- What and where are the cost-effective testing models and methods for testing AI functions?
- What and where are the adequate quality assessment criteria for AI functions?
- How to evaluate the training and test data sets?
- Where are the automatic tools supporting AI software testing?

In addition, we also found following common facts and features of current AI mobile software apps after validating numerous ones.

1. Limited data training and validation

– Most of our tested mobile apps with AI features are built based on machine learning models and techniques, and trained and validated with limited input data sets under ad-hoc contexts.

2. Data-driven learning features

– Many mobile apps with learning features provide static and/or dynamic learning capabilities that affect the under-test software outcomes, results and actions.

3. Uncertainty in system outputs, responses, or actions

– Since existing AI-based models are dependent on statistics algorithms, this brings the outcome uncertainty of AI software. We experienced many mobile apps with AI functions generated inconsistent outcomes for the same input test data when context conditions are changed. Similarly, we also encountered accuracy issues when we changed the training test data sets and/or test data sets.

These unique AI software features causes new difficulties and challenges in quality validation, hence, performing AI quality validation becomes a critical concern and a hot research subject for current and future AI software development and engineering. Although there have been numerous published papers addressing data quality and quality assurance in the past [2] [3] [4], seldom researches focus on validation for AI software from a function and feature view. As more and more intelligent software systems are developed, there will be a strong demand in quality testing and assurance services for AI software system.

This paper is written to provide our perspective views on AI software testing and quality evaluation based on our practical experience. The paper is organized as follows. Section 2 discusses the basic concepts about AI software testing, including test focuses, features, and process. Section 3 reviews the recent work in AI-based testing and AI machine testing. Section 4 discusses AI classification function testing process and its test modeling, and presents an

innovative block-box AI classification function testing based on a new model, known as 3D Classification Decision Table. Section 5 discuss AI function quality evaluation and test coverage analysis. Later, Section 6 discusses major issues, challenges, and needs are presented, and conclusion remarks are presented in Section 7.

2. What Is AI Software Testing?

Intuitively, AI software testing refers to diverse quality testing activities for AI-based software systems using well-defined quality validation models, methods, and tools. Its major objective is to validate system functions and features developed based machine learning models, techniques and technologies. AI software testing includes the following primary goals:

- Establish AI function quality testing requirements and assessment criteria
- Detect AI function issues, limitations, and quantitative and quality problems
- Gain the quality confidence of AI functional features developed based on AI techniques and machine learning models.
- Evaluate AI system quality against well-established quality requirements and standards.

2.1. Major Testing Focuses and Scope

In the past two years, we have validated different types of AI mobile apps with diverse AI capabilities and features. Figure 1 shows the scope of AI software testing, which covers different types of artificial intelligent features and capabilities. Our students have tested numerous mobile apps powered with diverse machine learning models and AI algorithms. Here are some typical examples.

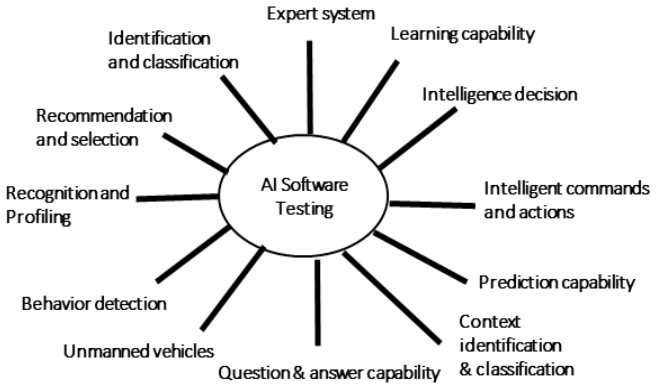

Figure 1. AI Software Testing Scope

- ***Apple Siri*** **-** It is a built-in, voice-controlled personal assistant for Apple users. The idea is that you talk to her as you would a friend and she aims to help you get things done, whether that be making a dinner reservation or sending a message.
- ***Calorie MAMA*** **–** It is a smart camera app that uses deep learning to track nutrition from food images.
- ***Seeing AI*** – It is a free app that narrates the world around you. Designed for the blind and low vision community, this ongoing research project with powered AI techniques. Its goal is to open up the visual world for users by describing nearby people, text and objects.
- ***Check My Age*** **-** It is a biometric face detection and age estimation application. It uses world best Neurotechnology face recognition algorithms to find the age from the look of the face.

The major quality validation focuses for mobile apps with AI features can be summarized below.

(a) Data quality validation:

Since AI-powered functions/features are developed and trained based on selected/developed machine learning models using large-scale training data in a sample-based training approach. Many recent machine learning project experience suggest that the quality of training data plays a critical role in AI function training and development. Hence, training data quality validation is not necessary but also critical. In our recent machine learning project and class experience, we encountered serious issues in training data quality checking for large-scale unstructured training data (such as images, audios, and videos). Three major issues are highlighted here.

Issue #1 - Domain-specific training data quality checking could be very costly and time consuming. For example, training data for medical machine learning projects require the quality validation and confirmation from medical doctors. This is not only costly but very time consuming.

Issue #2 - There is a lack of automatic unstructured data quality validation tools. Although some existing tools (such as AAAA, BBB) are available for raw data quality checking, we could not find automatic tools for validating annotated rich media training data (i.e. video, audio and images). This becomes a serious issue in training data quality validation.

Issue #3 – There is a lack of well-defined data quality evaluation models and assessment metrics for unstructured training data, including images, videos, and audios.

(b) White-box testing focuses:

In a conventional software system with AI functions/features, its program is structured with complex control structures to generate expected software behaviors and operations. A software program usually is written based on function-oriented logic algorithms, limited data parameters, and Boolean conditions for business rules and system constraints. White-box testing for a conventional software system focuses on the detection of the program errors in its structures and logics, and limited data value settings, as well as Boolean conditions and logics. The typical white-box testing goal is to achieve pre-defined program logic structure coverage, such as code coverage, branch coverage, and condition coverage.

Based on our recent teaching and research experience in AI software projects, we found that AI software development is highly based on selected machine learning models using statistic algorithms, and these models are trained using different learning approaches based on provided large-scale training and testing data. Using the current machine learning platform (such as TensorFlow), developed AI software programs usually include simple control program structures, limited business logics, basic system behavior conditions and constraints due to the nature of statistic model-based coding. This implies that the program-based white-box testing becomes a **trivial task**.

However, most provided machine models in a machine learning development platforms (or frameworks) are provided as open-source components.

The quality of AI functions is highly dependent on the selected training models and provided data quality, as well as the applied training process and methods. In our view, white-box testing for AI software must pay attentions to design training and test data to achieve certain adequate AI-model coverage.

(c) Black-box testing focuses:

Similar to conventional black-box function testing, AI function validation is performed without understanding of its programs and underlying machine learning models. Its major quality validation focus for any given AI function/feature is to assure its outcome quality in accuracy, consistency, relevancy, timeliness, correctness in responding its classified inputs and classified context conditions.

(d) System testing focuses

Like conventional software system quality testing, we need to check system QoS parameters based on well-defined quality standards and assessment criteria. The QoS parameters include system performance, reliability, scalability, availability, robustness, and security, and etc.

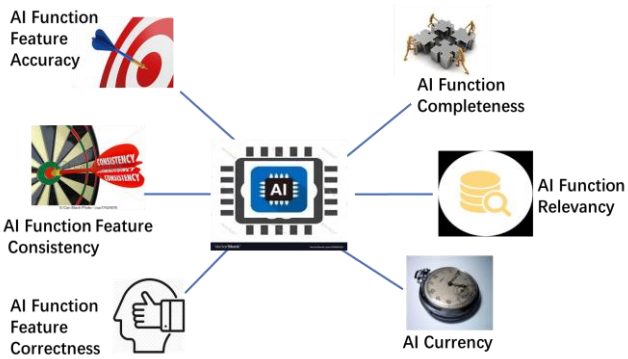

Figure 2 AI Function Quality Testing Parameters (A Black-Box View)

In addition to the system quality parameters presented above, we must pay attentions to a number of quality parameters for AI software functions. Figure 2 lists a common set. Each of them is illustrated below.

- *Function Accuracy* – This quality parameter is used to evaluate the accuracy of a system function when it is powered by machine learning models and algorithms to see if it yields accurate results for given test data sets. Many common AI functions need to validated to assure their accuracy. Typical examples include object/human/animal detection and classification, voice recognition and understanding, behavior pattern classification, and prediction functions based on historical data learning and training.

- *Function Consistency* - This quality indicator is useful to evaluate the consistency of a provided AI feature in different perspectives. Since current AI software are trained using machine learning models with statistics algorithms, this causes the inherent uncertainties of AI functional outcomes during system function validation. For example, a voice-based smart home AI software may perform differently with inconsistent outcomes in responding to the same client voice command due to context or condition changes.

- Function Correctness - This quality factor is used to evaluate the correctness of AI software features. In conventional system function testing, we validate if a system function performs correctly by generating function test cases in terms of inputs and expected outputs based on the given function requirements. In AI software testing, we found that its function correctness is highly dependent on machine learning model selection & training, and provided training data sets. For a same training data set, different AI models usually provide different quality outcomes. For the same AI model, different training and test data sets could generate different outcomes and lead to different correctness. This provides a new challenge for quality testing and assurance for AI software engineers.

- Function Currency - to measure how stale function is with respect to sources. Data is extracted from sources, then is processed and delivered to users. But source data may have changed since data extraction and users may receive stale data. The goal of function currency validation is to check if it is the same function between data extraction and data delivery and estimate the extent to which function are up-to-date. A datum value is up-to-date if it is currently at a specific point in time; and it is outdated if it is currently at a preceding time but incorrect at later time. Hence, performing a temporal function check is needed to ensure the function with just-in data is current. For a real-time AI function feature, this quality parameter must be validated to make sure a given AI function is performed as expected at different timing environment for real-time input data.

- **Function Completeness –** This quality parameter is used to evaluate the completeness of AI functions. In conventional software function testing, we need to make sure that every specified function is validated including its sub-functions. Similar to this, AI function testing also need to pay an attention in checking every specified AI function and its underlying sub-function. However, we found that many AI function must be examined for its sub-functions based on identification and classifications of its training and test data classes. For example, a dog detection and classification function for a given image must be validated by preparing test data sets covering different dog classes, diverse dog occurrences and postures, as well as various contexts and conditions.

2.2. AI Testing Approaches and Services

AI software testing could be carried out using the following approaches, shown in Figure 3.

- **Rule-based AI software testing,** in which pre-defined expert-based rules are established and used in AI test generation and validation. This approach has been reported in [X] long time ago.
- **Classification-based AI software testing,** in which classification models for inputs, contexts, and outputs and events are setup for AI software testing to assure the adequate testing coverage of diverse input data classes, classified contexts and conditions, and corresponding outputs and classes.
- **Model-based AI software testing**, in which selected machine learning models are extended to be traceable and testable AI test models to facilitate AI software testing and operations in quality evaluation of training data and test data.

- **AI-based testing for AI software,** in which AI models and data-driven techniques are used to facilitate and optimize AI software testing in different perspective.
- **Metamorphic (Non-Oracle) testing,** in which a property-based software testing technique is used as an effective approach for addressing the test oracle problem and test case generation problem.
- **Testing robots for AI software,** where automatic software test robots are built and used to learn and follow experienced testers to perform user-oriented testing operation using collected user testing scenarios and test data.
- **Learning-based AI software testing using the crowd-sourced approach,** in which selected machine learning models and approaches are used to learn from crowd-sources testers in a service platform.

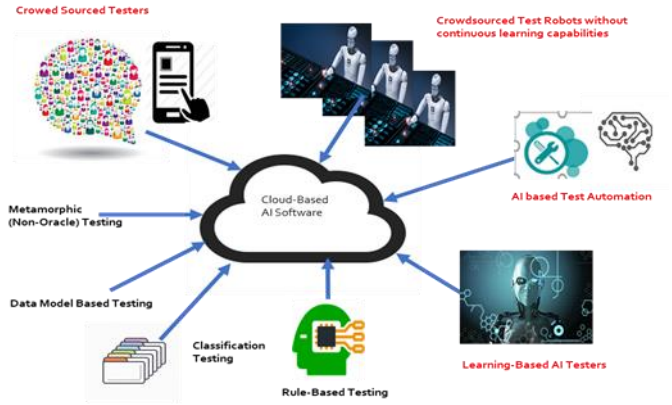

Figure 3. Different Testing Approaches for AI Software

AI Software Testing Services

There are several types of testing services for AI software.

- ***AI testing services for AI software vendors and workshops*** to provide them cost-effective quality testing and assessment using well-defined models and methods.
- ***AI quality certification services as a third-party*** for government agencies, and customers to assure AI software quality.
- ***AI software quality testing services for AI software customers and users*** to help them to perform acceptance AI testing.

3. Why Do We Need AI Software Testing?

Since 2017, we asked our software quality testing class students (CMPE287) in San Jose State University to validate numerous selected mobile apps with AI features. We requested the students to validate the selected mobile apps by targeting functions/features built-in with machine learning techniques. Students are required to perform their testing projects using selected conventional testing methods and approaches, such as scenario-based testing and decision table testing approaches.

In general, AI software need to be tested at both function and system levels. At the system level, all of system quality of service (QoS) parameters must be validated like conventional system testing. The common QoS parameters include system performance, security, reliability, availability, and scalability.

Compared to conventional software testing, AI function testing process primarily focuses on their unique features, such as oracle problems, learning capability, and timeliness testing. Figure 4 shows a sample AI function test process. Our students have followed in this process to carry out their AI testing projects.

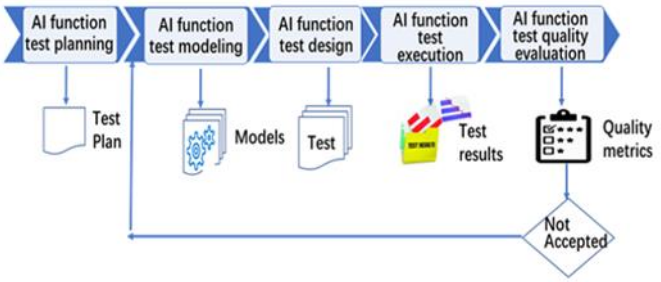

Figure 4. AI Software Function Validation Process

Step #1: AI function test planning

In this step, testers need to perform several tasks:

- ***Task partitions and scheduling*** – In our project experience, students found that this is not easy task due to the complexity of AI function testing as well as the lack of AI knowledge and understanding.
- ***AI function identification, understanding and partitioning*** – For many selected AI mobile APPs, students have difficulties in identifying and partitioning AI functions due to the lack of well-defined AI function requirements given by vendors. In many cases, several AI functions are aggregated together to generate system outcomes. ‘
- ***AI function test requirement analysis*** – Many CMPE 287 class students have encountered the difficulty in generate AI function test requirements in a system approach although most of them have used the scenario analysis approach. One of the major reasons is the lack of effective test requirement analysis approaches to assist testers to perform AI function test requirement analysis from data and context perspectives.
- ***Tool selection*** – Although many existing software testing tools are available for white-box testing, and system GUI and performance testing, students have problems to find any useful and practical tools for AI function validation.

Step #2: AI function test modeling – In this step, testers need to perform test modeling based on the established test requirements. Although many of student groups have selected existing test models, such as decision tables, state diagram or event-based GUI models, for their AI software testing, they have found that they are not suitable or adequate for them to come out sound and effective test models for their selected AI functions.

- **Step #3: AI function test design -** This step focuses on AI function test design and test case generation in terms of system input and expected system outcomes. In conventional function testing, test design is performed by applying existing testing methods, test cases are generated by identifying input test data and expected output data. In our AI testing projects, may student found the following two facts:

- Many AI functions are developed to accept diverse rich media inputs (text, image, audio or event video), and generated multiple outcomes in terms of rich media and events/actions. This brings more complexity in test case preparation and test data generation.
- Many existing test design methods have their limits on addressing large-scale data-centered AI function testing design needs, and are not effective to come out quality test cases to detect AI function quality issues relating to oracle problems, AI function accuracy, correctness, and consistency. One of the primary reasons has something to with the fact that most of existing test methods are designed to detect system function errors relating program logics, business rules, and system functions based on limited input data parameters with limited data scope and simple data types.

- **Step #4: AI function test execution – I**n this step, testers need to perform and execute their test cases with input data to detect AI function quality issues by checking the actual outcomes and results. In addition, they need to report the detected problems (bugs) during their testing. One major difficulty in their bug reporting and analysis is to identify and locate the causes of AI function quality issues, such as correctness, consistency, and accuracy. Data scientist and AI software engineers may have encountered the same problem in identifying the isolating the cause of AI function quality issues.
- **Step #5: AI function test quality evaluation –** In this step, testers need to evaluate their testing quality and decide if they have done enough in function validation. Since AI software has special features such as non-oracles, timeliness and learning capability, here function test quality evaluation is added particularly as the final step of AI software testing process. In this step, different quality parameters are measured using the pre-defined quality metrics based on testing result analysis. If the evaluation results are not accepted by stakeholders, the testing step goes to test modeling again for a new testing iteration Our students have also encountered the issues in AI function test quality evaluation due to the lack of well-defined AI function quality assessment models, metrics, and tools.

Now, let's summarize the major causes to conduct AI testing:

- **Cause #1** - Current existing software testing models and methods have limits to address AI software testing needs in supporting multi-models with unstructured input data, addressing large-scale classified inputs, and considering oracle problems, and quality accuracy, consistency, and correctness as well as relevance.
- **Cause #2** – Most current AI software are built-in with machine learning models developed by data scientists through large-scale data training using scientific algorithmically approaches instead of engineering approaches. Hence, there is a big gap in considering quality validation and quality assurance from engineering perspectives. Hence, AI testing research is needed to study and develop new and effective quality standards and evaluation methods.
- **Cause #3** – Building powerful AI software needs to use large-scale training and test data sets. The current train methods and data generation lack of quality consideration, quality assessment, and certification. Hence, how to come out quality training data models, develop large-scale quality test data generation methods will be needed.

Therefore, ***AI function testing*** targets at built-in AI features in AI software applications. It refers to different testing activities to find AI software errors, verify evaluate quality parameters with well-defined testing models and quality assessment methods. The testing goal is to validate well-defined test requirements, meet pre-defined testing criteria, and standards of quality assurance of the under-test AI software.

4. AI-Based Software Testing

A related topic for AI Software testing is AI-based software testing. AI-based software testing refers to the leverage and applications of AI methods and solutions to automatically optimize a software test process in test strategy selection, test generation, test selection and execution, bug detection and analysis, and quality prediction. In the recent years, there are numerous research papers addressing this subject in different areas. They could be useful in facilitating AI software testing and automation.

(a) Test selection and reduction using AI techniques

To deal with this problem, Souri et.al [14] used an AI-based testing technique named as Multi Objective Genetic algorithm (MOGA) to reduce the number of test cases for testing web applications yet achieve maximum coverage with reduced cost, time and space. Considering manual testing is a tedious and time-consuming task, and it may also result in insufficient testing being performed and critical defects going unidentified, Straub and Huber [15] proposed an artificial intelligence test case producer (AITCP) to test artificial intelligence system (AIS). AITCP starts from a human generated test scenario and makes changes to it based upon a modification algorithm such as ant colony optimization and genetic approaches. The authors compared the results of AI-based method and manual-based method for testing an autonomous navigation control system based on selected four scenarios. The study results show that AITCP can be utilized to effectively test AIS for both surface (two-dimensional) and airborne (three-dimensional) robots.

(b) Test result validation using AI techniques

Although there are many successful studies about automated generation of test cases, determining whether a program has passed a given test remains largely manual. Langdon et.al [16] proposed the use of search-based learning from existing open source test suites to automatically generate partially correct test oracles. They argued that mutation testing, n-version computing and machine learning could be combined to allow automated output checking to catch up with progress on automated input generation.

(c) AI-Based Machine Testing

AI-based machine learning requires a huge number of inputs as the knowledge and different intelligent algorithms in order to make the right decision. By looking at an example using technology in unmanned vehicles, there will be a basic understanding of how machine learning or machine intelligence works. The development of machine intelligence is still far from mimicking the cognitive competence of human brain. Yet, it is possible to simulate the driving

activity from human driving skills. The process of brain cognition of driving includes: audio-visual cognition, attention, memory, thinking, decision-making, interaction, and other tasks during driving. In driving activity process, the driving map and driving operation are derived from the knowledge from long-term memory. The rapid development of the ground mapping with higher accuracy and sensor technologies gain more and more data from the surrounding, but it is still challenging to handle with those data effectively and making a driving decision accurately and quickly [17]. Machine learning sometimes return an inaccurate prediction based on the collection of training data and an engineer needs to make some adjustments to avoid significant losses in terms of public safety.

Deep Learning is designed to continually analyze data with a logic structure as mimicking how a human can draw a conclusion. The deep learning needs a huge number of data sets to use input in the algorithms in order to result a more accurate prediction. For instance, Google's AlphaGo, a sharp intellect and intuition game, learns by itself without predefined data. It makes a more specific move and becomes the greatest player of all. Deep Learning defines a new paradigm based on data driven programming. Currently, some research and news show that there are various vulnerabilities in the current state-of-the-art deep learning systems that are likely to cause more serious losses or catastrophic consequences when applied to practical security-related fields. Since Machine Intelligence or Deep Learning depends on the training data, the accuracy and quality of data play a vital role for a public safety using machine learning in autonomous vehicles.

(e) AI-Based Fault Detection and Prediction

Numerous papers have been published to focus on AI-based fault detection and prediction in a software test process. The basic idea is to use artificial intelligent models which are trained based on the collected historical data in software testing, and bug problem reporting and analysis. In the recent years, many researches attempt to find solutions for current obstacles of Machine Learning Systems. To draw an optimal decision making, approaches such as Fault Tree Analysis [20], Fuzzy Logic [21], Metaheuristic Algorithm [22] [23], and Artificial Neural Network [24] are developed to test with huge amount of training data by using different algorithms.

However, the sufficiency and versatility of Deep Learning systems are based on the accuracy of test data set. It is difficult to provide an adequate support due to the accessibility of test data quality issue. The current Deep Learning systems has various vulnerabilities and their system analysis and defect detection are extremely difficult. Unlike traditional software systems, Machine Intelligence does not have a clear controllable logic and understandability since the process to make decision rely on the training data. News have been broadcasting about the accidents caused by unexpected unforeseen driving conditions in smart vehicles. The reason behind these consequences are due to lacking a systematic way of adequate testing in Deep Learning systems. The recent study shows two major vulnerabilities in Deep Learning systems: Software quality from output of Deep Learning alone is not adequate; and Failure in unseen attacks even though Deep Learning are immune to known types of attacks [18, 19].

4. TEST MODELING FOR AI FUNCTIONS

Like conventional software testing, AI software testing also needs establish test requirements and test model to support quality testing. To facilitate student testing projects in CMPE287 class, we ask students to use a new classification model, known as a **three-dimensional classification decision table (3D Classification Table)** to help them to conduct classification-based test requirement analysis and modeling for any given mobile apps powered with AI functions in object detection, classification, and prediction. Our experience indicated that this model provides students a step-be-step systematic approach in building their AI function test models.

The basic test modeling procedure for each selected AI function consists of the following steps:

- **Step #1: AI function context classification modeling** – In this step, a tester needs to identify and classified diverse context conditions and parameters, and present the classification results using a context classification model, known as a context classification tree.
  A context classification tree is a 3-tuples, denoted as GCT= (NCT, ECT, RCT), where NCT is a finite nonempty set of nodes with a node label. There are three types of nodes in NCT, including a root node (NR), intermediate nodes (NI) and leaf nodes (NL). Figure 5 shows a simple example. ECT consists of a set of edges, and each connects two nodes in the tree (GCT), and represents of different category semantic relations between them. These semantic relations are included in RCT as its elements. There are four types semantic relations: AND, XOR, SELET-1 and SELECT-M. The table below shows the detailed descriptions.

Semantic Relations	Descriptions
AND (N_P, <N_{C1}, … N_{Cn}>)	N_P has an AND relation with its n child nodes when all of its child nodes must be included.
XOR (N_P, <N_{C1}, N_{C2}>)	N_P holds an XOR relation with its two child nodes if only one of its two child nodes could be selected.
SELECT-1 (N_P, <N_{C1}, … N_{Cn}>)	N_P has a SELECT-1 relation with its child nodes when only of its child nodes could be selected.
SELECT-M (N_P, <N_{C1}, … N_{Cn}>)	N_P has a SELECT-M relation with its n child nodes if and only if m of n child nodes could be selected.

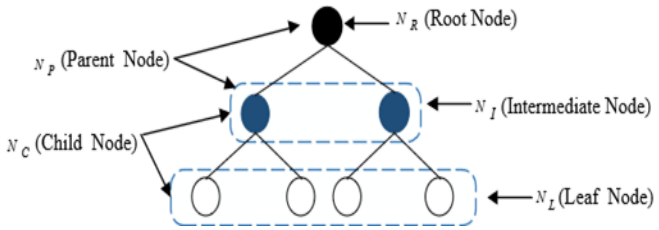

Figure 5 A tree sample and with nodes notations

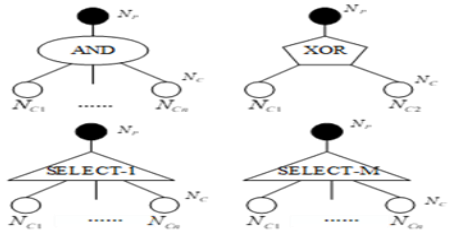

Figure 6. The Semantic Relations between Nodes

Figure 7 shows an example of the context tree model for a mobile App (known as HomeKit). This intelligent mobile App is Apple's smart-home software. It works in the background on modern Apple devices, and allows you to control your connected lightbulbs, locks and thermostats without jumping to a different

app for each one. HomeKit supports both text inputs and voice inputs.

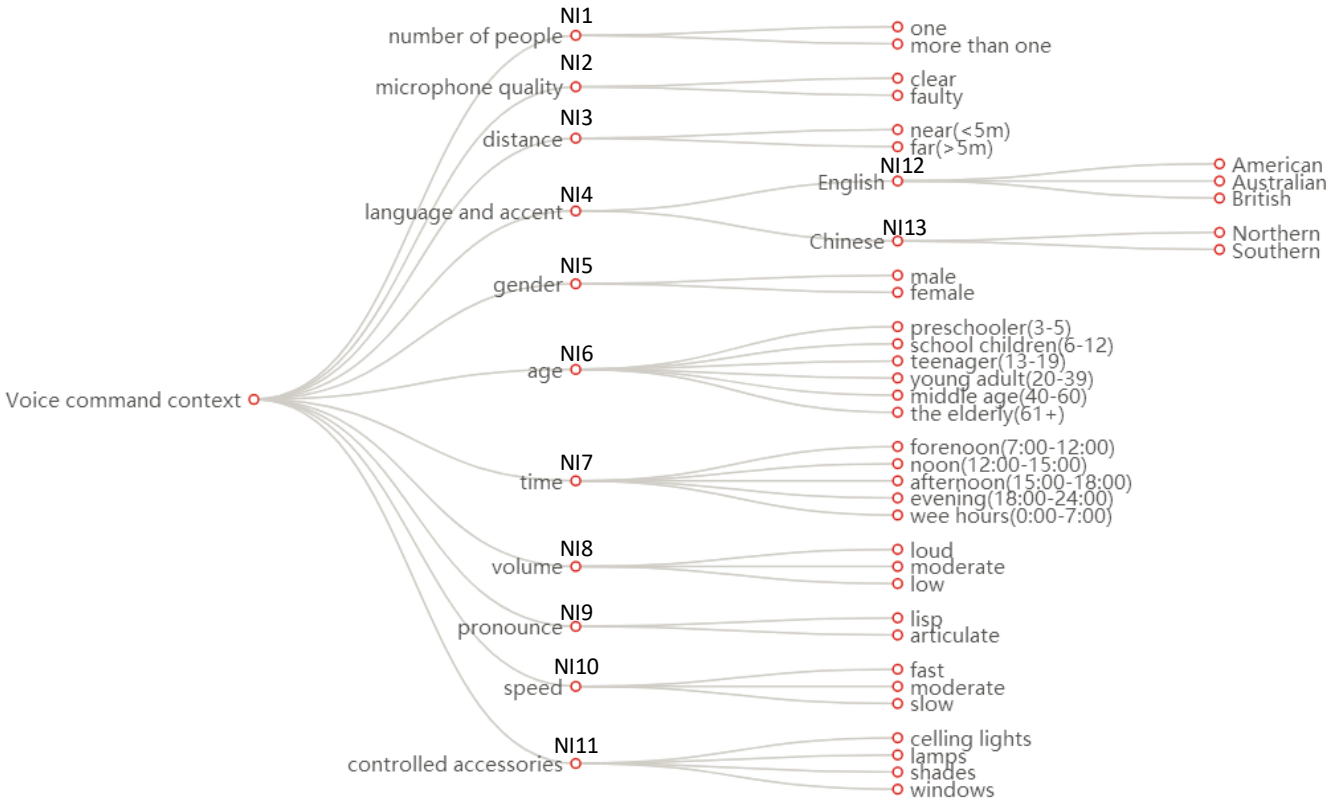

Figure 7
The Context Classification Tree for HomeKit's Voice-Command Function

- **Step #2: AI function input classification modeling -** In this step, a tester needs to focus on input classification to identify and classified diverse input data in terms of its category classes and their sub-classes. When an AI-based function accepts multiple input media formats (such as video, audio, and image, and text), each of them should be examined and classified. To effectively support a tester to conduct input classification, we used an input classification model (known as input classification tree) as our analysis and test model to facilitate and represent diverse input data classes and their sub-class using a category approach. Figure 8 shows an example of an input classification tree generated for HomeKit.

- **Step #3: AI function outcome classification modeling** – In this step, a tester focuses on the classification of diverse AI function outputs, including texts, audio, video, and images, or events (or actions). Similar to input classification, an output classification tree model is generated as the outcome of this step.

- **Step #4: Generate a 3D classification decision table** – In this step, a tester generates a new decision table for each under-test AI function feature, known as 3D classification decision table, to identify three dimensional mappings among disjoint classified context conditions, disjoint classified inputs, and disjoint classified outputs.

Figure 8. The Context Classification Tree Model for Voice-Command Function.

In the past, a conventional decision table has been used as an effective testing method to validate a system's business rules and cause-effect actions behaviors. A conventional decision table represents complex business rules based on a set of conditions. Each decision table consists of condition stubs and entries, corresponding action stubs and action entries. Each column of a decision table is a rule that specifies the conditions under which the actions listed in the action stub will take place. Decision table is a systematic approach where the different input combinations and their corresponding system behaviors (or outputs) are captured in a tabular form. It can

be also called as a cause-effect table where cause and effects are captured for better test coverage. Decision table can enumerate complex problems in all possible situations, concise and it could avoid missing. The decision table can be used to design a complete set of test cases.

The proposed 3D classification decision table for any AI function is influenced by the concept of conventional decision tables. Its major objective is to model any given AI classification function using a three-dimensional tabular view as shown in Figure 9. Each 3D classification decision table consists of three views for each AI system function:

a) ***AI function context classification view,*** where identified diverse context conditions are listed as context stubs, and the related disjoint condition selection are represented as distinct context classification rules.

b) ***AI function input classification view***, where identified diverse inputs are listed as classified input stubs, and the related disjoint selections are represented as distinct input classification rules.

c) ***AI function outcome classification view***, where identified diverse AI function outputs (events or outputs) are listed as classified output stubs, and the related disjoint selections are represented as distinct output classification rules.

Initiatively, the mappings among classified disjoint context conditions, classified input selections, and classified AI function outputs are the major testing focus for a propose 3D classification table. These mappings are known as **AI function classification rules**. Each of them represents the conjunction among three different views. Clearly, test case design and generation based on 3D classification decision table must cover these AI function classification rules. Adequate AI classification function testing coverage could be assessed based on our proposed 3D classification decision table.

To facilitate systematic test case generation, we discovered that an effective way is to identify and generate spanning trees for each identified classification tree models in three dimensions. Figure 10 shows two identified spanning trees based on the context classification tree for HomeKit in Figure 7. Each of them shows one disjoint classification of identified context conditions for HomeKit's voice command function. We have developed a spanning tree identification and generation algorithm for the 3D classification decision table, a test automation project has started to implement the proposed idea and classification models. The details will be reported in the future publication.

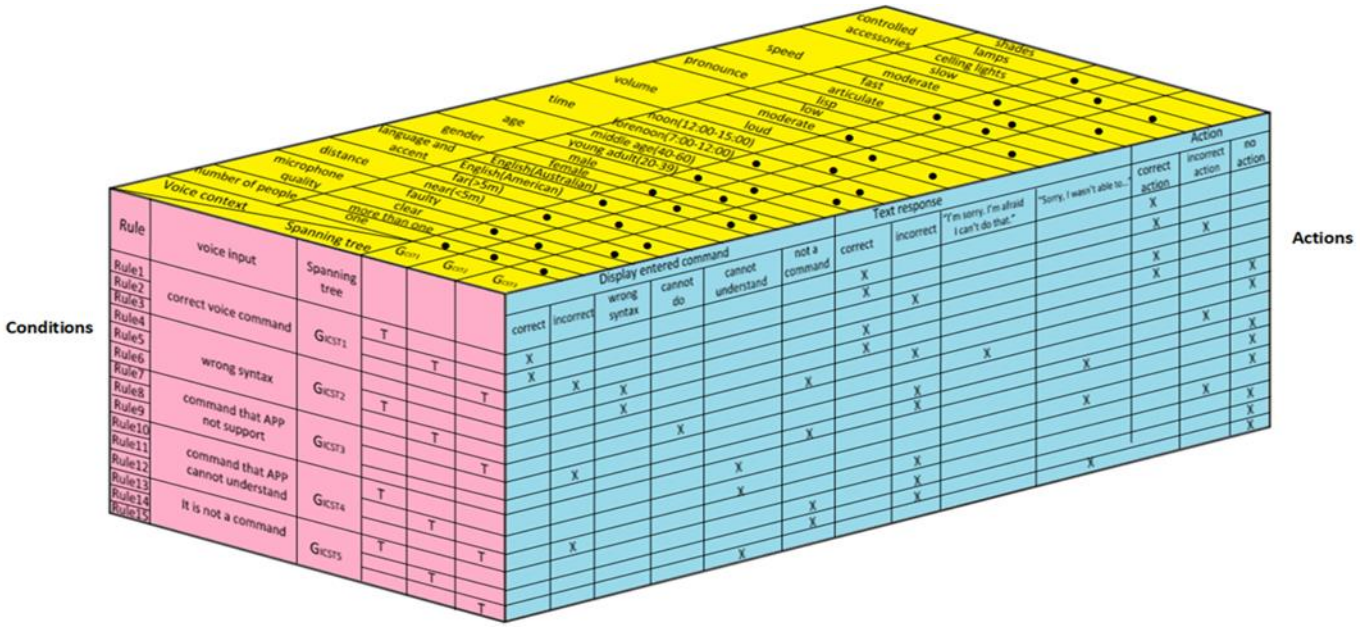

Figure 9. 3D Classification Decision Table for HomeKit Voice-Command AI Function

Figure 10. Two Context Spanning Tree Examples (GCST1 and GCST2)

5. CHALLENGES, ISSUES, AND NEEDS

AI software quality validation has a number of major challenges due to the lack of research work results and engineering experience reports. These challenges are summarized below.

- **Challenge #1:** How to establish the quality assurance requirements and testing coverage criteria for AI systems which are built using machine learning methods based on big data?
- **Challenge #2:** How to use systematic methods to establish and develop quality test models for learning-based AI systems?
- **Challenge #3:** How to use a systematic method to prepare quality training datasets and coverage-oriented test datasets for AI-based functional features in learning-based AI for todays' AI systems?
- **Challenge #4:** How to define quality assurance standards systems, and develop adequate quality test coverage?
- **Challenge #5:** How to develop automatic solutions and tools to support AI-based system validation?

In addition, there are a number of issues in AI software testing. Here are the primary ones summarized below.

Issue #1– There is a lack of well-defined adequate validation models and test coverage criteria for AI functions and features?

The increasing deployment of artificial intelligent and machine learning models and techniques in modern software applications raises quality validation modeling concerns. Most existing white-box and black-box software test models are developed to present program functions, dynamic behaviors, and structures. These models do not address quality validation needs relating artificial intelligent features and functions, such as rich oracle functions, detection and classification, recommendation and prediction. This leads to the first need described below.

Need #1 – Developing well-defined adequate validation models and criteria to address and present the special features and needs in testing AI-based functional features, such as object detection and classification, recommendation and prediction features, and so on.

Issue #2 – Where are the well-defined quality assurance and validation standards, including approaches, processes, assessment metrics, regulations, and policies?

The existing software quality assurance standards, processes, and validation criteria have been developed to address the quality validation and assurance needs for conventional function-based software and applications. These systems are developed to perform specific-based functions and operations, and generate outputs and actions based on well-defined input value spaces with data inputs, signals and events, limited business rules, specified dynamic behaviors. The existing quality assurance are set up to validate two sets of quality parameters. The first set is related functional quality, including program correctness in program execution follows, dynamic behaviors, business decision-makings, program input-and-output pairing, and system operations. The other quality parameter set includes non-function quality parameters, including performance, scalability, security, reliability, and availability, and so on.

Unlike conventional software systems, current AI-based systems are developed based on machine learning models using data-driven training and validation based on limit datasets selected from one or multiple big data spaces.

This brings to new quality assurance needs in evaluating AI-based program quality parameters in correctness, accuracy, consistency, relevancy, and so on. These new quality evaluation focuses lead to the second demand below.

Need #2 – Establishing well-defined quality assurance programs and standards to address the special quality parameters relating to AI functional features in AI-based systems.

Issue #3– Where are automatic solutions and tools supporting efficient and large-scale automatic testing operations for AI-based functions in modern intelligent systems build based on machine learning models and big data?

In the past three decades, many test automation tools and solutions have been developed for engineers in assisting their test automation activities and operations in diverse software application systems. Unfortunately, most of these tools are only useful to validate conventional functions, program behaviors, program structures, as well as system non-functional quality parameters, including performance, reliability, scalability, and so on. As discussed in section 3, there are a few existing AI testing solutions for AI-based system validation. However, there is a clear lack of research work on automatic validation methods and solutions for AI software systems. Therefore, the third emergent need for AI software testing is listed below.

Need #3 –More innovative adequate testing methods and test automation tools to address the special needs and features of AI software and applications to deal with the coverage of big data spaces.

Unlike conventional software test automation tools, these expected test automation solutions must consider AI's special features and needs listed below:

- Large-scale big data inputs with diverse formats, and structured and non-structured data;
- Learning-based and knowledge-based system evolutions;
- Non-oracles problems and rich oracle functions with uncertainty;
- New QoS parameters, such as accuracy, consistency, correctness, accountability, usability, and
- Automated quality test data generation and discovery using crow-sourcing approaches and learning-based solutions.

5. CONCLUSIONS

This paper provides perspective views on AI software validation, including the tutorial concepts, test features and focuses, as well as validation process and test modeling for AI classification function testing. Moreover, the primary types of AI software testing and existing validation approaches are analyzed and discussed. Moreover, the paper also discusses The primary challenges, issues and needs are presented in the end.

References

[1] B. Wang. Automated Support for Classifying Software Failure Reports. In Proc. of the 25th International Conference on Software Engineering (ICSE), pp. 465–475, 2003.

[2] P. Francis, D. Leon, M. Minch, and A. Podgurski. Tree-Based Methods for Classifying Software Failures. In Proc. of the 15th International Symposium on Software Reliability Engineering (ISSRE), pp. 451–462. 2004.

[3] J. F. Bowring, J. M. Rehg, and M. J. Harrold. Active Learning for Automatic Classification of Software Behavior. In Proc. of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pp.195–205, 2004.

[4] X. X. Xie, J.W. Ho, C. Murphy, G. Kaiser, B. W. Xu, and T.Y. Chen. Testing and Validating Machine Learning Classifiers by Metamorphic Testing. Journal of System and Software, 84 (4):544–558, 2011.

[5] C. Murphy, G. Kaiser, L. Hu, and L. Wu, Properties of Machine Learning Applications for Use in Metamorphic Testing. In Proc. of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), pp.867–872, 2008.

[6] T. Y. Chen, J.W.K. Ho, H. Liu, and X. Xie. An Innovative Approach for Testing Bioinformatics Programs Using Metamorphic Testing, BMC Bioinform. 10(2009).

[7] T. Y. Chen, J. Feng, and T.H. Tse. Metamorphic Testing of Programs on Partial Differential Equations: A Case Study. In Proc. of the 26th Annual International Computer Software and Applications Conference, (COMPSAC), pp. 327–333, 2002.

[8] J. Mayer and R. Guderlei. On Random Testing of Image Processing Applications, In Proc. of the 6th International Conference on Quality Software (QSIC), pp. 85–92, 2006.

[9] K. Meinke, and F. Niu. A Learning-Based Approach to Unit Testing of Numerical Software, In Proc. of the 22nd IFIP WG 6.1 International Conference on Testing Software and Systems, pp. 221–235, 2010.

[10] S. A. Vilkomir, W. T. Swain, J. H. Poore, and K. T. Clarno. Modeling Input Space for Testing Scientific Computational Software: A Case Study. In Prof. of the 8th International Conference on Computational Science, pp. 291–300, 2008.

[11] W. H. Deason，D.B. Brown，and K.H. Chang. A Rule-Based Software Test Data Generator. IEEE Transactions on Knowledge and Data Engineering, 3(1): 108-117, 1991.

[12] A. Andrews, A.O. Fallon, and T. Chen. A Rule-Based Software Testing Method for VHDL Models. VLSI-SOC 2003: 92.

[13] P. Kumudha, R. Venkatesan, Cost-Sensitive Radial Basis Function Neural Network Classifier for Software Defect Prediction, vol. 2016, 2016.

[14] A. Tosun, A. Bener, B. Turhan, T. Menzies, "Practical considerations in deploying statistical methods for defect prediction: A case study within the Turkish telecommunications industry", Inf. Softw. Technol., vol. 52, no. 11, pp. 1242.

[15] Aleem, L. F. Capretz, F. Ahmed, "Benchmarking Machine Learning Techniques for Software Defect Detection", Int. J. Softw. Eng. Appl., vol. 6, no. 3, pp. 11-23, 2015.

[16] Z. He, F. Peters, T. Menzies, Y. Yang, "Learning from open-source projects: An empirical study on defect prediction", Int. Symp. Empir. Softw. Eng. Meas., pp. 45-54, 2013.

[17] W . B . Langdon, S. Yoo, M. Harman, “Inferring automatic test oracles[C]”, 2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST). IEEE, 2017:5-6.

[18] R. Malhotra; Malhotra, Ruchika, “A systematic review of machine learning techniques for software fault prediction”, Applied Soft Computing Applied soft computing, 2015, Vol.27, p.504-518.

[19] Abdul Rauf, Mohammad N. Alanazi, “Using artificial intelligence to automatically test GUI”, The 9th International Conference on Computer Science & Education (ICCSE 2014) August 22-24, 2014. Vancouver, Canada.

[20] Dai, Wenbin, et al. "A Cloud-Based Decision Support System for Self-Healing in Distributed Automation Systems Using Fault Tree Analysis." IEEE Transactions on Industrial Informatics 14.3 (2018): 989-1000.

[21] Tahvili, Sahar, et al. "Towards earlier fault detection by value-driven prioritization of test cases using fuzzy topsis." Information Technology: New Generations. Springer, Cham, 2016. 745-759.

[22] Ahmed, Bestoun S. "Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing”, Engineering Science and Technology, An International Journal 19.2 (2016): 737-753.

[23] Huang, Rubing, et al. "Poster: An Experimental Analysis of Fault Detection Capabilities of Covering Array Constructors." 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). IEEE, 2018.

[24] Shreejith, Shanker, Bezborah Anshuman, and Suhaib A. Fahmy. "Accelerated artificial neural networks on FPGA for fault detection in automotive systems." 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2016.

[25] Jameel T, Mengxiang L, Chao L. “Automatic test Oracle for image processing applications using support vector machines,” 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS). IEEE, 2015: 1110 1113.

[26] https://www.apple.com/siri/
[27] https://www.caloriemama.ai/
[28] https://www.microsoft.com/en-us/seeing-ai
[29] https://play.google.com/store/apps/details?id=com.neurotec. c hec kmyage&hl =en _US
[30] https://w w w.lakeima ge.co m /pro duc ts/di sco very- m ulti scan/
[31] http://firsteigen.com/databuck/

ACKNOWLEDGEMENT

This paper is supported by the National Key Research and Development Program No.2018YFB1003902; the National Natural Science Foundation of China under Grant No.61402229 and No.61602267, and the Open Fund of the State Key Laboratory for Novel Software Technology (KFKT2018B19).