Testing AI Software– Perspectives, Challenges, Issues, and Needs

Chuanqi Tao[①②], Jerry Gao[③], Shengqiang Lu[④], Tiexin Wang[①]

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

③Department of Computer Engineering , San Jose State University, San Jose, USA

④College of Computer Science and Technology, 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 diverse AI-based software in different application domains is becoming a very hot research and topic among academic and industry communities, and government agencies. Therefore, many machine learning models and artificial techniques have used to develop smart applications based on multimedia inputs to achieve intelligent features, such as recommendation, object detection, classification, and prediction, natural language processing and translation, and so on. This brings strong demand in quality validation of AI software to assure the quality of AI features. 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 testing for quality validation, including validation focuses, features, and process, and potential testing approaches. Moreover, the paper presents test quality evaluation and test coverage problems. Furthermore, the primary issues, challenges, and needs in testing AI software are presented.

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 software and applications could be classified into the following categories: a) natural language processing (NLP) systems with language understanding and translation; b) detection and recognition systems, for example, human face identification, voice recognition and object detection; c) recommendation systems in e-commerce and advertising; d) unman-controlled vehicles, robots, and UAVs. Based on our recent testing experience with diverse mobile AI apps, testing AI software has new problems, challenges and needs due to their special features below.

- Scientific-based development instead of engineering-based development: Most AI software and applications are developed using scientific approaches based on AI models and training data by data scientists and big data engineers without well-defined AI software engineering process and development methods with clear quality validation requirements and criteria.

- Limited data training and validation - AI software is built based on machine learning models and techniques, and trained and validated with limited input data sets under ad-hoc contexts.

- Data-driven learning features – These features provide static and/or dynamic learning capabilities that affect the under-test software results and actions.

- Uncertainty in system outputs, responses and decision makings – Since existing AI-based models are dependent on statistics algorithms, this brings the uncertainty in the outcomes of AI software.

These unique AI software features causes new difficulties and challenges in quality validation, hence, perform AI quality validation and assurance becomes a critical concern and a hot research subject. 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 function or feature view. There is an emergent need in research work to quality study issues and quality assurance solutions for AI software and applications. Testing AI software can be considered as diverse testing activities with the intent of finding AI-based software bugs (errors or other defects), verifying that the AI-based software products is fit or use, assuring AI functional features' adequate quality and AI software's QoS (quality of system service) parameters. Well-defined quality validation models, methods, techniques, and tools must be developed and applied for AI-based software to facilitate the test activities to achieve well-defined test requirements and meet pre-selected adequate testing criteria and quality assurance standards.

This paper is written to provide our perspective views on AI (specific to feature or function) testing for quality validation. The paper is organized as follows. Section 2 discusses the tutorial concepts about AI software testing, including test focuses, features, and process. Section 3 reviews AI-based machine testing, AI software function testing, as well as the existing testing methods potentially-used for AI software validation. Section 4 discusses AI software testing quality parameters and evaluation as well as test coverage analysis. The major issues, challenges, and needs are presented in Section 5. Conclusion remarks are in Section 6.

2. UNDERSTANDING AI SOFTWARE TESTING

Why do we need AI software testing? The fast-growing AI software and the popularity of big data-based applications bring new needs and motivations. Numerous current and future software will be built with AI-based features and functions. Existing techniques and tools are not adequate to test AI-based features and functions. There are a lack of well-defined and experience-approved quality validation models and assessment criteria. In addition, there are a lack of AI-based testing methods and solutions for AI software. Thus, the meaning of testing AI software is illustrated in a definition below.

*"**Testing AI software** refers to diverse testing activities for AI-based software/systems. Well-defined quality validation models, methods, techniques, and tools must be developed and applied for AI-based software to facilitate the test activities to achieve well-defined test requirements and meet pre-selected adequate testing criteria and quality assurance standards."*

Therefore, testing AI features of the software includes different testing activities to find software errors, verify the performance of software, and assuring quality validation methods needs to be developed. The testing goal is to achieve well-defined test requirements, meet pre-defined testing criteria, and standards of quality assurance of the under-test AI software.

2.1 Test Scope and Major Focuses

Since AI software are built with diverse machine learning models and data-driven technologies, the scope of AI software testing should cover current typically-used intelligent features, such as prediction, recognition, and recommendation. Figure 1 shows the primary scope of AI software testing. Objects (human, animal) related testing such as object identification, recognition, and behavior detection are an important part of AI software testing. Various intelligent applications such as business decision, recommendation and selection, intelligent commands and actions, analytics and prediction capability, as well as question and answer capability are current key AI testing topics. In addition, with the advance of unmanned vehicles and their potential huge markets, how to perform control validation and healthcare check will be a big challenge for AI testing and quality validation. Moreover, AI software usually involves context issues, such as scenario, location, time, and stakeholders, thereby causing new testing issues in context identification and classification. The major focuses of AI software testing are summarized as follows.

(a) Testing AI functional features to assure their adequate quality in accuracy, consistency, relevancy, timeliness, correctness, and so on using data-driven and AI approaches.

(b)Testing AI software's quality of system service parameters based on well-defined quality standards and assessment criteria. These include system performance, reliability, scalability, availability, robustness, and security, and etc.

(c) Apply data-driven AI techniques to facilitate AI testing processes and test automation.

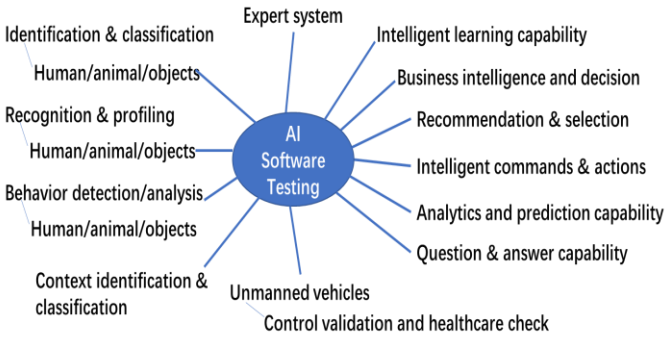

Figure 1 The Scope of AI Software Testing

2.2 New Testing Features and Requirement Analysis for AI Software

As discussed above, AI software and applications have numerous unique testing features such as uncertainty and limited training/test dataset. These unique features bring more interesting quality validation and QoS requirements, challenges, and needs. Based on the recent feedbacks from engineers at Silicon Valley, how to assure the quality of AI software becomes a critical concern and research subject currently. The primary testing features are presented as follows.

Multiple dimension-based rich media input data with multi-input models. – This refers to new testing solutions to deal with multi-dimensional large-scale input data sets (such as numerous image graph and videos) of AI software. For example, the well-known AI application *Seeit*[1] supports text, graph, voice, and audio with diverse input domains both offline and online.

Test data set selection from big data pools. – This refers to test data selection to address the special testing features of AI software. In traditional software, test data is used for finding software bugs. Nevertheless, in AI software, test data is not just used for functional or program bugs. Bugs or defects existed in training and learning models in AI software are also needed to be discovered using specific test data. A typical face recognition application 'how old do I look' from Microsoft[2] can be tested with thousands of pictures to indicate its correctness and accuracy. However, how to select effective test data to discover its identification problems, e.g., the accuracy of 'how old do I look' is affected by lighting condition or background objects. Furthermore, bugs from models or learning algorithms can be detected with more test data with specific goals.

Knowledge-based AI software features and behaviors – This refers to apply the domain-specific knowledge to assist in testing correct and precise AI software features and behaviors.

Uncertainty of AI software features and behaviors. – This refers to how to define and modeling testing objects in a certain way and obtain testable functions through different test strategies, such as metamorphic testing, mutation testing and fuzzy testing.

Learning-based AI software features and behaviors. – This refers to finding new testing approaches to address the leaning features of AI software. For instance, the learning capability of AI software is needed to be tested in an evolved environment.

[1] https://itunes.apple.com/cn/app/seeit/id721911549?l=en&mt=8

[2] https://www.how-old.net/

Real-time context-based diverse inputs affecting system outputs, actions, and behaviors. – This refers to modeling complex context factors in a real-time instance, and analyze the relationship among diverse contexts, inputs, outputs, and actions.

After identifying the primary AI features, AI function features are analyzed for testing. For each identified feature, AI testing requirements are needed to analyze for future testing. For example, before testing an object of AI software, in order to facilitate function or scenario testing, diverse features are required to classified with a well-defined category. Test models are necessary to represent the diverse features under testing. In general, models can be constructed from different perspectives for AI software, such as knowledge test model, feature test model, object test model, and data test model. As shown in Figure 2, features of object relation, object identification, object behavior, object classification, and object context are selected for function testing with diverse sub-features.

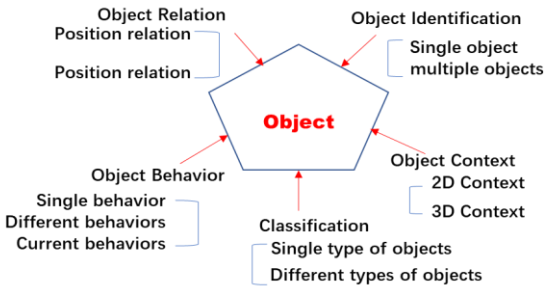

Figure 2 A Sample Object Model-Based AI Software

Figure 3 shows a sample modeling for object identification through different test features and requirements, including object under different contexts, diverse object classes and subclasses, different object attributes, incorrect objects as well as no objects.

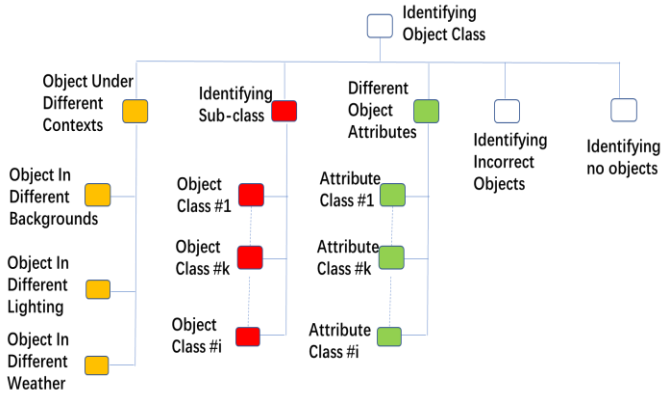

Figure 3 A Sample Test Modeling for Object Identification AI Software

2.3 Testing Process for AI Software

Compared to conventional software testing, a test process of AI applications primarily focuses on their unique features, such as oracle problems, learning capability, and timeliness testing. Figure 4 shows a sample test process (function and system testing).

In general, AI software need to be tested at both function and system levels. Test planning, test modeling, test design, and test execution are the indispensable parts of the overall testing process for both AI software and traditional software. 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. As shown in the bottom of Figure 4, AI system testing has similar validation process to that of function testing.

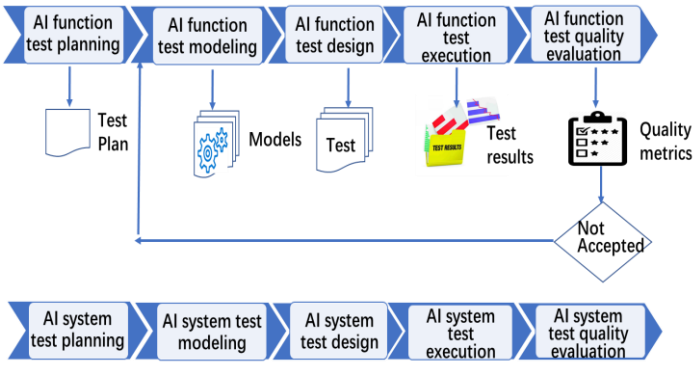

Figure 4 AI Software Validation Process

3. AI SOFTWARE QUALITY VALIDATION CATEGORY AND APPROACHES

This section firstly illustrates a category of AI software testing, including Turing testing, testing AI software, AI-based software testing and AI-based machine testing. Then several existing and potential approaches to AI software testing will be presented and discussed.

3.1 Turing Testing

Turing test was introduced by Turing as the imitation game in 1950 [5], aiming to test a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human. Turing proposed that a tester would ask the testee freely through some devices (such as a keyboard) in the case where the tester is separated from the testee (one person and one machine). After multiple tests, if more than 30% of the testers are unable to determine whether the testee is a human or a machine, then the machine passes the test and is considered to have human intelligence. The turning test has been considered as the "beginning" of artificial intelligence (AI) [6], and it has also become an important concept related to AI system testing. Although the Turing test was designed to advance the development of artificial intelligence, it also has several shortcomings [7].

3.2 AI Software Testing

In this section, the main focus is on validating AI software functions, external behaviors, and external visibility of QoS using black box testing techniques. To test software functions

and features, engineers could adopt convention black-box approaches to validate software quality. Typical examples include scenario analysis, decision table testing, equivalence partitioning, boundary value analysis, cause-effect graph, and so on.

However, AI software testing differs from traditional software testing, since AI applications are characterized by uncertainty and probabilities, dependence on big data, random input/output, difficulty in predicting all application scenarios, and constant self-learning from past behavior. In recent years, many studies have worked on researching how to test AI software or systems [7-11].

Broggi et.al proposed the Public Road Urban Driverless (PROUD) test which conducted in Parma from the University campus to the town center through different scenarios such as urban, rural, and highway roads [7]. Similarly, Li et.al [8] indicated the difficulties of intelligence test from four aspects, and presented an example on how to design intelligence test for intelligent vehicles. The authors gave the definition and generation of intelligence test tasks for vehicles to combine the benefits of scenario-based testing and functionality-based testing approaches based on a semantic relation diagram for driving intelligence proposed in [9]. In addition, the authors applied the parallel learning method to the vehicle intelligent test and proposed a parallel system framework that combined the real-world and simulation-world for testing [10] [11].

As discussed above, the process of testing AI functions includes test planning, test modeling, test case generation, test execution, and test quality evaluation. Decision table testing design technique determines the different combination of inputs with their associated outputs and implements the business requirements or rules of the system. It is also a represented type of cause-and-effect testing or logical testing. Black box testing is basically used to test the end user requirements [12] [13]. It attempts to uncover the errors in the following categories: missing or incorrect functions, interface errors, behavior or performance errors, and initialization or termination errors.

Let us take *Siri*[3] from Apple for instance. The functions of Siri based on voice command input are listed as below: received voice commands, convert voice commands into text commands (display entered commands), find the text response and actions that match the recognized commands, text response, action response. To verify the AI functions of the software, traditional scenario analysis method is applied to analyze the scenarios of applications and test whether the main functions is implemented correctly from the perspective of the scene. Table 1 shows a description of five scenarios in testing *Siri.*

Table 1 A Sample Traditional Scenario Analysis on Siri

Scenario Description
1. Input voice command correctly; output the correct text response and correct action.
2. Input voice command with wrong syntax; output the correct text response and correct action.
3. Input voice command that cannot do; display command that cannot do; APP response: “I’m sorry. I’m afraid I can’t do that.”, and then do not put any action.
4. Input voice command that cannot understand; display command that cannot understand; APP response: “Sorry, I wasn’t able to…” or “Sorry, I didn’t get that…”, and then do not put any action.
5. Input voice content that is not a voice command; display sentence that is not command; APP reply incorrect text response, and then do not put any action.

Based on the analyzed results and testing experiences, we conclude that the test cases designed by scenario analysis is practical and effective to validate common features and conditions. However, there are some defects to generate test cases using scenario analysis as follows.

a. As a typical intelligent software application with AI features, *Siri* has rich context information. The different test contexts affect the results of testing Siri, such as the background noise, the tester’s gender, age and accent. However, the traditional scenario analysis does not consider these external conditions for testing. Hence, the designed use cases are incomplete, and execution results of some test cases failed.

b. Advanced AI software or systems have the ability to learn from data and experiences. Furthermore, some AI systems even learn from environmental interactions and learn dynamically during interaction with users. Thus, the more time you spend on using Siri, the better it will understand you. *Siri* achieved this by learning about your accent and some other characteristics of your voice. Therefore, if the same tester repeatedly tests Siri for the same voice command, its overall recognition of dialects and accents will continue to improve, test results will be also affected. Unfortunately, traditional scenario analysis does not take this into account.

In order to test the voice command-based AI functions more precisely, we should take different voice testing environments into account with context factors and modeling multi-dimensional testing space for AI features. Currently, we are working on this in another paper.

3.3 AI-Based Software Testing

AI-based Software testing refers to the leverage and applications of AI methods and solutions to automatically optimize a software testing process in test strategy selection, test generation, test selection and execution, bug detection and analysis, and quality prediction. It includes different testing activities in AI-based software testing. Due to the complexity of AI software and applications, traditional methods and test tools cannot meet the demands of testing these AI systems. Given this, a more effective method to test AI systems is desirable.

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

[3] https://www.apple.com/siri/

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.

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.

3.4 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.

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, Fuzzy Logic, Metaheuristic Algorithm, and Artificial Neural Network 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].

Thus, how to make machine intelligent testable is a great challenge for future AI-based machine testing.

3.5 Typical Valiation Approaches for AI Software

This section discusses the primary validation approaches from existing work and could be utilized potentially for testing AI software from different perspectives.

Classification-based testing-A classification approach to program testing usually involves two steps: a) training a classifier to distinguish failures from successful cases on a selected subset of results, and then b) applying the trained classifier to identify failures in the main set of results. A resembling reference model is usually used to train a classifier. More specifically, there are techniques for applying pattern classifications to alleviate the test oracle problems. Last et al. [20] and Vanmali et al. [21] applied a data mining approach to augment the incomplete specification of legacy systems. They train classifiers to learn the casual input-output relationships of a legacy system. Podgurski et al. classified failure cases into categories [22]. However, they did not consider how to distinguish correct and failure behaviors of programs. Later, their research group further proposed a classification tree approach to refine the results obtained from classifiers [23]. Bowring et al. used a progressive machine learning approach to train a classifier on different software behaviors [24]. They applied their technique in regression testing of a consecutive sequence of minor revisions of a program.

Metamorphic testing (MT) - This is a classic approach to validating software that do not have test oracles. Whenever a formal oracle is not available or costly to apply, test engineers run into a *test oracle problem*. A test oracle is a mechanism against which testers can check the output of a program and decide whether it is correct. When an oracle is not available,

other means of determining whether the test result is correct are known as *pseudo-oracles*. MT operates by checking whether a program under test behaves according to an expected set of properties known as metamorphic relations. A metamorphic relation specifies how a particular change to the input of the program should change the output. MT was used for testing scientific applications in different areas such as machine learning applications [25, 26], bioinformatics programs [27], programs solving partial differential equations [28] and image processing applications [29]. When testing programs solving partial differential equations, MT uncovered faults that cannot be uncovered by special value testing [28].

Learning-based testing – This involves how to adopt the various learning approaches and mechanisms to support testing for AI software. Meinke et al. developed a technique for automatic test case generation for numerical software based on learning-based testing (LBT) [30]. The authors first created a polynomial model as an abstraction of the program under test. Then the test cases are generated by applying a satisfiability algorithm to the learned model.

Crowd-sourced testing – This testing approach uses freelance testers and/or contracted engineers in a crowd sourcing community. It is a cost-effective method to validate a machine-learning based application systems, such as a human face recognition system. Currently, crowd-sourced testing has been used in mobile app testing and mobile TaaS (Testing as a Service). One good example is uTest (http://www.utest.com/company). In addition, crowdsourced test robots will reduce test manual costs and improve test efficiency significantly. However, those robots lack of continuous learning capability. Therefore, the ternary *human-cyber-physical* testing will be focused more and more for AI software in the future.

Data model-based testing – Since big data are the input values for AI software and applications, diverse data models can be used to assist test case generations. Vilkomir et al. presented a method to automatically generate test cases for a scientific program having many input parameters with dependencies [31]. They used a directed graph to model the input data space, including parameters and values as well as their dependencies. Valid test cases can be automatically generated based on the directed graph model. Since their model satisfies the probability law of Markov chains, it can be used to generate random and weighted test cases according to the likelihood of taking the parameter values.

Rule-based software testing – This approach could be used in testing rule-based or knowledge-based systems. The basic idea is to design test cases based on the rules specified in an expert system. Deason et al. in [32] proposed a rule-based test data generation method for Ada programs. They demonstrated that rule-based test data generation is feasible. The paper shows a great promise in assisting test engineers in test generation. Andrews et al. presented a test pattern generation approach based on VHDL specific heuristic rules [33]. Their results indicated rule-based approach leading to a better test coverage.

4. TESTING QUALITY ASSESSMENT AND TEST ADEQUANCY ANALYSIS

Conventional system quality parameters such as performance, robustness, security, etc., can be applicable onto AI software and applications. They are listed below.

- ***System Data Security*** –This parameter could be used to evaluate the security of AI software in different perspectives. Using this parameter**,** data security could be evaluated in various perspectives at the different levels.
- ***System Reliability*** –This parameter is used to evaluate the durability of the system when performing a required function under stated conditions for a specified period of time.
- ***System Robustness*** - This parameter evaluates the ability of a system to resist change without adapting its initial stable configuration.
- ***System Usability*** –This parameter indicate show well AI application service can be used. This can be very subjective due to different developers and users have diverse user experiences.
- ***System Performance***–This is a distinct quality factor for big data-based AI software, and it is useful to evaluate how well big data are collected, generated, trained and tested to support large-scale AI application services.

4.1 AI System Testing Quality Parameters and Quality Assessment

In addition to the system quality parameters presented above, there are a number of quality parameters for AI software from function and feature view. Here we list the typical ones shown in Figure 5.

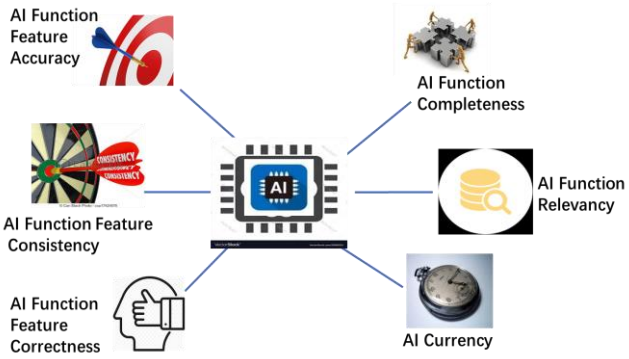

Figure 5 AI Software Testing Quality Parameters

- ***Function Accuracy, which*** is used to evaluate if the system yields true (no systematic errors), and consistent (no random errors) results. Some AI software are developed to find previously unknown answers, thereby only approximate solutions might be available. This can be called uncontrollable prediction. Some prediction is used to prevent something happening in the future, and the prediction result will affect actions or behaviors. In turn, those actions can promote the prediction result.
- ***Function Consistency, which*** *is* a quality indicator useful to evaluate the consistency of the targeted software in different perspectives. Due to the inherent uncertainties in

AI models, some applications do not produce single correct output for a given set on inputs. This leads to hardly determining the expected behaviors of the software. In such situation, domain-specific experts could provide opinions to support system consistency.

- ***Function Correctness, which is a quality*** factor used to evaluate the correctness of AI software. Unlike the conventional system, AI software are hard to validate their correctness. For instance, prediction–related software is mainly developed to make predictions or better understand about real world activities. Hence, it is difficult to determine the correct output for those types of software. Correctness is related to the prediction pattern or model. For instance, some models are more likely used to predict point of inflexion values while some other models are doing well in predicting continuity. Thus, in order to verify the correctness of the software and applications effectively, engineers need to evaluate the capability of prediction under the specified conditions and environments.
- ***Function Currency, which*** is 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.
- ***Function Completeness, which*** *is to measure* there are enough functions to meet business demands in diverse environments or contexts. Are some functions missing, or in an unusable state? Taking an object identification software for example, when some objects in a record cannot be recognized, we say the function is not complete. For another example, every registered citizen must have nationality information. A null value in a citizen nationality record means that a citizen nationality is unknown. Then the personal information in this record is not complete. So, a null value check, which is an attribute check, is a type of completeness validation.

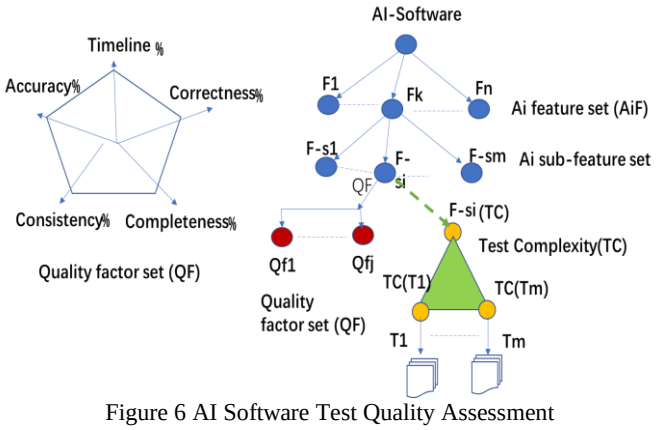

Figure 6 AI Software Test Quality Assessment

Based on the discussed quality parameters above, testing results are analyzed and evaluated for quality assessment. For example, there are five quality factors in the set (QF) here as shown in Figure 6. As we mentioned, AI software have a number of features (F_1,…,F_n), composed of corresponding sub-features(F-s1,…, F-si,…, F-sm). For each measurable feature, we could perform test complexity (TC) analysis. In addition, the quality factors can be measured in terms of pre-defined quality metrics to show their percentage value. Quality Measurement results can be represented using a Radar Chart shown in the left part of Figure 6. Nevertheless, those measurement results need to be validated in practice to indicate their effectiveness.

4.2 AI Software Test Adequacy and Coverage

When AI software can be operated under different context and environments, it must be validated under diverse environments to achieve certain context test criteria for vendors and customers. Thus, engineers need well-defined test criteria and an effective test coverage analysis solution. As we discussed in Section 2.2, constructed diverse test models can be utilized for test coverage analysis. For a knowledge model, AI knowledge test coverage analysis need to be performed; for a feature model, AI features, sub-features, and feature classification need to be analyzed for test coverage; and for a data-based model, data classification, data relation, data format, data range, and etc., need to be addressed for test coverage analysis.

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.

6. CONCLUSIONS

With the fast advance of artificial intelligence technology and data-driven machine learning techniques, how to build high-quality AI software and applications becomes a very hot subject. The special features of AI software bring new challenges and issues for validating software or system quality. Aiming to clarify the primary issues on AI software testing, this paper provides perspective views on AI software validation, including the tutorial concepts, test features and focuses, as well as validation process. Moreover, the primary types of AI software testing and existing validation approaches are analyzed and discussed. The paper also points out the test quality evaluation and coverage problems in AI software. The primary challenges, issues and needs are presented in the end.

REFERENCES

[1] https://www.marketsandmarkets.com/Market-Reports/automation-testing-market-113583451.html

[2] J. Gao, C.L. Xie, and C.Q. Tao. Quality Assurance for Big Data–Issuses, Challenges, and Needs. In Prof. of IEEE 9th International Symposium on Service oriented System Engineering, pp.433-441, 2016.

[3] A.O. Mohammed, S.A. Talab. Enhanced Extraction Clinical Data Technique to Improve Data Quality in Clinical Data Warehouse. International Journal of Database Theory and Application, 8(3): 333-342, 2015.

[4] M. R. Wigan, R. Clake. Big Data's Big Unintended Consequences. IEEE Computer, 46 (6):46-53, 2013.

[5] A.M. Turing, Computing Machinery and Intelligence, Mind, vol. 59, pp. 433-460, 1950.

[6] A. P. Saygin, I. Cicekli, V.Akman. Turing Test: 50 Years Later. Minds & Machines, 10(4):463-518, 2000.

[7] A. Broggi, P.Cerri, S.Debattisti, et al. PROUD—Public Road Urban Driverless-Car Test. IEEE Transactions on Intelligent Transportation Systems, 16(6):3508-3519, 2015.

[8] L. Li, Y. L. Lin, N. N. Zheng, et al. Artificial Intelligence Test: A Case Study of Intelligent Vehicles. Artificial Intelligence Review, (10)3:441-465, 2018.

[9] L. Li , W. L. Huang, Y. Liu, et al. Intelligence Testing for Autonomous Vehicles: A New Approach. IEEE Transactions on Intelligent Vehicles, 1(2):158-166, 2017.

[10] L. Li, D. Wen. Parallel Systems for Traffic Control: A Rethinking. IEEE Transactions on Intelligent Transportation Systems, 17(4):1179-1182, 2016.

[11] L. Li, Y. Lin, N. Zheng, et al. Parallel Learning: a Perspective and a Framework. IEEE/CAA Journal of Automatica Sinica, 4(3):389-395, 2017.

[12] T. J. Ostrand and M. J. Balcer. The Category-Partition Method for Specifying and Generating Fuctional Tests. Communications of the ACM, 31(6): 676-686, 1988.

[13] L. Copeland. A Practitioner's Guide to Software Test Design. Artech House, 2004.

[14] A. Souri, M. E. Akbari, and A. Salehpour. Reduction and Modification of Test Cases in Web Applications by Using Multi Objective Genetic Algorithm. Journal of American Science 8(4):757-762, 2012.

[15] J. Straub and J. Huber. A Characterization of the Utility of Using Artificial Intelligence to Test Two Artificial Intelligence Systems. Computers, (2013)2: 67-87.

[16] W. B. Langdon, S. Yoo, and M. Harman. Inferring Automatic Test Oracles, In Prof. of the IEEE/ACM International Workshop on Search-Based Software Testing (SBST). pp. 5-6, 2017.

[17] X. Y. Zhang, H. B. Gao, M. Guo, G. Li, Y. Liu, and D. Li, A Study on Key Technologies of Unmanned Driving, CAAI Transactions on Intelligence Technology, 1(1):4-13, 2016.

[18] Y. Liu, J. Zhao, Y. Wang, et al. DeepGauge: Multi-granularity Testing Criteria for Deep Learning Systems," In Prof. of the 33rd ACM/IEEE International Conference on Automated Software Engineering(ASE), pp.120-131, 2018.

[19] S. Kovach (18 January 2017). Google Quietly Stopped Publishing Monthly Accident Reports for Its Self Driving Cars. Business Insider. Retrieved 13 June 2018.

[20] M. Last, M. Friedman,and A. Kandel. The Data Mining approach to Automated Software Testing. In Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery andData Mining (KDD), pp. 388–396, 2003.

[21] M. Vanmali, M. Last, and A. Kandel. Using a Neural Network in the Software Testing Process. International Journal of Intelligent Systems, 17 (1): 45–62, 2002.

[22] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang. Automated Support for Classifying Software Failure Reports. In Proc. of the 25th International Conference on Software Engineering (ICSE), pp. 465–475, 2003.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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).

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

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).