International Telecommunication Union

ITU-T FG-AI4H Deliverable

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

16 March 2023

PRE-PUBLISHED
VERSION

DEL7.2

Artificial intelligence technical test specification

Summary

Deliverable 7.2 specifies how an AI can and should be tested in silico. Among other aspects, best practices for test procedures known from (but not exclusively) AI challenges are being reviewed in this document. Important testing paradigms that are not exclusively related to AI applications are mentioned too.

Keywords

Artificial intelligence, assessment platform, bias, data preparation, health data, machine learning, metrics, performance, software testing, test specification, training

Change Log

This document contains Version 1 of Deliverable DEL7.2 on "*Artificial intelligence technical test specification*" approved on 16 March 2023 via the online approval process for the ITU-T Focus Group on AI for Health (FG-AI4H).

Editor: Auss Abbood
Robert Koch Institute
Germany

E-mail: AbboodA@rki.de

CONTENTS

List of Tables

ITU-T FG-AI4H Deliverable DEL7.2

Artificial intelligence technical test specification

1 Scope

This document is part of WG-DAISAM's endeavour to produce an assessment platform for the Focus Group AI4H. An assessment platform is meant to be a software framework that allows user from different topic groups to assess AIs in a transparent and standardized fashion on (real) health data. While such an assessment platform describes a pipeline of several diverse operations that need to be described, this document focusses on the testing phase, i.e., the part where the AI's performance is tested and evaluated in a numerical and/or graphical way. This includes the training of the AI on data that is assumed to be cleared of biases and represents the problem-to-solve appropriately. Data preparation is outside the scope of this document and can be found in FG-AI4H Deliverable 5. The test specification then includes the submission of this trained model to the assessment platform, its run on the undisclosed test data, and the calculation of the performance metrics. These metrics need to give a representative estimate of the AI's performance and need to be well adjusted to detect non-robust, biased, or otherwise flawed AI's. How these metrics are defined and chosen is outside the scope of this document and more closely discussed in deliverable 7.3.

2 References

[DEL0.1] ITU-T FG-AI4H DEL0.1 (2022), *Common unified terms in artificial intelligence for health*.

[DEL5] ITU-T FG-AI4H DEL05, *Data Specification*.
https://extranet.itu.int/sites/itu-t/focusgroups/ai4h/Deliverables/DEL05.docx

[DEL7.3] ITU-T FG-AI4H DEL7.3, *ML4H trial audits–Iteration 2.0 Playbook (Version 3.0)*.
https://extranet.itu.int/sites/itu-t/focusgroups/ai4h/Deliverables/DEL07_3.docx

3 Terms and definitions

3.1 Terms defined elsewhere

This document uses the terms defined in [DEL0.1].

3.2 Terms defined here

This document does not define new terms.

4 Abbreviations

AI	Artificial Intelligence
AI4H	Artificial Intelligence for Health
DAISAM	Data and AI Solution Assessment
FG-AI4H	ITU/WHO Focus Group on Artificial Intelligence for Health
ISTQB	International Software Testing Qualifications Board
ML	Machine Learning
TDD	Topic Description Document

TG	Topic Group
WG	Working Group

5 State of the art in testing

Software testing has been a crucial part of software development for several decades. *The art of software testing* by Glenford J. Myers is one such standard reference that was one the first examples on how to define software testing which was previously part of software debugging, the removal of errors in the code. While debugging is the process of eradicating errors in software, tests help detecting them. The International Software Testing Qualifications Board (ISTQB) is a not-for-profit association that certifies competences in software testing and has also published a document that defines and describes testing in depth.

5.1 Testing principles

Before different established tests are introduced, we may wish to look at the high-level idea of testing that can be summarized by seven principles that were established since the emergence of software testing and can be found in ISTQB’s syllabus for testing software: First, testing shows the presence of errors, not their absence. Second, exhaustive testing is impossible which is important to keep in mind when we discuss testing of AI. Third, early testing saves time. Fourth, errors cluster together, i.e., a small fraction of the software contains most of the defects discovered through testing. This is helpful to optimize testing such that we want to predict and/or find a cluster and focus our testing efforts to a module of the software that contains such a cluster. Fifth, new tests and test data needs to be included into testing, and test data and tests need to be change regularly to detect new defects. Otherwise, testing suffers from the pesticide paradox, i.e., tests are no longer effective at finding defects. Sixth, testing is dependent on the context. E.g., a recreational smartphone app requires another intensity of testing compared to a health AI. Seventh, absence-of-errors does not mean that the user’s requirements are met, the software system is better than a competing product, or that the software is convenient to use.

5.2 Test levels

Testing can be divided into several levels of testing and forms of testing. Testing levels are *component testing* (often referred to as unit testing), integration testing, system testing, and acceptance testing. The lowest level of testing is component testing and tests the functionality of the components of software which are functions, classes, etc. The overall software might still not be operational, but unit testing assures that the components a product is composed of are working as intended. *Integration testing* tests the interactions between the prior tested components and units or between systems such as microservices or packages. Such larger systems are tested using *system testing*. A system can be the aforementioned microservices but it might also be the overall end-to-end application that either consists of several components or several smaller systems. The highest level of testing is acceptance testing and is similar to system testing in that it tests the whole end-to-end application but it does not look for errors in the sense of software development. This step is rather used to verify that the system behaves as intended according to customer, system administrators, contracts, regulators, etc.

5.3 Test types

Table 1 contains the several forms of testing. All forms can in general be combined with the aforementioned levels of testing. However, Table 1 contains a suggestion on which combinations might be more appropriate. More information on testing can be found in ISO/IEC/IEEE 29119, or BS 7925.

Table 1 – Forms of testing

Test Type	Explanation
Functional Testing	Tests *what* the system should do by specifying some precondition, running code and then compare the result of this execution with some postcondition. It is applied at each level of testing although in acceptance testing most implemented functions should already work. Coverage is a measure of thoroughness of functional testing.
Non-functional Testing	Test how *well* a system performs. This includes testing of usability, performance efficiency, or security of a system and other characteristics found at ISO/IEC 25010. This test can be performed on all levels of test. Coverage for non-functional testing means how many of such characteristics were tested for.
White-box Testing	Tests the internal structure of a system or its implementation. It is mostly tested in component and system testing. Coverage in this test measures the proportion of code components that have been tested as is part of component and
Black-box Testing	Opposed to white-box testing, here we treat software as a *black box* with no knowledge on how software achieves its intended functionality. Merely the output of this form of testing is compared with the expected output or behaviour. The advantage of black-box testing is that no programming knowledge is required and therefore well equipped to detect biases that arise if only programmer write and test software. This test can be applied at all levels of testing.
Maintenance Testing	Tests changes of already delivered software for functional and non-functional quality characteristics.
Static Testing	Form of testing that does not execute code but *manually* examines the system, i.e. through reviews, linters, or formal proofs of the program.
Change-related Testing	Tests whether changes corrected (confirmation testing) or caused errors (regression testing). Change-related testing can be applied on all levels of testing.
Destructive Testing	This test aims to make the software fail by proving unintended inputs which tests the robustness of the software. This can be applied on all levels of software testing.

5.4 Coverage

A reliable measure of testing is test coverage. 100% statements (such as functions) coverage means that each statement of code has been executed during test time. The next step of coverage quality is the decision coverage which means that each program decision was executed such that each possible outcome occurs at least once. This can also be applied for inference of an AI model, e.g., a classifier. This becomes particularly interesting for rare classes. Most desirable is the condition coverage where each condition of a program takes on all possible outcomes at least once or in multi-condition coverage where all combinations of conditions in a program decision are exercised.

5.5 Verification and validation for machine learning

According to the National Institute for Standards and Technology (NIST), benchmarks, metrics, and the correct data is curated to successfully test an AI (https://www.nist.gov/artificial-intelligence). This is described in a concrete example when looking at the testing process for automated face recognition for flight boarding described by NIST (https://www.nist.gov/newsevents/news/2021/07/nist-evaluates-face-recognition-softwares-accuracy-flight-boarding). It becomes clear that the right combination of false/true-positives, and -negatives plays a role, data curation, and non-discriminatory capabilities of the AI, mostly driven by the data curation for the benchmark. In other cases, collecting a representative collection of data that is non-discriminatory and yields a high AI performance may not be possible or affordable. We deem those non-testable and this also frequently includes cryptographic and scientific software – no only machine learning.

However, NIST advertises the concept of metamorphic testing to tackle such a problem for, e.g., testing of autonomous vehicles (https://csrc.nist.gov/csrc/media/Projects/automated-combinatorial-testing-for-software/documents/FromNC2SA_IEEEComp_SI_Auton_Veh_Aug2021.pdf). The difficulty in deep neural networks (DNNs) is that statement coverage does not equate to a full coverage of the network. DeepXplore has a 100% statement coverage with only 34% of the neurons ever being activated. We can automate the generation of transformed images, so called pseudo-oracles, when training two DNNs to classify images. The goal of both DNNs is that they should classify images differently as much as possible as well as increasing the neuron activity. A greedy search algorithm can be used to find transformations that maximizes the neuron activity per image.

Combining the information on the type of input transformation, the deviation of the AI from its expected behaviour, and the amount of neuron activity can help technical and domain knowledge experts estimate the quality of the DNN as well as the test suit. There are also guidelines that help teams set up a metamorphic test suite that provides a list of questions to go through to accomplish verification and validation (https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=931851).

Due to the increasing demand for tested ML systems, the term ML ops (operations) is steadily increasing. It appreciates the aforementioned tight link between data, model, and context. It treats the whole pipeline of necessary steps that leads to the ML's output as one. The pipeline can be tested and monitored to achieve a well functional AI/ML model (https://ml-ops.org/content/mlops-principles, https://ieeexplore.ieee.org/document/8258038).

There are tools to help make testing easier which we are going to introduce in the next section.

6 Identification of gaps needed for the assessment platform

While there are many suggestions made in section 5 on how to test software in various depths and from different perspective, there are still some omissions which would be recommended to test when we want to test a health AI. We produce two sets of inputs where we expect similar outputs. When producing more input is expensive, other techniques can be used such as data transformation methods (increasing brightness of image data).

6.1 Gaps in functional and black-box testing

Although most software has the possibility to be tested for an output given a predefined input, this approach proofs difficult for AI models.

Most state-of-the-art AI algorithms are tied to some set of training data and AI-specific software-hardware-combinations. An AI is thus enclosed by several steps of computation that should be viewed as inseparable from the AI. This process needs to be streamlined to be tested and reproduced (e.g., with tools like MLFlow, Sacred etc.) to build a tight connection between data and AI and containerization (e.g., Docker) to tie the necessary libraries and software to the AI. One figurehead for such a pipeline could be the assessment platform of this Focus Group's assessment platform that will try to find a horizontal solution on how to structure a whole AI-pipeline. In the following, we will explain why testing AIs together with data, a container and hardware-specifications are important and not covered by standard testing recommendations.

Certain algorithms rely on assumptions about the (input) data. To assure its functionality, input data needs to be tested for these assumptions to assure that the output of the AI works as expected including edge cases in the input. This assumption might be fuzzier compared to traditional software testing and thus needs to be viewed slightly differently. However, even for non-AI-based software, the US Food and Drug Administration (FDA) acknowledges in the document *General Principles of Software Validation; Final Guidance for Industry and FDA Staff* that not all types of input can be tested. When specifying tests for the input, we always need to keep in mind that although all tests have passed, our AI might still break in the future due to non-foreseen input data. Continued testing and evaluation for unaccounted input is important.

Especially algorithms that utilize "special" hardware/hardware specification (e.g. graphics card/16-bit floating operations) might only run (or produce the exact expected results) on an exhaustive list of hardware-software combinations. This might be quite prominent in the medical setting. Especially data from the medical setting produces data that has strong device-specific properties which can be picked up by the AI. The AI usually cannot easily generalize over these device-specific difference because the training data most of the time is in-house data and therefore originates from the same data acquisition process. A dataset containing samples with lesser privacy concerns can be easier crowdsourced/combined assuring that different kinds of hardware/software combinations were used to produce such data. Thus, it is desirable to test whether an AI performs equally well on both, different hardware and software (e.g., automatic preprocessing of raw hardware output before file is saved) for data acquisition.

Given that we possess a closed unit of all the necessary components to run an AI, the tests to verify its functionality probably falls into the category of functional black-box testing. We have a clear idea of what the AI is supposed to do, e.g., to detect cancer in an x-ray which is the functional part but we do not necessarily have any insights of the inner workings of the algorithm. When running a benchmarking challenge on some assessment platform, then there might be the problem that the algorithm must not be disclosed since it might be part of the company's intellectual property. Or we have some large machine learning or deep learning model comprised of a large number of parameters that are only accessible through specialized algorithms that do not necessarily reflect how the algorithm works. In short, we either must not see how the algorithm works or it is hardly human-interpretable. Thus, we assume black-box testing.

6.2 Gaps in non-functional testing

While the AI algorithm might run without errors performing the aforementioned testing, we have certain expectations of the algorithm how it should work, e.g., not only to detect cancer in x-rays but also rare forms equally well as more common ones or to work well independent of the used machine.

Input data can change in its quality but does not necessarily break the AI. Thus, certain measures need to be established to detect a concept drift. A learning goal is mostly linked to some metric or other performance measure. It is important to specify metrics and tests to assess whether an algorithm and/or dataset achieves the expected performance of the user/stakeholder. Learning goals needs to be tested as well to avoid typical pitfalls such as data leakage, bias, etc. We refer to the deliverable N° 7.3. "Data and artificial intelligence assessment methods (DAISAM) reference" for a thorough treatise of this topic.

Performance, in some cases, cannot be simply represented by numerical values. Acceptance testing is equally important in case the output is no simple scalar indicating some classification or regression performance but consists of a set of interactions with the user (i.e. chatbot). Human evaluation can lead to ambiguities since different individuals will rate the same AI output differently. It is best to agree on a number of human-tester and how their opinions are blended to achieve comparability between different AIs.

Each AI should contain a reference paper or implementation that shows that the planned AI system tackles the problem adequately. Much time can be wasted not handling known problems or using known solutions.

6.3 Leaderboard probing

From another perspective, when organizing an AI benchmark or challenge, one needs to test how certain modification of aggregation, missing data handling, and metrics can affect the outcome of the benchmark or competition. One example is that certain metrics allow the competitors, if probed correctly, to gain knowledge about the test data. Such vulnerabilities to so-called leader board probing need to be avoided by testing vulnerability to such methods.

Another problem with established metrics used in benchmarking is that they don’t differentiate between hard and easy samples when evaluating the AI. A model could have a high accuracy but make wrong predictions with high confidence. This is problematic in a production environment and makes it hard to debug the model or develop trust in its predictions. A model with less accuracy but a more truthful confidence range may be more applicable for a production environment. A way to weight hard samples higher, e.g., many appearances of the word “not” in text being classified in a sentiment classification task with an expected positive prediction, could be to weight samples by intensity of them being out of distribution (https://arxiv.org/pdf/2106.05532.pdf)

Overfitting is also a way of leaderboard training. Depending on how different the training and testing data set in a benchmark is, it makes sense to optimize the way an AI overfits. The amount of overfitting you are willing to make depends on how different the distributions are for your training and test data. This can be made visible with adversarial validation where you remove labels from training and test data and train a classifier to distinguish them given new labels. If the AUC is 0.5, the model has a hard time to differentiate between both proving that their distribution is similar. Otherwise, we need to accommodate to the test distribution (https://www.google.com/search?client=firefox-b-e&q=the+kaggle+book). Information on the test data can be retrieved through dummy models that help you find out properties of the undisclosed test data.

Bibliography

Anderson-Cook et al. (2019). How to host an effective data competition: Statistical advice for competition design and analysis. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 12(4), 271-289. https://doi.org/10.1002/sam.11404

Reinke et al. (2018). How to exploit weaknesses in biomedical challenge design and organization. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (pp. 388-395). Springer, Cham. https://doi.org/10.1007/978-3-030-00937-3_45

Wiegand, T., Krishnamurthy, R., Kuglitsch, M., Lee, N., Pujari, S., Salathé, M., Wenzel, M. and Xu, S. (2019). WHO and ITU establish benchmarking process for artificial intelligence in health. *The Lancet*, 394(10192), 9-11. https://doi.org/10.1016/S0140-6736(19)30762-7

[DEL0.1] ITU/WHO FG-AI4H deliverable DEL0.1, Common unified terms in artificial intelligence for health.

____