Received August 9, 2019, accepted August 19, 2019, date of publication August 23, 2019, date of current version September 9, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2937107

Testing and Quality Validation for AI Software–Perspectives, Issues, and Practices

CHUANQI TAO[1,2,3], JERRY GAO[4], AND TIEXIN WANG[1,2]

[1]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[2]Ministry Key Laboratory for Safety-Critical Software Development and Verification, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[3]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
[4]Department of Computer Engineering, San José State University, San Jose, CA 95192-01809, USA

Corresponding author: Chuanqi Tao (taochuanqi@nuaa.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2018YFB1003900, in part by the National Natural Science Foundation of China under Grant 61402229 and Grant 61602267, in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization, in part by the Fundamental Research Funds for the Central Universities under Grant NS2019058, and in part by the Open Fund of the State Key Laboratory for Novel Software Technology under Grant KFKT2018B19.

ABSTRACT With the fast growth of artificial intelligence and big data computing technologies, more and more software service systems have been developed using diverse machine learning models and technologies to make business and intelligent decisions based on their multimedia input to achieve intelligent features, such as image recognition, recommendation, decision making, prediction, etc. Nevertheless, there are increasing quality problems resulting in erroneous testing costs in enterprises and businesses. Existing work seldom discusses how to perform testing and quality validation for AI software. This paper focuses on quality validation for AI software function features. The paper provides our understanding of AI software testing for new features and requirements. In addition, current AI software testing categories are presented and different testing approaches are discussed. Moreover, test quality assessment and criteria analysis are illustrated. Furthermore, a practical study on quality validation for an image recognition system is performed through a metamorphic testing method. Study results show the feasibility and effectiveness of the approach.

INDEX TERMS
AI software quality validation, AI testing, testing AI software.

I. INTRODUCTION

With the fast advance of big data analytics and AI technologies, numerous AI-based software and applications have been widely accepted and used in people's daily life. AI 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 are classified such as natural language processing systems, object recognition systems, recommendation systems, unman-controlled vehicles and so on. Therefore, how to perform quality validation for AI software becomes a critical concern and research topic from both academic and industrial focuses. 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). Based on recent testing experiences from industry on AI applications such as intelligent mobile 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.

The associate editor coordinating the review of this article and approving it for publication was Honghao Gao.

- 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 above cause new difficulties and challenges in testing and quality validation. Therefore, AI quality validation and assurance becomes a critical concern and a hot research subject. Although there have been many published papers addressing data quality and quality assurance in the past [2]–[4], seldom researches focus on validation for AI software from function or feature view. There is an emergent need in current research to quality validation 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 are fit or use, assuring AI functional features' adequate quality and AI software's QoS (quality of system service) parameters [41], [43]. 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. Typical issues of quality assurance and validation for AI software and applications are listed below.

- How to perform quality assurance for big data which could be utilized as training data or testing data for intelligent algorithms?

- How to make quality validation for application service, e.g. what is the precision of the recommendation service?

- How to validate the quality of diverse intelligent algorithms and models, such as data mining and machine learning methods.

This paper is written to provide our perspective views on AI software (specific to feature or function) testing for quality validation. The paper is organized as follows. Section II discusses the tutorial concepts about AI software testing, including test focuses, features, and requirements. Section III reviews AI-based machine testing, AI software function testing, as well as the existing testing methods potentially-used for AI software validation. Section IV discusses AI software testing quality parameters and evaluation as well as test coverage analysis. Section V presents case studies on an image recognition system using the proposed quality validation approach. The conclusion remarks are in Section VI.

II. 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 is 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.

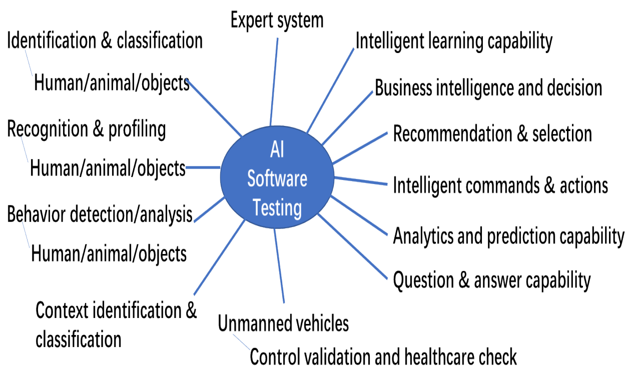
AI Software Testing
Expert system
Identification & classification
Human/animal/objects
Recognition & profiling
Human/animal/objects
Behavior detection/analysis
Human/animal/objects
Context identification & classification
Unmanned vehicles
Control validation and healthcare check
Intelligent learning capability
Business intelligence and decision
Recommendation & selection
Intelligent commands & actions
Analytics and prediction capability
Question & answer capability

FIGURE 1. **The scope of AI software testing.**

*"**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 need 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.

A. TEST SCOPE AND MAJOR FOCUSES

Since AI software is 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. Fig. 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 [35], [36], [45], intelligent commands and actions, analytics and prediction capability [37], [38], [40], [46], 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 [35], 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.

B. 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 feedback 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 graphs 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/

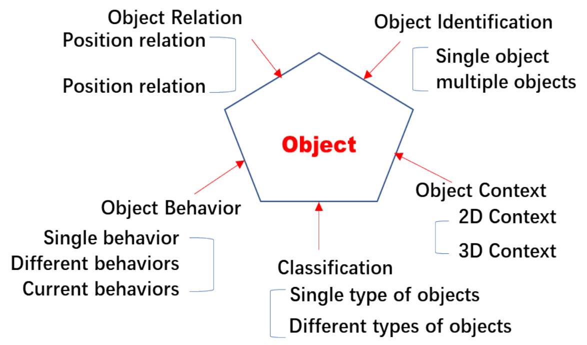

FIGURE 2. A sample object model-based AI software.

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 classify 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 a knowledge test model, feature test model, object test model, and data test model. As shown in Fig. 2, features of object relation, object identification, object behavior, object classification, and object context are selected for function testing with diverse sub-features.

In general, AI software needs 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.

III. 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. Moreover, test quality evaluation and test adequacy analysis are illustrated.

A. 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].

B. 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 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 tests from four aspects and presented an example of how to design intelligence tests 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 combinations 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 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.

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 the 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.

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, the traditional scenario analysis method is applied to analyze the scenarios of applications and test whether the main functions are implemented correctly from the perspective of the scene. Table 1 shows a description of five scenarios in testing *Siri*.

Based on the analyzed results and testing experiences, we conclude that the test cases designed by scenario analysis are 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 the 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

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

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.

C. 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 [39], [42], [47]. 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 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 the AI-based method and the 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 the 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.

AI software testing differs from AI-based software testing in diverse views such as test objectives, test focuses, test scope, test coverage as well as test techniques and tools. For example, AI-based testing primarily aims to increase efficiency for a test process, reduce testing costs by reduce human operations, and increase bug detection effectiveness and speed. AI testing aims to provide on-demand testing services for AI software to support software validation and quality engineering process. AI-based testing majorly focuses on test selection, automatic test execution, bug detection and prediction based large-scale testing history data and AI techniques. In addition, AI testing needs innovative continuous, timeliness, and currency testing techniques.

D. 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 work. The development of machine intelligence is still far from mimicking the cognitive competence of the human brain. It is still challenging to deal with those data effectively and making a driving decision accurately and quickly [17]. Machine learning sometimes returns 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 in 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. Since Machine Intelligence or Deep Learning depends on the training data, the accuracy and quality of data play a vital role for public safety using machine learning in autonomous vehicles.

Many kinds of research attempt to find solutions for the current obstacles of Machine Learning Systems. To draw optimal decision making, approaches such as Fault Tree Analysis, Fuzzy Logic, Metaheuristic Algorithm, and Artificial Neural Network are developed to test with a huge amount of training data by using different algorithms. However, the sufficiency and versatility of Deep Learning systems are based on the accuracy of the test data set. It is difficult to provide adequate support due to the accessibility of test data quality issue. The current Deep Learning systems have 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 decisions rely on the training data. The recent study shows two major vulnerabilities in Deep Learning systems: Software quality from the output of Deep Learning alone is not adequate; and Failure in unseen attacks even though Deep Learning is 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.

E. TYPICAL VALIDATION APPROACHES FOR AI SOFTWARE

AI software testing could be performed using the following approaches from different perspectives.

- **Classification-based AI software testing**, in which classification models for test inputs, contexts, and outputs and events are set up to ensure the adequate testing coverage of diverse input data classes, classified contexts and conditions, and corresponding outputs and classes [20]–[24].
- **Model-based AI software testing**, in which selected intelligent learning models and data models are extended to be traceable and testable AI test models to facilitate AI software testing and operations in quality assessment of training data and test data.
- **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 [25]–[28]. The key element of **metamorphic testing** (MT) is a set of Metamorphic Relations (MRs), which are necessary features of the target function or algorithm in relation to multiple inputs and their expected outputs.
- **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 [30].
- **Rule-based AI software testing**, in which pre-defined expert-based rules are established and used in AI test generation and validation [32], [34].

Nevertheless, how to utilize the existing traditional or intelligent approaches to AI software testing is still a great challenge currently.

F. DATA QUALITY VALIDATION FOR AI-BASED SOFTWARE

In recent years, data (such as image and video image) quality assessment has attracted significant attention. Besides, the quality of big image/video datasets with labeled also have an important impact on machine learning algorithms, such as deep learning. Using a deep learning approach to train artificial AI programs based on annotated training data sets is a popular way to develop intelligent software using a supervised learning approach. With the increasing installation of video cameras in many cities, image data quality assessment is becoming a very hot research topic in computer vision and smart cities.

There are a number of causes affecting the quality of image data [48], [49], such as sharpness, noise, tone reproduction, contrast, distortion, etc. Thus, the typical image quality factors are listed as accuracy, accessibility, readability and understandability, consistency [44], etc.

According to the recent 2018 IEEE NAVIDA AI City challenge [33], manually generating annotated data sets based on image datasets from city street transportation cameras bring diverse data quality issues in a deep learning process. Their case study result clearly indicates that the accuracy and quality of derived AI city transportation programs using a deep learning approach highly depends on the quality of annotated training data sets. Based on their experience report, all of the challenge teams encountered diverse data quality issues in annotated training datasets. And they also discovered the urgent needs in quality validation models, methods, and automatic tools for annotated datasets although there are numerous data validation tools for structure data. Therefore, the key issues of quality assurance for big data applications are how to validate unstructured data quality and how to validate system quality in terms of various quality factors.

Data quality validation and services in a deep learning process for AI software has three dimensions. They are shown as follows.

- ***Raw data quality checking***, which refers to the quality checking process and activities for collected raw data, such as camera-generated images, and videos. The primary objective is to perform raw data cleaning, quality monitoring, and evaluation to ensure high-quality raw data could be collected.
- ***Training data quality validation***, which refers to quality validation processes and activities for manually or semi-automatically generated training data sets, such as annotated data sets. Its objective is to improve the generation of training data quality in a deep learning process to increase the training quality for an underlying AI software. The typical concerns include: a) training data scope and coverage, b) training data classification, c) training data quality, and d) training data coverage.
- ***Test data quality evaluation,*** which refers to test data quality evaluation based on the validation results of a targeted domain-specific application. For a machine learning application system, the major focus of this task should be facilitating AI system quality problem detection, defect improvement, training quality coverage and domain-based knowledge modeling issues for AI systems.

IV. TESTING QUALITY ASSESSMENT AND ADEQUACY ANALYSIS

A. TESTING QUALITY PARAMETERS AND QUALITY ASSESSMENT FOR AI SOFTWARE

Like conventional software quality testing, quality parameters such as performance, robustness, security, etc., can be applicable to AI software and applications. In addition to the system quality parameters, we must pay attention to specific quality parameters for AI software functions and features. Sample quality parameters for image recognition software are presented as follows.

- ***Correctness –*** This quality factor reflects if the recognition result is true when faced with Boolean recognition

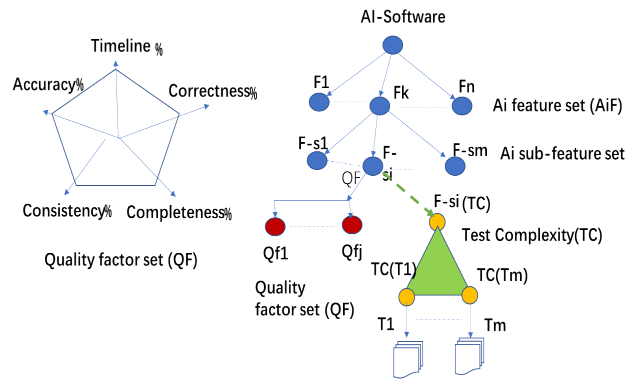

FIGURE 3. AI software test quality assessment.

items, such as gender, buy or not, recommend or not, age group, etc.

- ***Accuracy*** – This reflects the accuracy of the recognition result when faced with numerical recognition items, such as age, gender, and color. Different math index can be used to measure it, such as mean difference, variance, standard deviation, distribution interval, confidence level, absolute mean or relative mean.
- ***System Stability*** – This reflects the stability of the recognition systems. For example, to recognize the same thing twice or more times, the result should be stable.
- ***Timeliness*** – This reflects some indicators related to time, such as the recognition time, training time, and classify time.
- ***Recognition Ratio*** – This reflects the recognition ratio of the image system, such as the perfect recognition ratio which means the system recognizes the picture well, or recognition ratio which is divided by absolute mean or relative mean.
- ***System Robustness*** – This parameter indicates the robustness of the system. For example, when performing special operations on the recognized picture, we need to check whether the system can still recognize it well. The transformation includes overturning, mirror image, enlarging or shrinking, shearing, shear, gray scale, and changing the dpi.
- ***Image Quality*** – This checks whether the recognition systems can deal with the changing of the quality attribute of image, such as gauss noise, spiced salt noise due to the unreliable network transmission, etc.

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 Fig. 3. 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 Fig. 3. Nevertheless, those measurement results need to be validated in practice to indicate their effectiveness.

B. AI SOFTWARE TEST ADEQUACY AND COVERAGE

When AI software can be operated under different contexts 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 II, diverse test models can be constructed and 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, etc., need to be addressed for test coverage analysis.

V. CASE STUDIES- QUALITY VALIDATION FOR ROBUSTNESS OF AN IMAGE RECOGNITION APPLICATION

We performed case studies to indicate the feasibility and effectiveness of the proposed quality validation approach provided in this paper. Here we selected a face recognition system as the study object. We performed a case study on a realistic AI application system- "Alibaba Cloud Computing Services Facial Age Recognition API" provided by Alibaba Company using the metamorphic testing method. The *base64* encoding of images is submitted to APIs, and the system returns with the recognition results. The experiment data sets are selected from the *wiki_crop.tar* in the open face dataset IMDB-WIKI. There are total of 52444 face data, and 10K images are selected randomly as experimental data sets.

A. QUALITY VALIDATION METHOD DESIGN

The designed quality validation method is based on the robustness of the age recognition system: The recognition result is deemed better when the real age and recognition age are closer to each other. Facial age recognition is a commonly-used AI application using diverse machine learning algorithms and pattern recognition strategies. There are existing non-oracle problems and due to the effect of picture quality (such as clarity, lighting, background, and expression), network or other reasons, the robustness of an age recognition system is a basic quality factor in quality assurance. Thereby we need to test the robustness of the system. Based on the understanding of facial age recognition system above, we adopt metamorphic testing to validate the quality of the system. We consider the possible situations that may occur in a recognition process, such as image rotation, translation, landscaping, a watermark of a picture, or the distance between face and camera.

In this study, we defined two major metamorphic relations MR1 and MR2. For each metamorphic relation, we define several sub-relations. For instance, in MR1, we give two sub-relations MR1-1 and MR1-2, i.e., a) recognized age is

TABLE 2. Metamorphic relation case partition.

metamorphic relationship	sub-metamorphic relationship	cases	sub-cases division	No. of cases
MR1	MR1-1 (mirror)	1000	mirror	1000
	MR1-2 (rotation)	12000	+5°	1000
			-5°	1000
			+10°	1000
			-10°	1000
			+15°	1000
			-15°	1000
			+20°	1000
			-20°	1000
			+25°	1000
			-25°	1000
			+30°	1000
			-30°	1000
MR2	MR2-1(translation)	12000	X20Y30	1000
			X40Y50	1000
			X50Y80	1000
			X70Y70	1000
			X80Y80	1000
			X100Y100	1000
			X130Y130	1000
			X150Y150	1000
			X-50Y-50	1000
			X-100Y-100	1000
			X-100Y100	1000
			X100Y-100	1000
	MR2-2(tailor)	4000	3:4	1000
			4:3	1000
			9:16	1000
			16:9	1000
	MR2-3(scaling)	15000	400*300	1000
			640*480	1000
			320*240	1000
			800*600	1000
			1024*600	1000
			100*100	1000
			200*200	1000
			300*300	1000
			400*400	1000
			500*500	1000
			600*600	1000
			700*700	1000
			800*800	1000
			900*900	1000
			1000*1000	1000

stable under the spherical transformation (mirror), and b) recognized age is stable under image rotation. In the study, we verified if the image system under testing satisfies the defined MRs. The detailed metamorphic relations and their sub-cases are shown in Table 2. The proposed metamorphic relations are illustrated as follows.

MR1: When the size and color are unchanged and we do some simple rotation transformation of the image, the recognition age of before and after the operation should stay the same.

MR2: When the color is unchanged and we do some change on the size of the image but not affect the face part, the recognition age of before and after the operation should stay the same.

The corresponding sub-metamorphic relations for MR1 and MR2 are listed as follows.

MR1-1: When we perform image mirror transformation, the recognition age of before and after the operation should stay the same.

MR1-2: When we perform image rotation transformation based on degrees, the recognition age of before and after the operation should stay the same.

MR2-1: When we perform image translation, the recognition age of before and after the operation should stay the same.

MR2-2: When we perform image tailor, the recognition age of before and after the operation should stay the same.

MR2-3: When we perform image scaling, the recognition age of before and after the operation should stay the same.

In addition, due to non-oracle problems in big data intelligence systems, we propose some quantitative quality evaluation metrics to analyze the metamorphic testing results. For instance, recognition accuracy is divided into two statistical indicators: *average absolute error* (AAE), *average relative error* (ARE). They are illustrated below.

- *Average absolute error*. It represents the error between real age and recognition age. When the value is lower, the closer the actual age and the age of recognition are, and the better the recognition effect is. The formula is below.

$$R_1 = \sum abs(\text{age1} - age2) \Big/ N \tag{1}$$

where *age1* and *age2* represents the real age and recognition age respectively, and N refers to the total cases that pass the recognition system.

- *Average relative error*. Here we use ARE to avoid the side-effect of age size. The formula is as follows.

$$R_2 = \sum \frac{abs(\text{age1} - age2)}{age1} \Big/ N \tag{2}$$

where *age1* and *age2* represents the real age and recognition age respectively, and N refers to the total cases that pass the recognition system.

B. STUDY RESULTS AND ANALYSIS

MR1-AAE Result: From Fig. 4 we can see the AAE value of MR1-1 is around 3, and AAE of MR1-2 depends on the rotation degrees. When the degree is larger, the absolute error is bigger. The rotation direction does not affect the study result significantly.

MR1-ARE Result: Fig. 5 shows that the relative error of MR1-1 is about 5%, and the relative error of MR1-2 depends on the rotation degrees. When the degree is larger, the relative error is bigger. The rotation direction seems to do not affect the result.

MR2-AAE Result: Fig. 6-8 shows the study result of AAE of MR2-1, MR2-2, and MR2-3 respectively. AAE value of MR2-1 and MR2-2 is relatively low and did not change significantly. In Fig. 8, we discover that when performing image uniform scaling, the AAE result is stable. However,

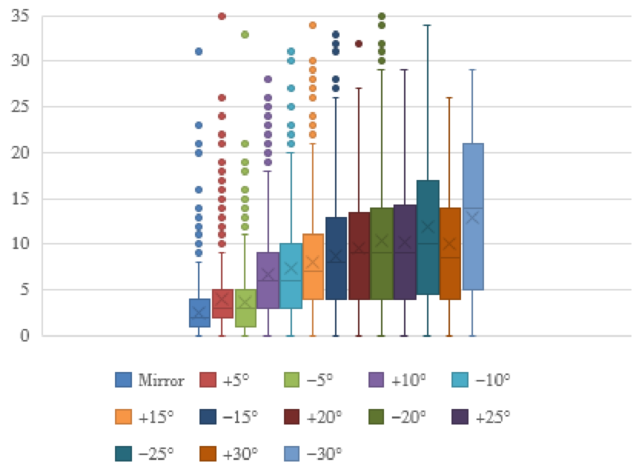

FIGURE 4. Study results of AAE in MR1.

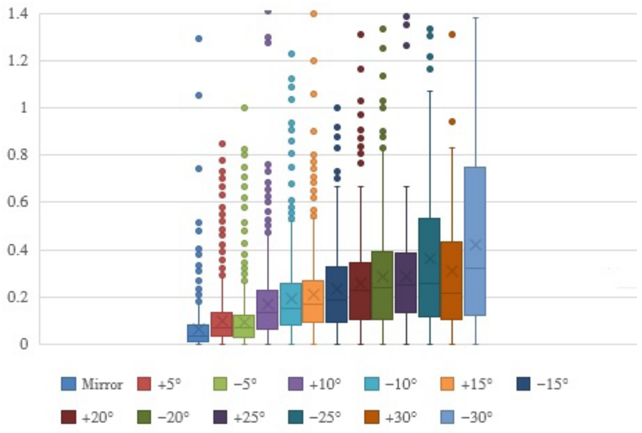

FIGURE 5. Study results of ARE in MR1.

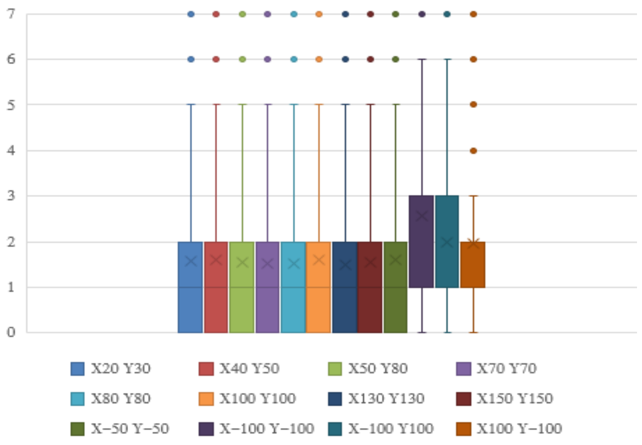

FIGURE 6. Study results of AAE in MR2-1.

when we perform image non-uniform scaling, the AAE result changes significantly.

MR2-ARE Result: Fig. 9-11 shows the study result of AAE of MR2-1, MR2-2, and MR2-3 respectively. ARE of MR2-1 and MR2-2 is relatively low and did not change

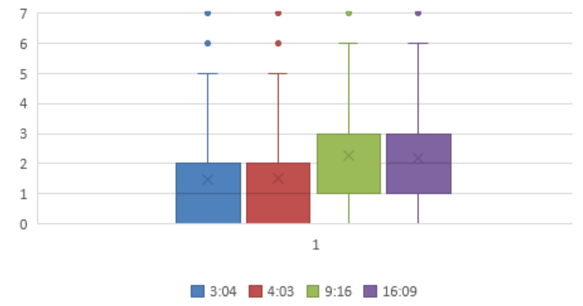

FIGURE 7. Study results of AAE in MR2-2.

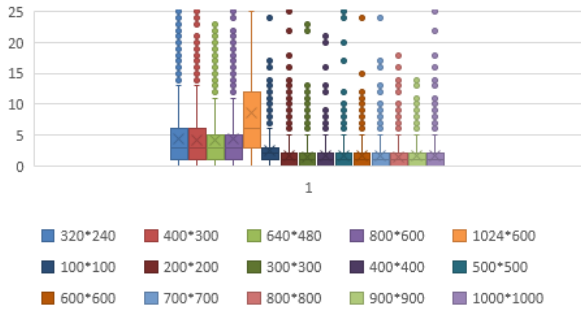

FIGURE 8. Study results of AAE in MR2-3.

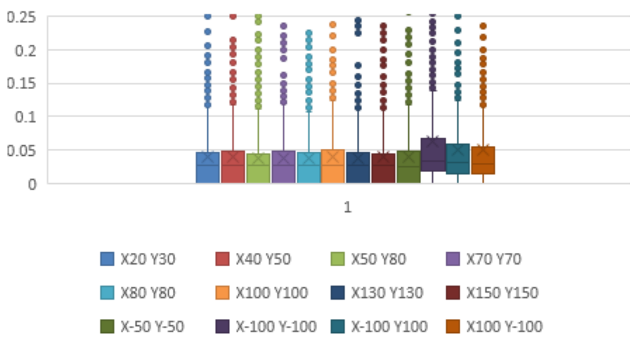

FIGURE 9. Study results of ARE in MR2-1.

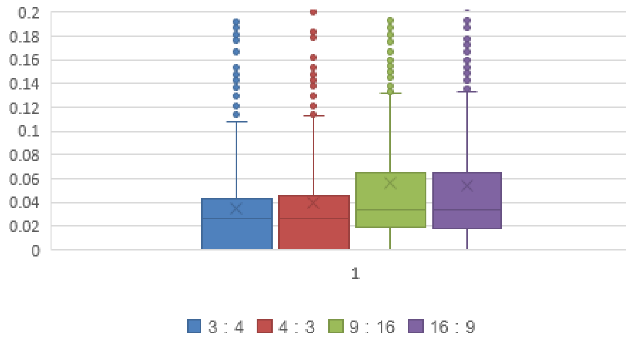

FIGURE 10. Study results of ARE in MR2-2.

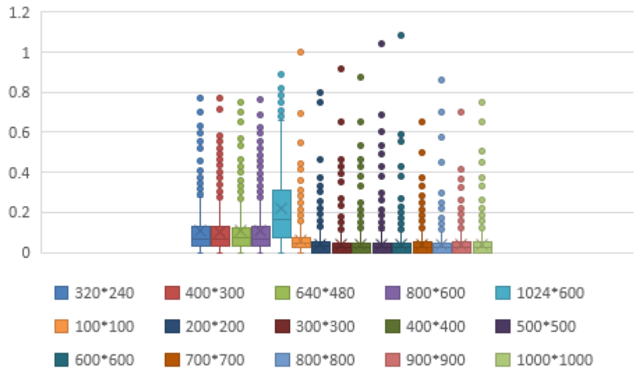

FIGURE 11. Study results of ARE in MR2-3.

significantly. Similar to MR2-AAE result, we discover that when performing image uniform scaling, the AAE result is stable, and when we perform image non-uniform scaling, the AAE result changes significantly.

In general, the studied system shows good performance in robustness. When performing image rotation, the AAE and ARE of the system become larger with the increasing of the rotation angle. Nevertheless, the system robustness is good if the rotation angle is kept unchanged regardless of rotation direction. According to our discussion with face recognition system developers, most of the current recognition algorithms could not deal with skew face issues (e.g., image rotation) very well. The system shows good robustness in image translation and tailor. However, when performing image non-uniform scaling, AAE and ARE of the system increase significantly. The possible reason is that image non-uniform scaling might make the facial features of each face image (such as the face, eyebrow spacing, etc.) occur large deformation, resulting in the recognition bugs of adopted facial feature recognition algorithms. In addition, scaling images may also cause picture distortion problems.

VI. CONCLUSION

With the fast advance of artificial intelligence technologies 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 quality validation, including the tutorial concepts, test features and focuses, as well as a 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. Furthermore, case studies on practical AI software applications are performed to indicate the feasibility and effectiveness of the proposed quality validation approach.

REFERENCES

[1] *Automation Testing Market by Technology (IoT, AI, and Big Data), Testing Type (Functional, Performance, Compatibility, and Security), Service (Advisory & Consulting, Managed, and Implementation), Endpoint Interface, and Region—Global Forecast to 2023*. Accessed: Jun. 18, 2019. [Online]. Available: 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-issuss, challenges, and needs," in *Proc. IEEE 10th Int. Symp. Service Oriented Syst. Eng.*, Apr. 2016, pp. 433–441.
[3] A. O. Mohammed and S. A. Talab, "Enhanced extraction clinical data technique to improve data quality in clinical data warehouse," *Int. J. Database Theory Appl.*, vol. 8, no. 3, pp. 333–342, 2015.
[4] M. R. Wigan and R. Clake, "Big data's big unintended consequences," *Computer*, vol. 46, no. 6, pp. 46–53, 2013.
[5] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, Oct. 1950.
[6] A. P. Saygin, I. Cicekli, and V. Akman, "Turing test: 50 years later," *Minds Mach.*, vol. 10, no. 4, pp. 463–518, 2000.
[7] A. Broggi, P. Cerri, S. Debattisti, M. C. Laghi, P. Medici, D. Molinari, M. Panciroli, and A. Prioletti, "PROUD—Public road urban driverless-car test," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 6, pp. 3508–3519, Dec. 2015.
[8] L. Li, Y.-L. Lin, N.-N. Zheng, F.-Y. Wang, Y. Liu, D. Cao, K. Wang, and W.-L. Huang, "Artificial intelligence test: A case study of intelligent vehicles," *Artif. Intell. Rev.*, vol. 10, no. 3, pp. 441–465, 2018.
[9] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, "Intelligence testing for autonomous vehicles: A new approach," *IEEE Trans. Intell. Veh.*, vol. 1, no. 2, pp. 158–166, Jun. 2016.
[10] L. Li and D. Wen, "Parallel systems for traffic control: A rethinking," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1179–1182, Apr. 2016.
[11] L. Li, Y. Lin, N. Zheng, and F.-Y. Wang, "Parallel learning: A perspective and a framework," *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 3, pp. 389–395, Jul. 2017.
[12] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," *Commun. ACM*, vol. 31, no. 6, pp. 676–686, 1988.
[13] L. Copeland, *A Practitioner's Guide to Software Test Design*. Norwood, MA, USA: 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," *J. Amer. Sci.*, vol. 8, no. 4, pp. 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*, vol. 2, no. 2, pp. 67–87, 2013.
[16] W. B. Langdon, S. Yoo, and M. Harman, "Inferring automatic test oracles," in *Proc. IEEE/ACM 10th Int. Workshop Search-Based Softw. Test. (SBST)*, May 2017, pp. 5–6.
[17] X. Zhang, H. Gao, M. Guo, G. Li, Y. Liu, and D. Li, "A study on key technologies of unmanned driving," *CAAI Trans. Intell. Technol.*, vol. 1, no. 1, pp. 4–13, 2016.
[18] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "DeepGauge: Multi-granularity testing criteria for deep learning systems," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2018, pp. 120–131.
[19] S. Kovach, *Google Quietly Stopped Publishing Monthly Accident Reports for Its Self Driving Cars*. New York, NY, USA: Business Insider, 2018.
[20] M. Last, M. Friedman, and A. Kandel, "The data mining approach to automated software testing," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2003, pp. 388–396.
[21] M. Vanmali, M. Last, and A. Kandel, "Using a neural network in the software testing process," *Int. J. Intell. Syst.*, vol. 17, no. 1, pp. 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. 25th Int. Conf. Softw. Eng. (ICSE)*, May 2003, pp. 465–475.
[23] P. Francis, D. Leon, M. Minch, and A. Podgurski, "Tree-based methods for classifying software failures," in *Proc. 15th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2004, pp. 451–462.
[24] J. F. Bowring, J. M. Rehg, and M. J. Harrold, "Active learning for automatic classification of software behavior," in *Proc. ACM SIGSOFT Int. Symp. Softw. Test. Anal. (ISSTA)*, 2004, pp. 195–205.
[25] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *J. Syst. Softw.*, vol. 84, no. 4, pp. 544–558, 2011.
[26] C. Murphy, G. E. Kaiser, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *Proc. 20th Int. Conf. Softw. Eng. Knowl. Eng. (SEKE)*, 2008, pp. 867–872.
[27] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC Bioinf.*, vol. 10, p. 24, Jan. 2009.
[28] T. Y. Chen, J. Feng, and T. H. Tse, "Metamorphic testing of programs on partial differential equations: A case study," in *Proc. 26th Annu. Int. Comput. Softw. Appl. (CMPSAC)*, Aug. 2002, pp. 327–333.
[29] J. Mayer and R. Guderlei, "On random testing of image processing applications," in *Proc. 6th Int. Conf. Qual. Softw. (QSIC)*, Oct. 2006, pp. 85–92.
[30] K. Meinke and F. Niu, "A learning-based approach to unit testing of numerical software," in *Proc. 22nd IFIP WG 6.1 Int. Conf. Test. Softw. Syst.*, 2010, pp. 221–235.
[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 *Proc. 8th Int. Conf. Comput. Sci.*, 2008, pp. 291–300.
[32] W. H. Deason, D. B. Brown, K.-H. Chang, and J. H. Cross, II, "A rule-based software test data generator," *IEEE Trans. Knowl. Data Eng.*, vol. 3, no. 1, pp. 108–117, Mar. 1991.
[33] M. Naphade, D. C. Anastasiu, A. Sharma, V. Jagrlamudi, H. Jeon, K. Liu, M.-C. Chang, S. Lyu, and Z. Gao, "The NVIDIA AI City Challenge," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug. 2017, pp. 1–6.
[34] A. Andrews, A. O'Fallon, and T. Chen, "A rule-based software testing method for VHDL models," in *Proc. VLSI-SOC*, vol. 92, 2003, pp. 1–6.
[35] Y. Yin, L. Chen, Y. Xu, and J. Wan, "Location-aware service recommendation with enhanced probabilistic matrix factorization," *IEEE Access*, vol. 6, pp. 62815–62825, 2018.
[36] X. Sun, H. Yang, X. Xia, and B. Li, "Enhancing developer recommendation with supplementary information via mining historical commits," *J. Syst. Softw.*, vol. 134, pp. 355–368, Dec. 2017.
[37] H. Gao, K. Zhang, J. Yang, F. Wu, and H. Liu, "Applying improved particle swarm optimization for dynamic service composition focusing on quality of service evaluations under hybrid networks," *Int. J. Distrib. Sensor Netw.*, vol. 14, no. 2, pp. 1–14, 2018.
[38] Y. Yin, W. Xu, Y. Xu, H. Li, and L. Yu, "Collaborative QoS prediction for mobile service with data filtering and SlopeOne model," *Mobile Inf. Syst.*, vol. 2017, Jun. 2017, Art. no. 7356213.
[39] X. Sun, X. Peng, K. Zhang, Y. Liu, and Y. Cai, "How security bugs are fixed and what can be improved: An empirical study with Mozilla," *Sci. China Inf. Sci.*, vol. 62, no. 1, 2019, Art. no. 19102.
[40] Y. Yin, S. Aihua, G. Min, X. Yueshen, and W. Shuoping, "QoS prediction for Web service recommendation with network location-aware neighbor selection," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 4, pp. 611–632, 2016.
[41] H. Gao, W. Huang, X. Yang, Y. Duan, and Y. Yin, "Toward service selection for workflow reconfiguration: An interface-based computing solution," *Future Gener. Comput. Syst.*, vol. 87, pp. 298–311, Oct. 2018.
[42] X. Sun, T. Zhou, G. Li, J. Hu, H. Yang, and B. Li, "An empirical study on real bugs for machine learning programs," in *Proc. 24th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2017, pp. 348–357.
[43] H. Gao, S. Mao, W. Huang, and X. Yang, "Applying probabilistic model checking to financial production risk evaluation and control: A case study of Alibaba's Yu'e Bao," *IEEE Trans. Comput. Social Syst.*, vol. 5, no. 3, pp. 785–795, Sep. 2018.
[44] H. Gao, Y. Duan, H. Miao, and Y. Yin, "An approach to data consistency checking for the dynamic replacement of service process," *IEEE Access*, vol. 5, pp. 11700–11711, 2017.
[45] L. Qi, W. Dou, W. Wang, G. Li, H. Yu, and S. Wan, "Dynamic mobile crowdsourcing selection for electricity load forecasting," *IEEE Access*, vol. 6, pp. 46926–46937, 2018.
[46] L. Qi, X. Zhang, W. Dou, and Q. Ni, "A distributed locality-sensitive hashing-based approach for cloud service recommendation from multi-source data," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2616–2624, Nov. 2017.
[47] X. Sun, X. Peng, H. Leung, and B. Li, "ComboRT: A new approach for generating regression test cases for evolving programs," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 6, pp. 1001–1026, 2016.
[48] Y. Weng, T. Zhou, L. Liu, and C. Xia, "Automatic convolutional neural architecture search for image classification under different scenes," *IEEE Access*, vol. 7, pp. 38495–38506, 2019.
[49] Y. Weng, T. Zhou, Y. Li, and X. Qiu, "NAS-Unet: Neural architecture search for medical image segmentation," *IEEE Access*, vol. 7, pp. 44247–44257, 2019.

CHUANQI TAO received the Ph.D. degree in software engineering from the School of Computer Science and Engineering, Southeast University, in 2013. From August 2010 to September 2011, he was a Visiting Scholar with San José State University, CA, USA. He is currently an Associate Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include intelligent software testing, regression testing, cloud-based mobile testing as a service, and quality assurance for big data applications. He has published over 30 articles in the software testing and quality assurance domain in IEEE journals, international conferences, and workshops.

JERRY GAO is currently a Full Professor with the College of Computer Engineering, San José State University, USA. He had over 15 years of academic research and teaching experience and over ten years of industry working and management experience on software engineering and IT development applications. He has published over 100 (180) publications in IEEE/ACM journals, magazines, international conferences, and workshops. His research interests include big data testing and quality validation, cloud-based mobile testing, and smart city.

TIEXIN WANG received the bachelor's and master's degrees from the Harbin Institute of Technology (HIT), in 2010 and 2012, respectively, the another master's degree from University of Bordeaux, in 2012, and the Ph.D. degree from the École des Mines d'Albi, France, supervised by Prof. F. Benaben, in October 2015. He did one year Postdoctoral research at ARMINES, France. He is currently with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include enterprise engineering, collaboration management, model transformation, and semantic checks. He has published around 20 research articles. He used to take part in a H2020 Project "Cloud Collaborative Manufacturing Networks: C2Net."