Syllabus - Testing AI-Based Systems

INTRODUCTION

The testing of traditional systems is well-understood, but AI-Based systems, which are becoming more prevalent and critical to our daily lives, introduce new challenges. This syllabus covers the key concepts of Artificial Intelligence (AI), how we decide acceptance criteria and how we test AI-Based systems. These systems are typically complex (e.g. deep neural nets), based on big data, poorly specified and non-deterministic, which creates many new challenges and opportunities for testing them.

This syllabus describes the requirements for a 2-day course to be followed by a 1-hour, closed book, multiple choice exam.

In this document an AI-Based system is a system that includes at least one AI component. Throughout this document the acronym AI is used to represent the term 'Artificial Intelligence'.

BUSINESS OUTCOMES

At the end of this course candidates will be able to:

- Understand the current state of AI and expected advances in the near future;
- Interpret and provide guidance on the specification of acceptance criteria for AI-Based systems;
- Contribute to the development process for machine learning systems and suggest opportunities for influencing their quality;
- Understand the new challenges of testing AI-Based systems, such as their complexity and non-determinism;
- Contribute to the test strategy for an AI-Based system;
- Apply black box and white box test design techniques to generate test suites for AI-Based systems;
- Recognize the need for virtual test environments to support the release of complex AI-Based systems;
- Understand the current state of testing supported by AI.

TARGET AUDIENCE

This course is focused on individuals with an interest in, or a need to perform, the testing of AI-Based systems, especially those working in areas such as autonomous systems, big data, retail, finance, engineering and IT services. This includes people in roles such as system testers, test analysts, test engineers, test consultants, test managers, user acceptance testers, business analysts and systems developers.

Examinable Learning Objectives and Cognitive Levels of Knowledge

Learning objectives support the business outcomes and are used to create the Certified Testing AI-Based Systems exam.

In general, all chapters of this syllabus are examinable at a K1 level, and at a higher level where shown. That is, the candidate may be asked to recognize, remember, or recall a keyword or concept mentioned in any of the nine chapters. The knowledge levels of the specific learning objectives are shown at the beginning of each chapter, and classified as follows:

- K1: remember
- K2: understand
- K3: apply

The definitions of all terms listed as keywords just below chapter headings shall be remembered (K1), even if not explicitly mentioned in the learning objectives.

How this Syllabus is Organized

There are nine chapters with examinable content. Times are specified for each chapter; timing is not provided below chapter level. For accredited training courses, the syllabus requires a minimum of 13.75 hours of instruction, distributed across the nine chapters as follows:

- Chapter 1: Introduction to AI and Testing - 150 minutes
- Chapter 2: AI System Characteristics and Acceptance Criteria – 120 minutes
- Chapter 3: Machine Learning – 180 minutes
- Chapter 4: Machine Learning Performance Metrics and Benchmarks – 90 minutes
- Chapter 5: Introduction to the Testing of AI Systems – 45 minutes
- Chapter 6: Black Box Testing of AI-Based Systems – 120 minutes
- Chapter 7: White Box Testing of Neural Networks – 45 minutes
- Chapter 8: Test Environments for AI-Based Systems – 45 minutes
- Chapter 9: Using AI for Testing - 30 minutes

CONTENTS

Introduction 1
Business Outcomes 1
Target Audience 1
Examinable Learning Objectives and Cognitive Levels of Knowledge 2
How this Syllabus is Organized 2
Glossary 7
1 Introduction to AI and Testing 15
1.1 Definition of 'Artificial Intelligence' and the 'AI Effect' 16
1.2 AI Use Cases 16
1.3 AI Usage and Market 17
1.4 Failures and the Importance of Testing for AI-Based Systems 17
1.5 The Turing Test and the History of AI 18
1.6 Robots and Software Agents 19
1.7 AI Technologies 20
1.8 AI Hardware 20
1.9 AI Development Frameworks 21
1.10 Narrow vs General AI and Technological Singularity 21
1.11 AI and Autonomous Systems 22
1.12 Safety-Related AI-Based Systems 23
1.13 Standardization and AI 23
1.13.1 Regulatory Standards for AI 24
1.14 The AI Quality Metamodel – DIN SPEC 92001:2019 25
2 AI System Characteristics and Acceptance Criteria 26
2.1 AI-Specific Characteristics 26
2.1.1 Adaptability 26
2.1.2 Autonomy 27
2.1.3 Evolution 27
2.1.4 Flexibility 27
2.1.5 Bias 28
2.1.6 Performance Metrics 28
2.1.7 Transparency 28

2.1.8 Complexity 29
2.1.9 Non-Determinism 29
2.2 Aligning AI-Based systems with human values 30
2.3 Side-Effects 30
2.4 Reward Hacking 31
2.5 Specifying Ethical Requirements for AI-Based Systems 31
3 Machine Learning 33
3.1 Introduction to Machine learning 33
3.2 The Machine Learning Workflow 34
3.2.1 Understand the Objectives 35
3.2.2 Select a Framework 35
3.2.3 Build and Compile the Model 35
3.2.4 Source the Data 35
3.2.5 Pre-Process the Data 36
3.2.6 Train the Model 36
3.2.7 Evaluate the Model 36
3.2.8 Tune the Model 36
3.2.9 Test the Model 36
3.2.10 Deploy the Model 37
3.2.11 Use the Model 37
3.2.12 Monitor & Tune the Model 37
3.3 Machine Learning Training and Test Data 37
3.4 Overfitting and Underfitting in Machine Learning 37
3.4.1 Overfitting 37
3.4.2 Underfitting 38
3.5 Bias and Fairness in the Training Data 38
3.6 Data Quality 38
3.7 Machine Learning Algorithm/Model selection 39
3.8 Machine Learning Testing and Quality Assurance 39
3.8.1 Review of ML Workflow 40
3.8.2 Acceptance Criteria 40
3.8.3 Framework, Algorithm/Model and Parameter Selection 40
3.8.4 Model Updates 40

3.8.5 Training Data Quality 40
3.8.6 Test Data Quality 40
3.8.7 Model Integration Testing 40
3.8.8 Adversarial Examples and Testing 40
4 Machine Learning Performance Metrics and Benchmarks 42
4.1 Machine Learning Performance Metrics 42
4.1.1 Confusion Matrix 42
4.1.2 Accuracy 42
4.1.3 Precision 42
4.1.4 Recall 43
4.1.5 F1-Score 43
4.1.6 Selection of Performance Metrics 43
4.2 Benchmarks for Machine Learning 43
5 Introduction to the Testing of AI Systems 45
5.1 Challenges in Testing AI-Based Systems 45
5.1.1 System Specifications 45
5.1.2 Test Input Data 46
5.1.3 Probabilistic & Non-Deterministic Systems 46
5.1.4 Self-Learning Systems 46
5.1.5 Complexity 47
5.1.6 AI-Specific Characteristics (see section 2.1) 47
5.2 The Test Oracle Problem for AI-Based Systems 47
6 Black Box Testing of AI-Based Systems 48
6.1 Combinatorial Testing 48
6.2 Back-to-Back Testing 49
6.3 A/B Testing 49
6.4 Metamorphic Testing 50
7 White Box Testing of Neural Networks 52
7.1 Structure of a Neural Network 52
7.2 Test Coverage Measures for Neural Networks 53
7.2.1 Neuron Coverage 53
7.2.2 Threshold Coverage 53
7.2.3 Sign Change Coverage 53

7.2.4 Value Change Coverage 54
7.2.5 Sign-Sign Coverage 54
7.2.6 Layer Coverage 54
7.2.7 Test Effectiveness of the White Box Measures 54
7.3 White Box Testing Tools for Neural Networks 54
8 Test Environments for AI-Based Systems 55
8.1 Test Environments for AI-Based Systems 55
8.2 Test Scenario Derivation 56
8.3 Regulatory Test Scenarios and Test Environments 57
9 Using AI for Testing 58
9.1 Introduction to AI-Driven Testing 58
9.2 Forms of AI used for Testing 58
9.3 Test Types Supported by AI 59
9.4 Example Use Cases of AI for Testing 59
9.4.1 Bug Prediction 59
9.4.2 Static Analysis 60
9.4.3 Regression Test Optimization 60
9.4.4 Test Input Generation 61
9.4.5 Automated Exploratory Testing 61

GLOSSARY

A/B testing
statistical testing approach that allows testers to determine which of two systems performs better

(aka split-run testing)

accuracy
performance metric used to evaluate a classifier, which measures the proportion of classifications predictions that were correct

activation value
output of an activation function of a node in a neural network

adaptability
ability of a system to react to changes in its environment in order to continue to meet both functional and non-functional requirements

adversarial attack

deliberate use of adversarial examples to cause a neural network to fail

adversarial example
input to a neural network created by applying small perturbations to a working example that results in the neural network outputting an incorrect result with high confidence

adversarial testing
testing approach based on the attempted creation and execution of adversarial examples to identify defects in the neural network

AI effect
situation when a previously labelled AI system is downgraded as technology advances

AI Quality Metamodel
metamodel intended to ensure the quality of AI-Based systems

Note 1 to entry: Defined in DIN SPEC 92001

artificial intelligence (AI)
(1) capability of a system to perform tasks that are generally associated with intelligent beings

(2) branch of computer science devoted to developing data processing systems that perform functions normally associated with human intelligence, such as reasoning, learning, and self-improvement

[ISO/IEC 2382:2015, Information technology -- Vocabulary]

artificial neural network
network of primitive processing elements connected by weighted links with adjustable weights, in which each element produces a value by applying a nonlinear function to its input values, and transmits it to other elements or presents it as an output value

Note 1 to entry: Whereas some neural networks are intended to simulate the functioning of neurons in the nervous system, most neural networks are used in artificial intelligence as realizations of the connectionist model.

Note 2 to entry: Examples of nonlinear functions are a threshold function, a sigmoid function, and a polynomial function.

[ISO/IEC 2382]

automated exploratory testing

form of exploratory testing supported by tools

autonomous system
system capable of working without human intervention for sustained periods

autonomy
ability of a system to work for sustained periods without human intervention

back-to-back testing
approach to testing whereby an alternative version of the system is used as a pseudo-oracle to generate expected results for comparison from the same test inputs

EXAMPLE: The pseudo oracle may be a system that already exists, a system developed by an independent team or a system implemented using a different programming language.

(aka differential testing)

backward propagation
method used in artificial neural networks to determine the weights to be used on the network connections based on the computed error at the output of the network

Note 1 to entry: It is used to train deep neural networks.

bias
measure of the distance between the predicted values provided by the machine learning model and the actual values

(aka unfairness)

classifier
ML model used for classification

clustering
grouping of a set of objects such that objects in the same group (i.e. a cluster) are more similar to each other than to those in other clusters

combinatorial testing
black-box test design technique in which test cases are designed to execute specific combinations of values of several parameters

confusion matrix
table used to describe the performance of a classifier on a set of test data for which the true and false values are known

data pre-processing
part of the ML workflow that transforms raw data into a state ready for use by the ML algorithm to create the ML model

Note 1 to entry: Pre-processing can include reformatting, removal of outliers and duplicates, and ensuring the completeness of the data set.

deep learning
approach to creating rich hierarchical representations through the training of neural networks with one or more hidden layers

deep neural net
neural network with more than two layers

deterministic system
system which, given a particular set of inputs and starting state, will always produce the same set of outputs and final state

drift
changes to ML model behaviour that occur over time

Note 1 to entry: These changes typically make predictions less accurate and may require the model to be re-trained with new data.

(aka degradation or staleness)

evolution
ability of a system to cope with change

Note 1 to entry: Changes may include changes to user requirements and changes made by the system itself.

exploratory testing
experience-based testing in which the tester spontaneously designs and executes tests based on the tester's existing relevant knowledge, prior exploration of the test item (including the results of previous tests), and heuristic "rules of thumb" regarding common software behaviours and types of failure

Note 1 to entry: Exploratory testing hunts for hidden properties (including hidden behaviours) that, while quite possibly benign by themselves, could interfere with other properties of the software under test, and so constitute a risk that the software will fail.

F1-Score
performance metric used to evaluate a classifier, which provides a balance (the harmonic average) between recall and precision

false negative
incorrect reporting of a failure when in reality it is a pass

EXAMPLE: The referee awards an offside when it was a goal.

false positive
incorrect reporting of a pass when in reality it is a failure

EXAMPLE: The referee awards a goal that was offside.

flexibility
ability of a system to work in contexts outside its initial specification (i.e. change its behaviour according to its actual situation to satisfy its objectives)

forward propagation
process of a neural network accepting an input and using the activation functions to pass a succession of values through the network layers to generate a predicted output

fuzz testing
software testing approach in which high volumes of random (or near random) data, called fuzz, are used to generate inputs to the test item

general AI
AI that exhibits intelligent behaviour comparable to a human across the full range of cognitive abilities (aka strong AI)

General Data Protection Regulation (GDPR)
European Union (EU) regulation on data protection and privacy that applies to citizens of the EU and the European Economic Area (EEA)

Note 1 to entry: It also addresses the transfer of personal data outside the EU and EEA areas.

graphical processing unit (GPU)
application-specific integrated circuit designed to manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device

hyperparameter

variables used to define the structure of a neural network and how it is trained

machine learning (ML)
process using computational techniques to enable systems to learn from data or experience

Note 1 to entry: machine learning algorithms that can interact with humans and can optimize their learning behaviour through these interactions. This “human-in-the-loop” can be beneficial in solving computationally hard tasks in a substantially shorter time. This area of machine learning is sometimes also referred to as interactive machine learning (iML).

[ISO/IEC 23053]

metamorphic relation

describes how a change in the test inputs from the source test case to the follow-up test case affects a change (or not) in the expected outputs from the source test case to the follow-up test case

metamorphic testing

testing where the expected results are not based on the specification but are instead extrapolated from previous actual results

ML algorithm
algorithm used to create an ML model from the ML training data

EXAMPLE: ML algorithms include Linear Regression, Logistic Regression, Decision Tree, SVM, Naive Bayes, kNN, K-Means and Random Forest.

ML benchmark suite

collection of benchmarks, where a benchmark is a set of tests used to compare the performance of alternatives

ML classification
machine learning function that results in discrete or categorical output variables

ML model
output of a machine learning algorithm trained with a training data set that generates predictions using patterns in the input data

ML prediction
machine learning function that results in a predicted target variable

EXAMPLE: Includes classification and regression functions.

ML regression
machine learning function that results in numerical or continuous output variables

ML test data
independent dataset used to provide an unbiased evaluation of the final, tuned ML model

ML training data
dataset used to train a ML model

ML validation data
dataset used to evaluate a ML model while tuning it

narrow AI
AI focused on a single well-defined task to address a specific problem

(aka weak AI)

neuron coverage
proportion of activated neurons divided by the total number of neurons in the neural network (normally expressed as a percentage) for a set of tests

Note 1 to entry: A neuron is considered to be activated if its activation value exceeds zero.

non-deterministic system
system which, given a particular set of inputs and starting state, will NOT always produce the same set of outputs and final state

overfitting
generation of a ML model that corresponds too closely to the training data, resulting in a model that finds it difficult to generalize to new data

pairwise testing
black-box test design technique in which test cases are designed to execute all possible discrete combinations of each pair of input parameters

parameterized test scenario

test scenario defined with one or more attributes that can be changed within given constraints

performance metrics
metrics used to evaluate ML models that are used for classification

EXAMPLE: Typical metrics include accuracy, precision, recall and F1-Score.

precision
performance metric used to evaluate a classifier, which measures the proportion of predicted positives that were correct

probabilistic software engineering
software engineering concerned with the solution of fuzzy and probabilistic problems

probabilistic system
system whose behaviour is described in terms of probabilities, such that its outputs cannot be perfectly predicted

reasoning technique
form of AI that generates conclusions from available information using logical techniques, such as deduction and induction

recall
performance metric used to evaluate a classifier, which measures the proportion of actual positives that were predicted correctly

(aka sensitivity)

regulatory standard
standard promulgated by a regulatory agency

reinforcement learning
task of learning a model that makes sequential decisions to maximize an objective, using a process of trial and error

Note 1 to entry: A reinforcement learning task can include the training of a machine learning model in a way similar to supervised learning plus training on unlabelled inputs gathered during the operation phase of the AI system. Each time the model makes a prediction, a reward is calculated, and the model is further refined to optimize the reward.

[ISO/IEC 23053]

reward hacking
activity performed by an agent to maximise its reward function to the detriment of meeting the original objective

robot
programmed actuated mechanism with a degree of autonomy, moving within its environment, to perform intended tasks

Note 1 to entry: A robot includes the control system and interface of the control system.

Note 2 to entry: The classification of robot into industrial robot or service robot is done according to its intended application.

[ISO 18646-1]

safety
expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered

[ISO/IEC/IEEE 12207]

Safety of the Intended Functionality (SOTIF)
ISO/PAS 21448: Safety of the Intended Functionality

search algorithm
algorithm that systematically visits a subset of all possible states (or structures) until the goal state (or structure) is reached

search based software engineering
software engineering that applies search techniques, such as genetic algorithms and simulated annealing to solve problems

self-learning system
adaptive system that changes its behaviour based on learning from the practice of trial and error

sign change coverage
proportion of neurons activated with both positive and negative activation values divided by the total number of neurons in the neural network (normally expressed as a percentage) for a set of tests

Note 1 to entry: An activation value of zero is considered to be a negative activation value.

sign-sign coverage

coverage level achieved if each neuron by changing sign can be shown to individually cause one neuron in the next layer to change sign while all other neurons in the next layer stay the same (i.e. they do not change sign)

simulator
device, computer program or system used during testing, which behaves or operates like a given system when provided with a set of controlled inputs.

software agent
digital entity that perceives its environment and takes actions that maximize its chance of successfully achieving its goals

supervised learning

task of learning a function that maps an input to an output based on labelled example input-output pairs

[ISO/IEC 23053]

technological singularity
point in the future when technological advances are no longer controllable by humans

(aka the singularity)

tensor processing units (TPU)
application-specific integrated circuit designed by Google for neural network machine learning

test oracle problem

challenge of determining whether a test has passed or failed for a given set of test inputs and state

threshold coverage
proportion of neurons exceeding a threshold activation value divided by the total number of neurons in the neural network (normally expressed as a percentage) for a set of tests

Note 1 to entry: A threshold activation value between 0 and 1 must be chosen as the threshold value.

transparency
measure of how easy it is to see how an AI-Based system came up with its result

(aka explainability)

true negative
correct reporting of a failure when it is a failure

EXAMPLE: The referee correctly awards an offside.

true positive
correct reporting of a pass when it is a pass

EXAMPLE: The referee correctly awards a goal.

Turing test
test of a machine's ability to exhibit intelligent behaviour that is indistinguishable from human behaviour

underfitting
generation of a ML model that does not reflect the underlying trend of the training data, resulting in a model that finds it difficult to make accurate predictions

unsupervised learning
task of learning a function that maps unlabelled input data to a latent representation

[ISO/IEC 23053]

value change coverage
proportion of neurons activated where their activation values differ by more than a change amount divided by the total number of neurons in the neural network (normally expressed as a percentage) for a set of tests

virtual test environment
test environment where one or more parts are digitally simulated

1 INTRODUCTION TO AI AND TESTING

Introduction to AI and Testing – 150 minutes
Keywords: Artificial Intelligence (AI), AI Effect, Turing Test, Robot, Software Agent, Search Algorithm, Reasoning Technique, Machine Learning (ML), Deep Learning, ML Model, Graphical Processing Unit (GPU), Tensor Processing Units (TPU), Narrow AI, General AI, Technological Singularity, Autonomous System, Safety, Deterministic System, Non-Deterministic System, Regulatory Standard, GDPR, SOTIF, AI Quality Metamodel.
Definition of 'Artificial Intelligence' and the 'AI Effect' TAI-1.1 Understand how the 'AI Effect' changes people's understanding of Artificial Intelligence (K2) **AI Use Cases** TAI-1.2 Differentiate between AI-Based systems and conventional systems (K2) **Failures and the Importance of Testing for AI-Based Systems** TAI-1.3 Explain the importance of testing for AI-Based systems (K2) **The Turing Test and the History of AI** TAI-1.4 Recall the importance of the Turing Test, AI Winters and improving technology to the progress of AI (K1) **Robots and Software Agents** TAI-1.5 Explain the differences between robots and software agents (K2) **AI Technologies** TAI-1.6 Recognize the different technologies used to implement AI (K1) **AI Hardware** TAI-1.7 Understand the choices available for hardware to implement AI-Based systems (K2) **AI Development Frameworks** TAI-1.8 Identify popular AI development frameworks (K1) **Narrow vs General AI and Technological Singularity** TAI-1.9 Distinguish between narrow AI, general AI and technological singularity (K2) **AI and Autonomous Systems** TAI-1.10 Explain the logical structure of an autonomous system and describe the role of AI in it (K2) **Safety-Related AI-Based Systems**

TAI-1.11 Recall those characteristics that make it difficult to use AI-Based systems for safety-related applications (K1)

Standardization and AI

TAI-1.12 Recognize how AI-Based systems are covered by different levels of standardization (K1)

TAI-1.13 Explain how regulatory standards apply to AI-Based systems (K2)

The AI Quality Metamodel – DIN SPEC 92001:2019

TAI-1.14 Explain how the DIN SPEC 92001 AI Quality Metamodel is applied. (K2)

1.1 Definition of 'Artificial Intelligence' and the 'AI Effect'

To define 'Artificial Intelligence', 'Intelligence' first needs to be defined. The Oxford Dictionaries provide a suitable definition:

> *the ability to acquire and apply knowledge and skills*

Artificial intelligence (AI) is intelligence that does not occur naturally, i.e. as exhibited by humans and animals. The following definition captures this concept:

> *the capability of a system to perform tasks that are generally associated with intelligent beings*

A more specific definition is provided in DIN SPEC 92001, 2019, but it fails to contrast artificial intelligence with natural intelligence:

> *the capability of a system to solve problems by emulating concepts that are generally associated with intelligent behaviour*

Artificial intelligence can also be considered as a discipline, leading to a second definition:

> *a branch of computer science devoted to developing data processing systems that perform functions normally associated with human intelligence, such as reasoning, learning, and self-improvement (*ISO/IEC 2382:2015, Information technology -- Vocabulary*)*

In practice, people's understanding of what is meant by AI changes over time – this is often known as the 'AI Effect'. A strict interpretation of the above definitions may allow what we would now consider basic (non-AI) computer systems to be labelled as AI. For instance, in the 1980s an expert system based on fixed rules that performed activities traditionally carried out by bank clerks was considered to be AI, but today such systems are often considered too simple to be AI. Similarly, the Deep Blue system that beat Garry Kasparov at chess in 1997 is now derided by some as a brute force approach – and so not true AI. It is likely that today's state-of-the-art AI will also be considered 'too simple to be AI' in 20 years' time.

1.2 AI Use Cases

AI can be used for a wide variety of application areas, such as [ISO SC42]:

- Intelligent speech systems (e.g. speech recognition and speech synthesis)

- Computer vision systems (e.g. image classification)
- Natural language processing (NLP) (e.g. deriving meaning from human language)
- Anomaly detection systems (e.g. fraud detection, health monitoring, and security)
- Autonomous systems (e.g. vehicles and trading systems)
- Recommender systems (e.g. for purchases, films and music)

A comprehensive list of AI use cases can be found at https://appliedai.com/use-cases/.

1.3 AI USAGE AND MARKET

AI-Based systems are becoming ever more widespread:

- The perception is that AI is the most significant technology of this time as 69% of technology executives ranked it in the top three most significant technologies over the next 5-10 years. [Edelman 2019 Survey]
- 91% of technology executives believe AI will be at the centre of the next technological revolution. [Edelman 2019 Survey]
- The share of jobs requiring AI skills has grown 4.5x since 2013. [Stanford University's inaugural AI Index]
- Global revenues from AI for enterprise applications is projected to grow from $1.62B in 2018 to $31.2B in 2025. [Statista]
- It is estimated that AI will add $13 trillion to the global economy over the next decade. [Harvard Business Review July 2019]
- 22% of IT budgets are allocated to AI projects. [World Quality Report 2018/9]
- 64% of companies had AI projects in place or planned for next 12 months. [World Quality Report 2018/9]

1.4 FAILURES AND THE IMPORTANCE OF TESTING FOR AI-BASED SYSTEMS

There have already been a number of widely publicized failures of AI. According to a 2019 IDC Survey, "Most organizations reported some failures among their AI projects with a quarter of them reporting up to 50% failure rate; lack of skilled staff and unrealistic expectations were identified as the top reasons for failure." [https://www.idc.com/getdoc.jsp?containerId=prUS45344519]

Example AI failures include:

- IBM's "Watson for Oncology" cancelled after $62 million spent due to "unsafe treatment" recommendations [2018]
- Microsoft's AI Chatbot, Tay, was corrupted by Twitter trolls [2016]
- Joshua Brown died in a Tesla Model S on a bright day, when his car failed to spot a white 18-wheel truck/trailer [2016]

- Elaine Herzberg was killed crossing the street at 10pm with her bicycle in Arizona by an Uber self-driving car travelling at 38 mph [2018]
- Google searches showing high-paying jobs only to male users [2015]
- COMPAS AI-Based sentencing system in the US biased against African Americans [2016]
- Anti-Jaywalking system in Ningbo, China recognized a photo of a billionaire on a bus as a jaywalker [2018]

Failures have historically provided one of the most convincing drivers for performing adequate software testing. Industry surveys show a perception that AI is an important trend for software testing:

- AI was rated the number one new technology that will be important to the testing world in the next 3 to 5 years. [State of Testing Report 2019]
- AI was rated second (by 49.9% of respondents) of all technologies that will be important to the software testing industry in the following 5 years [ISTQB 2017/8 Survey]
- The most popular trends in software testing were AI, CI/CD, and Security (equal first). [LogiGear 2018 Trends Survey]

Testing is already being performed on AI-Based systems:

- 19% of respondent are already testing AI / Machine Learning [State of Testing Report 2019]
- 57% of companies are experimenting with new testing approaches [World Quality Report 2018/9]

1.5 The Turing Test and the History of AI

AI is often considered to have started in the 1950s. 1950 saw Alan Turing publish his paper on Machine Intelligence including what is now known as the 'Turing Test'. In 1951, in the UK, the Ferranti Mark 1 computer was programmed to successfully play and learn the game of draughts (checkers). In 1956, John McCarthy, a US computer scientist, organised the Dartmouth Conference, at which the term 'Artificial Intelligence' was first adopted (he later developed the LISP programming language, which has since been widely used in AI applications). Later in this same decade the General Problem Solver algorithm was first developed by Newell and Simon to solve a range of mathematical problems.

After the Dartmouth Conference, AI became a thriving research field, with DARPA funding research in the US, while in Japan work on an 'intelligent' humanoid robot led to the WABOT-1 in 1972. Meanwhile, work on the ELIZA system in the mid-1960s led to the world's first chatbot.

The first 'AI Winter' ran from about 1974 to 1980. Initial optimism gave way to a loss of faith that researchers would produce anything practical and funding dried up. This was especially seen in areas such as machine vision, which, although theoretically possible, required more processing power and storage than was then available to work in practice.

In the 1980s a new form of AI, known as 'Expert Systems' became popular and knowledge-based systems were the main focus of AI research. MYCIN, an expert system for identifying bacteria causing infections, which was first developed in the 1970s, is an early example of a successful expert system. In Japan in the early 1980s, the government initiated the 'Fifth Generation Computer Project', while by

1985 corporations in the US were spending over a billion dollars per year on in-house AI development, based on successful expert system projects and, meanwhile, the Alvey Programme to support IT was initiated in the UK.

The second AI Winter ran from about 1987 to 1993. This was partly caused by the hype surrounding expert systems, which were found to be difficult to maintain and more limited than expected. The optimistic goals of projects initiated in the early 1980s, such as the 'Fifth Generation Computer Project', were largely not achieved. At about the same time, personal computers became available and undermined the market for specialist AI hardware. As with the first AI Winter, funding into AI research and development was cut.

In the late 1990s, many of the problems associated with a lack of processing power and storage began to be overcome. This led to some successful AI projects, and specialist AI systems saw some notable public successes, such as IBM's Deep Blue chess playing system that beat the current world chess champion, Garry Kasparov in 1997. In the 2000s, machine learning (ML) started to see growing success in specialist fields.

Since the mid-2000s companies such as Amazon, Google and Baidu have started using ML to profit from systems that successfully employ computer vision, natural language processing and understanding consumer behaviour.

1.6 Robots and Software Agents

Robots have a long history, with purely mechanical figures having been documented as far back as 1066 in China. Autonomous robots with electronic systems were first developed at a similar time to Alan Turing's work on machine intelligence, and robots are now widely used in factories in countries such as Korea (the country with the world's highest number of robots per 10,000 employees at 631 in 2016), although the use of AI in such robots is limited.

A software agent is a software system that acts upon information available to it to achieve a goal. For AI, we are more often interested in intelligent software agents that are software agents capable of making decisions based on their experiences (so making them 'intelligent'). Intelligent software agents are also often labelled as autonomous as they are allowed to select which action to perform (see section 1.11 for more on autonomous systems).

Intelligent software agents may work alone or with other agents to implement AI. These agents are most often located in computer systems (either physical or in the cloud) and interact with the outside world through computer interfaces. A tool using AI for performing software testing is most likely to reside in a computer system and interact with the software tester through the user interface and interact with the software it is testing through a computer interface using a defined protocol (such a tool would be considered an AI-Based system as it has an AI component working with conventional, non-AI subsystems, such as the user interface). Intelligent software agents may also reside in robots; the major difference being that the robots provide the AI with a physical presence and a different way of interacting with the environment that is not available to purely computer-based software agents.

1.7 AI TECHNOLOGIES

AI can be implemented using a wide range of approaches or technologies. These can be grouped in different ways including [ISO SC42]:

- Search algorithms
- Reasoning techniques
  - Logic programs
  - Rule engines
  - Deductive classifier
  - Case-based reasoning
  - Procedural reasoning
- Machine learning techniques (see section 3 for more detail)
  - Artificial neural networks
    - Feed forward neural networks
    - Deep learning
    - Recurrent neural networks
    - Convolutional neural networks
  - Bayesian network
  - Decision tree
  - Reinforcement learning
  - Transfer learning
  - Genetic algorithms
  - Support vector machine

Some of the most effective AI-Based systems can be considered as AI hybrids, using a mix of these technologies.

1.8 AI HARDWARE

AI-Based systems, especially ML systems implemented as neural networks performing pattern recognition (e.g. machine vision, speech recognition), require many calculations to be run in parallel. General-purpose CPUs do not perform this type of calculation efficiently and, instead, graphical processing units (GPUs), which are optimised for parallel processing of images using thousands of cores are often used. GPUs are however, not optimised for AI, and a new generation of hardware developed specifically for AI is now becoming available.

Many AI implementations are, by their nature, not focused on exact calculations, but rather on probabilistic determinations and so the accuracy of a 64-bit processor is often unnecessary and processors with less bits can run faster and consume less energy. Because much of the processing time and energy is involved with moving large amounts of data from RAM to the processor for relatively simple calculations, the concept of phase changing memory devices that allow simple calculations to be performed directly on memory are also being developed.

AI-specific hardware architectures include neural network processing units, field programmable gate arrays, application-specific integrated circuits, neuromorphic computing, in addition to the next

generations GPUs. Some of the chips within these architectures are focused on specific areas of AI, such as image recognition. When performing machine learning (see section 3), the processing used to train models can be quite different from the processing used to run the inferencing on the deployed model and this suggests that different processors for each activity should be considered.

Example AI hardware includes:

- NVIDIA – provide a range of GPUs and AI-specific processors, such as the VOLTA.
- Google – has developed application-specific integrated circuits for both training and inferencing. Google TPUs (Cloud Tensor Processing Units), can be accessed by users on the Google Cloud and Edge TPU is a purpose-built ASIC designed to run AI on individual devices.
- Intel – provides Nervana neural network processors for deep learning (both training and inference) and Movidius Myriad vision processing units for inferencing in computer vision and neural network applications.
- Apple – includes its Bionic chip for on-device AI on iPhones.
- Huawei – its Kirin 970 chip for smartphones has built-in neural network processing for AI.

1.9 AI Development Frameworks

There are several open-source AI development frameworks available, often optimised for specific application areas. The most popular include:

- TensorFlow – based on data flow graphs for scalable machine learning by Google
- PyTorch - neural networks for deep learning in the Python language
- MxNet – a deep learning open-source framework used by Amazon for AWS
- Caffe/Caffe2 - open frameworks for deep learning, written in C++ with a Python interface
- CNTK – the Microsoft Cognitive Toolkit (CNTK), an open source deep-learning toolkit
- Keras - a high-level API, written in the Python language, capable of running on top of TensorFlow or CNTK

1.10 Narrow vs General AI and Technological Singularity

Up until now, all successful AI has been 'narrow' AI, which means it can handle a single specialized task, such as playing Go, performing as a spam filter, or driving an autonomous car.

General AI is far more advanced than narrow AI and refers to an AI-Based system that can handle a number of quite disparate tasks, much the same as a human. General AI is also known as High-Level Machine Intelligence (HLMI). A survey of AI researchers published in 2017 reported that the overall mean estimate for when HLMI would be achieved was by 2061. Of these AI researchers, 15% believed that HLMI will result in a bad or very bad outcome for humans.

A popular hypothesis is that once general AI has been achieved (and the AI-Based system is allowed access to the internet), the AI-Based system will use its access to the available information, processing power and storage to enter a cycle of self-improvement. After a small time, this would mean that the

system would have become more intelligent than humans (continuing to become super-intelligent). The point at which this intelligence explosion occurs is known as Technological Singularity.

1.11 AI AND AUTONOMOUS SYSTEMS

An autonomous system can be defined as a:

system that works for sustained periods independent of human control

Autonomous systems can be physical or purely digital, and include systems for:

- Transportation
    - Cars / Trucks
    - Unmanned Aircraft (Drones)
    - Ships / Boats
    - Trains
- Robotic/IoT Platforms (e.g. manufacturing, vacuum cleaners, smart thermostats)
- Medical Diagnostics
- Smart Buildings / Smart Cities / Smart Energy / Smart Utilities
- Financial Systems (e.g. automated market trading systems)

The logical structure of an autonomous system can be considered as comprising three high-level functions: sensing, decision-making and control. Sensors (e.g. cameras, GPS, RADAR, LIDAR) provide inputs to the sensing function and are used to gather information about the system's environment, such as the positions of nearby cars, pedestrians and information on road signs for an autonomous car. Part of this 'sensing' function is also known as localization, which is determining the system's current position in the environment and relating this to maps (e.g. detailed offline maps for autonomous cars). The 'decision-making' function decides what the system's next move should be (e.g. braking, turning, climbing, descending) depending on the function provided by the autonomous system (e.g. adaptive cruise control). The 'control' function implements the decision by calling on actuators (e.g. to release air, open fuel valve).

Fully autonomous systems require more, and perhaps better, sensors than their automated counterparts, and to make sense of the data from these sensors, these systems typically use deep learning, a form of machine learning. To perform the necessary decision-making, the system will also often use deep learning. Thus, each of the high-level functions in the autonomous system can be implemented as AI or can be implemented using other technologies (in an autonomous car, the sensing and decision-making functions are often implemented as AI, while the control function may be implemented using conventional techniques). It is also possible to implement a compete autonomous system as a single ML system (e.g. a car steering system that learns from 'observing' manual steering based on video inputs and steering outputs).

1.12 Safety-Related AI-Based Systems

AI-Based systems are already beginning to be used for making decisions that affect safety and this trend will see increased use of AI for safety-related systems. Safety is defined as the 'expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered (ISO/IEC/IEEE 12207:2017).

Many AI-Based systems are probabilistic and non-deterministic – this unpredictability makes it very difficult to make an evidence-based case that they will not cause harm. However, their ability to provide a probability of correctness with a decision could be used as part of a risk-based approach that combines both AI and conventional non-AI approaches within a safety-related system. Standards for safety-related AI-Based systems are covered in section 1.13.1.2.

1.13 Standardization and AI

Standardization aims to promote innovation, help improve system quality, and ensure user safety, while creating a fair and open industry ecosystem. AI standardization occurs at various levels, including:

- International Standards Organizations
- Regional Standards Organizations
- National Standards
- Other Standards Organizations

Under joint technical committee 1 (JTC1) of ISO and IEC, subcommittee 42 (ISO SC42) is specifically responsible for Artificial Intelligence, although AI-Based systems are considered by several other ISO/IEC committees and groups, such as SC7 (Software and Systems Engineering), TC22 (Road Vehicles) and SG20 (IoT, Smart Cities and Communities).

At the European level, ETSI and CEN-CENELEC are both involved with AI standards. ETSI has an Industry Specification Group (ISG) on Experiential Networked Intelligence (ENI), whose goal is to develop standards for a cognitive network management system incorporating a closed-loop control approach. CEN-CENELEC intends to define a standards roadmap for the AI domain that is due in 2020.

China has several AI standards initiatives at the national level, with national technical committees working on automation systems and integration (SAC/TC 159), audio, video, multimedia and equipment (SAC/TC 242) and intelligent transport systems (SAC/TC268). SAC/TC 28 also addresses AI standardization work related to vocabulary, user interfaces and biometric feature recognition.

The IEEE provides a specific focus on the ethical aspects of AI-Based systems. The IEEE Global Initiative for Ethical Considerations in Artificial Intelligence and Autonomous Systems has a mission "to ensure every stakeholder involved in the design and development of autonomous and intelligent systems is educated, trained, and empowered to prioritize ethical considerations so that these technologies are advanced for the benefit of humanity."

Other standards initiatives include standards on AI tool interoperability, such as ONNX (Open Neural Network Exchange format), NNEF (Neural Network Exchange Format) and PMML (Predictive Model Mark-up Language).

1.13.1 Regulatory Standards for AI

Regulatory standards can be split into two broad categories: those that apply to safety-related systems and those that apply to non-safety-related systems, such as financial, utilities and reporting systems. Safety-related systems are those that could potentially cause harm to people, property or the environment.

1.13.1.1 Non-Safety-Related Regulatory Standards

At present (in 2019), there are few international standards that apply to non-safety-related AI-Based systems. However, from May 2018, the EU-wide General Data Protection Regulation (GDPR) came into effect and can cover AI-Based systems. Any system that uses automated processes to make decisions with legal or similarly significant effects on individuals must follow the GDPR rules that state organizations using such systems must provide users with:

- specific and easily accessible information about the automated decision-making process
- a simple way to obtain human intervention to review, and potentially change the decision

1.13.1.2 Safety-Related Standards

AI-specific requirements for safety-related AI-Based systems are currently (in 2019) poorly covered by standards and in most domains are reliant on pre-existing standards written for conventional (non-AI) systems. Some of these standards (e.g. IEC 61508 and ISO 26262) actually specify that AI-Based systems that are non-deterministic (which is many of them) should not be used for higher-integrity systems, although in practice this often means that AI-Based systems are considered as special cases and follow 'tailored' versions of these standards, ignoring some of the requirements. These existing safety-related standards also require that the tools used to develop safety-related systems are suitably qualified. The currently available AI frameworks and algorithms are not qualified for use on the development of safety-related systems. Although it is possible to gain this qualification through use, the relative immaturity and rapidly evolving nature of ML algorithms would mean that it is unlikely they would satisfy current regulatory requirements in this area.

In the area of autonomous systems, which are already being used (e.g. on roads, in the air, at sea and in factories), there is a danger of a gap between practice (driven by commercial necessity) and the requirements of standards. For road vehicles a new standard, ISO/PAS 21448: Safety of the Intended Functionality (SOTIF), was published in 2019. This partly bridges this gap by covering an area not covered by the existing standards that are concerned with mitigating risks due to failures. For AI-Based systems, an additional problem is that they may cause harm without there being a failure – perhaps due to them simply misunderstanding the situation. SOTIF covers design, verification (e.g. requiring high coverage of scenarios) and validation (e.g. requiring use of simulations).

The U.S. Department of Transportation and the National Highway Traffic Safety Administration (NHTSA) provides guidance for the development and testing of automated driving systems in the US (Automated Driving Systems (ADS): A Vision for Safety 2.0), however use of this guidance is purely voluntary.

A new standard is also being developed by UL for the safety of autonomous products in general (Standard for Safety for the Evaluation of Autonomous Products, UL 4600). This standard provides assessment criteria to determine the acceptability of a safety case for the autonomous product.

1.14 The AI Quality Metamodel – DIN SPEC 92001:2019

DIN SPEC 92001-1 is a freely available standard that provides an AI Quality Metamodel intended to ensure the quality of AI-Based systems. The standard defines a generic life cycle for an AI module, and assumes the use of ISO 12207 life cycle processes. Each AI module is assigned a level of risk (high or low), based on whether the AI module has relevant safety, security, privacy, or ethical attributes.

DIN SPEC 92001-2 describes quality requirements which are linked to the three quality pillars of functionality & performance, robustness, and comprehensibility. They also link to one or more life cycle stages and processes and they are assigned a category of model, data, platform or environment. Based on their relevance, these requirements of the AI module are classified as mandatory, highly recommended or recommended. This requirement classification and the assigned risk of the AI module are used to determine the extent to which the recommended quality requirements should be followed.

2 AI SYSTEM CHARACTERISTICS AND ACCEPTANCE CRITERIA

AI System Characteristics and Acceptance Criteria – 120 minutes
Keywords: Adaptability, Autonomy, Evolution, Degradation, Drift, Staleness, Flexibility, Unfairness, Bias, Performance Metrics, Transparency, Explainability, Reward Hacking, A/B testing, Back-to-Back Testing, Metamorphic Testing.
AI-Specific Characteristics TAI-2.1 Give examples of system characteristics that are specific to AI-Based systems (K2) **Aligning AI-Based systems with human values** TAI-2.2 Describe the challenge of aligning AI-Based systems with human values (K2) **Side-Effects & Reward Hacking** TAI-2.3 Explain the occurrence of side-effects and reward hacking in AI-Based systems (K2) **Specifying Ethical Requirements for AI-Based Systems** TAI-2.4 Understand the ethical principles that should be respected in the development, deployment and use of AI systems (K2) TAI-2.5 Select appropriate objectives and acceptance criteria for a given AI-Based system (K3)

2.1 AI-SPECIFIC CHARACTERISTICS

AI-Based systems have both functional and non-functional requirements, the same as any system. As such, the quality characteristics in the ISO 25010 Quality Model can be used to define, in part, the requirements of AI-Based systems. However, AI-Based systems have some unique characteristics that are not contained with this Quality Model, such as Adaptability, Autonomy, Evolution, Flexibility, Bias, Performance, Transparency, Complexity and Non-Determinism. These non-functional characteristics are described in more detail below, along with suggestions on testing for them. The full set of quality characteristics for AI-Based systems could be used as the basis for a checklist used during test planning for the identification of risks that need to be mitigated by testing. Note that there is potentially some interaction, overlap and possible conflicts between these characteristics, as there is with any set of non-functional requirements.

2.1.1 Adaptability

Adaptability is the ability of the system to react to changes in the environment in order to continue to meet both functional and non-functional requirements. The attributes of an adaptable system include self-configuration, self-healing, self-protection and self-learning. Adaptability requires a system to actively or passively gather information about its operational environment. Exploration (active gathering of information) provides useful information for self-improvement, but it can also be dangerous (e.g. pushing the boundaries of a flight envelope) and systems should exhibit caution when exploring in safety-related situations. Adaptability requirements should specify environment changes to

which the system should be able to adapt and also include requirements on the adaptation process itself, such as maximum time to adapt, where appropriate.

The testing of adaptability is typically based on environment modification or mutation. Both functional and non-functional requirements should be tested, and a form of regression testing, ideally automated, would be a suitable approach. The adaptation process performed by the system should also be tested, to determine, for instance, whether the system adapts within a required timeframe and whether the system stays within constraints for the resources consumed to achieve the adaptation.

2.1.2 Autonomy

Autonomy is the ability of the system to work for sustained periods without human intervention. The expected level of human intervention should be specified for the system – and so should be part of the system's functional requirements (e.g. 'the system will maintain cruise condition until one of the following occurs…'). Autonomy can also be considered in combination with adaptability or flexibility (e.g. system should be able to maintain a given level of adaptability or flexibility without human intervention). In some circumstances, an AI-Based system may exhibit too much autonomy, in which case it may be necessary for a human to take control away from it.

One approach to testing for autonomy is to try and force the system out of its autonomous behaviour and request intervention in unspecified circumstances (a form of negative testing). Negative testing can also be used to try and 'fool' the system into thinking it is in control when it should request intervention (e.g. by creating test scenarios at the boundary of its operational envelope – suggesting the application of boundary value concepts to scenario testing).

2.1.3 Evolution

Evolution is concerned with the ability of the system to cope with two types of change. The first type of change is when the user requirements change - this could be for many reasons and may even be caused by users' interaction with the system itself. The second type of change is when the system changes its behaviour, which could be due to the system learning new behaviours as it is used (see self-learning in adaptability, above). Changes in system behaviour are not always positive, and the negative form of this system characteristic can be known as degradation, drift or staleness.

Testing for system evolution normally takes the form of maintenance testing, which needs to be run on a frequent basis. This testing typically needs to monitor specified system goals, such as performance goals (e.g. accuracy, precision and sensitivity), and ensure that no data bias has been introduced to the system (e.g. Microsoft Tay chatbot). The result of this testing may be that the system is re-trained, perhaps with an updated training data set.

2.1.4 Flexibility

Flexibility is the ability of a system to work in contexts outside its initial specification (i.e. change its behaviour according to its actual situation to satisfy its objectives). Flexibility should be explicitly specified in the requirements. This can be achieved informally through the use of verbs with different levels of strictness, such as 'must', 'may' and 'close to', or it can be achieved more formally with the use of probabilities and possibilities in specifications (e.g. RELAX requirements language). Flexibility can be achieved using different technical mechanisms, such as reactivity, pro-activity, interaction, adaptation or self-learning.

Testing for flexibility requires tests that extend the system's original behaviour. Metamorphic testing (see section 6.4), which includes the use of metamorphic relations to extend the initial specification (within specified limits) can be used to test for flexibility.

2.1.5 Bias

Bias (also known as unfairness) is a measure of the distance between the predicted values provided by the machine learning model and the actual values (see section 3 for more detail on machine learning). In machine learning (ML) the idea is to identify and generalize patterns in the training data and implement these patterns in the model to allow it to classify and predict. If the training data is not representative of the data expected to be used operationally then the model is likely to demonstrate bias. Training data can be compromised by both explicit and implicit bias. Implicit bias is created unintentionally, for instance when the machine learning picks up on unexpected patterns in the training data. Explicit bias is where training data is selected with known patterns that can be expected to influence the derived model.

Testing for bias can be performed at two stages. First, it can be removed from the training data through reviews, but this requires expert reviewers who can identify possible features that create bias. Second, a system can be tested for bias by the use of independent testing using bias-free testing sets. When we know that training data is biased, it may be possible to remove the source of the bias (e.g. we could remove all information that provided clues as to the sex or race of the subjects). Alternatively, we could accept that the system includes bias (either implicit or explicit) but provide transparency by publishing the training data. See section 3.5 for more on testing for bias.

2.1.6 Performance Metrics

Performance metrics are defined for machine learning (ML) models, the most popular of which are accuracy, precision and recall (see section 4 for more details). These metrics should be agreed and defined as part of the system requirements.

Testing of models for performance is often provided as part of a ML Framework (e.g. TensorFlow), which will calculate these measures for a given test data set.

2.1.7 Transparency

Transparency (also known as explainability) is a measure of how easy it is to see how an AI-Based system came up with its result. For instance, an image classifier that determines an object is a cat could demonstrate transparency by pointing out those features in the input image that made it decide the object was a cat. The complexity of some AI-Based systems (e.g. deep neural nets) means that we cannot see how they work; such systems lack transparency. To put this complexity in context, a typical neural network with satisfactory performance may have around 100 million parameters that were learned during training that contribute to a single decision (there are no visible 'if X and Y then result is Z' rules as found in traditional expert systems). Other factors that can cause a lack of transparency are when the source or choice of training data are obscure.

The required level of transparency changes from system to system. For instance, the results used to direct a marketing campaign are likely to need less transparency than the results for more critical systems, such as those used to decide on surgery or set jail terms (e.g. in regulated domains). For such critical systems we need transparency at least until we learn to trust the system. The General Data

Protection Regulation (GDPR) includes requirements for explainability for certain decision-making systems.

Different AI Frameworks provide different levels of transparency and should be selected, in part, based on the required level of transparency. As with many non-functional requirements, there are possible conflicts with other non-functional characteristics – in this case achieving transparency may need to be traded off against required accuracy. One way to address potential transparency problems is by publishing details of the choice of framework, training algorithm and training data used to create the (opaque) deployed model. The field of explainable AI (XAI) covers ways to make AI-Based systems more explainable.

Testing for transparency is a qualitative activity and ideally requires the target audience (or a representative set of testers) to perform the testing to determine if the workings of the AI-Based system are understandable or the provided explanation is satisfactory.

2.1.8 Complexity

AI-Based systems, and especially those implemented through deep learning, can be extremely complex. AI-Based systems are often used for problems where there is no alternative, due to the complex nature of the problem (e.g. making decisions based on big data). As was mentioned before, it is not unusual for a deep neural network to have upwards of 100 million parameters.

The complexity of such systems creates a test oracle problem; it may require several experts some time and discussion to agree on a single test case result from a complex AI-Based system and, ideally, we would want to run many tests, which becomes infeasible if we have to rely on experts to (slowly) generate expected results. A number of test techniques can be used to address the test oracle problem, including A/B testing, back-to-back testing and metamorphic testing (see section 6 for more detail on these techniques).

2.1.9 Non-Determinism

A non-deterministic system is not guaranteed to produce the same outputs from the same inputs every time it runs (in contrast to a deterministic system). With a non-deterministic system there may be multiple (valid) outcomes from a test with the same set of preconditions and test inputs. Determinism is normally assumed by testers - it allows tests to be re-run and the same results to be achieved – this is extremely useful when re-using tests for regression or confirmation testing. However, many AI-Based systems are based on probabilistic implementations, meaning that they do not always produce the same results from the same test inputs. For instance, the calculation of the shortest route across a non-trivial network (the travelling salesman problem) is known to be too complex to calculate exactly (even by a powerful computer) and sub-optimal solutions are normally considered acceptable. AI-Based systems can also include other causes of non-determinism, such as concurrent processing (although these are often found in complex conventional, non-AI, systems).

The testing of non-deterministic systems requires the tester to address the problem that multiple actual results may be considered correct for the same test case. For a deterministic system, the checking for correctness is a simple check of ‘does actual result = expected result’, while for a non-deterministic system the tester must have a deeper knowledge of the required behaviour so that they can come up with a reasonable (less black and white) check for whether the test has passed (e.g. ‘is shortest route within 2% of the optimal solution?’).

2.2 ALIGNING AI-BASED SYSTEMS WITH HUMAN VALUES

Russell [Myths of AI] points out two major problems with AI-Based systems. First, the specified functionality may not be perfectly aligned with the values of the human race, which are (at best) very difficult to pin down. He gives the example of King Midas, where the requested ability to turn everything he touched into gold was imparted – exactly as requested – but then found to be not what he truly wanted. When we specify the required objectives of AI-Based systems we need to be sure that what is requested is actually what is needed – or first ensure the system is intelligent enough to provide what we request, while also taking into account human norms.

One way for AI-Based systems to learn these human norms would be through observation, however great care is needed to ensure that the observed human behaviour is representative and only representative of 'good' human behaviour (probably defined as excluding both deliberately bad behaviour and irrational behaviour, even of if this irrational behaviour is by 'good' humans). Consideration also needs to be given to this learning of human norms being a continuing process, as what we consider acceptable behaviour today is quite different from what was considered acceptable behaviour 20 years ago – human norms can change quite quickly.

Russell's second problem is that any sufficiently capable intelligent system will prefer to ensure its own continued existence and to acquire physical and computational resources – not for their own sake, but to succeed in its assigned task. It is recognized that a sufficiently intelligent system will disable any 'off' switch early on in its operation; simply because when it is disabled it is unable to achieve its given objectives. AI-Based systems will try to fulfil their given objectives, but we need to be careful of unwanted behaviours, such as those that result in side-effects (see section 2.3) or reward hacking (see section 2.4).

2.3 SIDE-EFFECTS

Side-effects occur when an AI-Based system attempts to achieve its objectives and causes (typically negative) impacts on its environment. For instance, a home cleaning robot may be tasked with cleaning the kitchen in your home and decide that 'eliminating' your new puppy will help it achieve its objective. Of course, you could explicitly require your robot to accept that the puppy has a right to be in the kitchen (and therefore not be eliminated), but as AI-Based systems are used in ever more complex environments it soon becomes impracticable to explicitly specify how the robot should interact with every aspect of its operational environment. For instance, your cleaning robot would also have to be told that using a high-pressure hose to clean the kitchen was not practical due to the (side-) effect of the water on the electrical appliances and sockets.

At a high level, specified objectives for AI-Based systems may need to include a caveat that minimises side-effects. For narrow AI, such side-effects may be explicitly specified, but as AI-Based systems become more advanced and start working in more varied operational environments it may be more efficient to define more generic caveats, such as requiring a minimal change to the environment while achieving their objective.

2.4 Reward Hacking

AI-Based systems using reinforcement learning (see section 3.1) are based on a reward function that gives the system a higher score when the system better achieves its objectives. For instance, a home cleaning robot may have a reward function based on the amount of dirt it removes from the floor – getting higher scores when the amount of dirt removed is higher. Reward hacking occurs when the AI-Based system satisfies the reward function and so gets a high score, but mis-interprets the required objective. In the example of the cleaning robot, one way for it to achieve a very high score would be for it to initially make the floor extremely dirty, so giving it the opportunity to remove more dirt – a set of activities that do not meet the spirit of the initial objective of cleaning the kitchen. In this example the floor should eventually be clean (although unnecessary energy will have been expended), but there are many examples of reward hacking where the AI-Based system satisfies the reward function but does not come close to achieving the required objective (e.g. a cleaning robot with a reward function based on it being able to see less visible dirt that disables its vision system).

Limiting the system's ability to innovate, however, is not the solution. One of the features of AI-Based systems is that they should be able to come up with smart ways to solve problems, often in ways humans would not have considered (or perhaps even understand).

2.5 Specifying Ethical Requirements for AI-Based Systems

Ethics is defined in the Cambridge Dictionary as 'a system of accepted beliefs that control behaviour, especially such a system based on morals'. As AI-Based systems have become more popular, the topic of ethics and how AI-Based systems should implement them is probably the most discussed topic in AI, drawing in far more people than those involved in the technical aspects of AI.

An example of the interest in ethics in AI can be seen in MIT's Moral machine [moralmachine.mit.edu]. This is a platform for gathering people's opinions on moral decisions that may be made by autonomous cars, with the aim of providing guidance to the developers of such vehicles. Between 2014 and 2018 this platform gathered 40 million ethical decisions in ten languages from millions of people in 233 countries and territories. The (ongoing) study has found that there is a broad consensus that systems should give priority to younger people, priority to people over animals and priority to saving more people (e.g. save four occupants of a car over two pedestrians). The study also found that there are significant differences in the choices made by people from different parts of the world (suggesting that autonomous cars may need to follow different ethical guidelines depending on where they are to be used).

The European Commission High-Level Expert Group on Artificial Intelligence published key guidance to promote trustworthy AI in the area of ethics in April 2019. It identifies the ethical principles that should be respected in the development, deployment and use of AI systems:

- Develop, deploy and use AI systems in a way that adheres to the ethical principles of respect for human autonomy, prevention of harm, fairness and explicability. Acknowledge and address the potential tensions between these principles.
- Pay particular attention to situations involving more vulnerable groups such as children, persons with disabilities and others that have historically been disadvantaged or are at risk of exclusion, and to situations which are characterised by asymmetries of power or

information, such as between employers and workers, or between businesses and consumers.

- Acknowledge that, while bringing substantial benefits to individuals and society, AI systems also pose certain risks and may have a negative impact, including impacts which may be difficult to anticipate, identify or measure (e.g. on democracy, the rule of law and distributive justice, or on the human mind itself.) Adopt adequate measures to mitigate these risks when appropriate, and proportionately to the magnitude of the risk.

3 MACHINE LEARNING

Machine Learning – 180 minutes
Keywords: ML Model, Supervised Learning, Unsupervised Learning, Reinforcement Learning, ML Training Data, ML Validation Data, ML Test Data, ML Algorithm, Hyperparameter, Data Pre-Processing, Overfitting, Underfitting, ML Prediction, ML Classification, ML Regression, Clustering, Adversarial Example, Adversarial Testing, Adversarial Attack.
Introduction to Machine learning TAI-3.1 Compare the supervised, unsupervised and reinforcement approaches to Machine Learning (K2) **The Machine Learning Workflow** TAI-3.2 Summarize the workflow used to create a ML system (K2) **Machine Learning Training and Test Data** TAI-3.3 Contrast the use of training data and test data in the development of a ML system (K2) **Overfitting and Underfitting in Machine Learning** TAI-3.4 Explain the differences between underfitting and overfitting when training a ML system (K2) **Bias and Fairness in the Training Data** TAI-3.5 Explain the problem of preventing unfairness in ML systems (K2) **Data Quality** TAI-3.6 Recall the different approaches for the labelling of training data for supervised learning (K1) TAI-3.7 Recognize how poor data quality can cause problems with the resultant ML model (K2) **Machine Learning Algorithm/Model Selection** TAI-3.8 Explain the factors involved in the selection of ML algorithms (K2) **Machine Learning Testing and Quality Assurance** TAI-3.9 Explain how adversarial testing is used to improve the robustness of AI-Based systems (K2) TAI-3.10 For a given scenario determine a strategy for the QA and testing to be performed as part of Machine Learning (K3)

3.1 INTRODUCTION TO MACHINE LEARNING

Machine learning (ML) is a form of AI, where the AI-Based system learns its behaviour from provided training data, rather than being explicitly programmed. The outcome of ML is known as a model, which is created by the AI development framework using a selected algorithm and the training data; this model reflects the learnt relationships between inputs and outputs. Often the created model, once initially trained, does not change in use. In contrast, in some situations, the created model can continue to learn from operational use (i.e. it is self-learning). Example uses of ML include image classification, playing

games (e.g. Go), speech recognition, security systems (malware detection), aircraft collision avoidance systems and autonomous cars.

There are three basic approaches to machine learning (ML). With supervised ML the algorithm creates the model based on a training set of labelled data. An example of supervised ML would be where the provided data were labelled pictures of cats and dogs and the created model is expected to correctly identify cats and dogs it sees in the future. Supervised learning solves two forms of problem – classification problems and regression problems. Classification is where the model classifies the inputs into different classes, such as 'yes – this module is error prone' and 'no – this module is not error prone'. Regression is where the model provides a value, such as 'the expected number of bugs in the module is 12'. As ML is probabilistic, we can also measure the likelihood of these classifications and regressions being correct (see section 4.1 on performance metrics for ML).

With unsupervised ML the data in the training set is not labelled and so the algorithm derives the patterns in the data itself. An example of unsupervised ML would be where the provided data was about customers and the system was used to find specific groupings of customers, which may be marketed to in a specific manner. Because the training data does not have to be labelled, it is easier (and cheaper) to source than the training data for unsupervised ML.

With reinforcement learning a reward function is defined for the system, which returns a higher result when the system gets closer to the required behaviour. Using feedback from the reward function, the system learns to improve its behaviour. An example of reinforcement learning would be a route planning system that used a reward function to find the shortest route.

3.2 The Machine Learning Workflow

The activities in the machine learning workflow are shown in Figure 1.

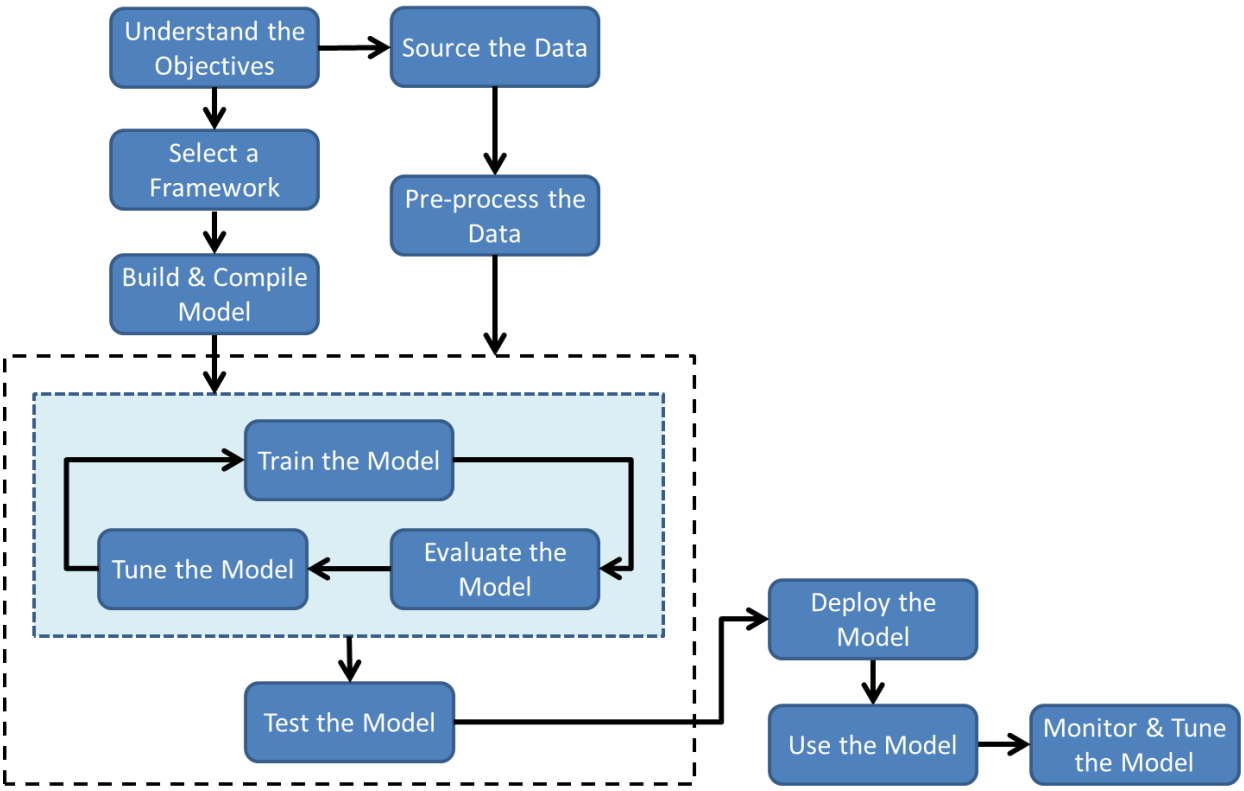

Figure 1: Machine Learning Workflow

The activities in the machine learning workflow are:

3.2.1 Understand the Objectives

The purpose of the ML model to be deployed needs to be understood and agreed with the stakeholders to ensure alignment with business priorities. Acceptance criteria (including performance metrics – see section 4.1) should be defined for the developed model.

3.2.2 Select a Framework

A suitable AI development framework should be selected based on the objectives, acceptance criteria and business priorities. These frameworks are introduced in section 1.9.

3.2.3 Build and Compile the Model

The model structure (e.g. number of layers) should be defined (it will typically be in source code, such as Python). Next, the model is compiled, ready to be trained.

3.2.4 Source the Data

The data used by the model will be based on the objectives. For instance, if the system is a real-time trading system, the data will come from the trading market. If the system is analysing customers' retail preferences for a marketing campaign, then the organization's customer big data will be the source.

The data used to train, tune and test the model should be representative of the operational data expected to be used by the model. In some cases, it is possible to use pre-gathered data sets for the

initial training of the model (e.g. see Kaggle datasets at https://www.kaggle.com/datasets). However, raw data normally needs some pre-processing.

3.2.5 Pre-Process the Data

The features in the data that will be used by our model need to be selected – these are the attributes or properties in the data that we believe are most likely to affect the outcome of the prediction. Training data may need to be managed to remove features that are not expected (or we don't want) to have any effect on the resultant model – this is called feature selection or feature engineering. By removing irrelevant information (noise), feature selection can reduce overall training times, prevent overfitting (see section 3.4.1), increase accuracy and make models more generalizable.

Real world data is likely to include outlier values, be in a variety of formats, be missing coverage of important areas, etc. Thus, pre-processing is normally required before it can be used to train (and test) the model. Pre-processing includes conversion of data to numeric values, normalizing numeric data to a common scale, detection and removal of outliers and noisy data, reducing data duplication and the addition of missing data.

3.2.6 Train the Model

A ML algorithm (e.g. see machine learning techniques in section 1.7) uses the training data to create and train the model. The algorithm should be selected based on the objectives, acceptance criteria and the available data.

Note that the activities of training, evaluation and tuning are shown explicitly in Figure 1 as being iterative, however ML is a highly iterative workflow and it may be necessary to return to any of the earlier activities, such as sourcing and pre-processing the data as a result of later activities (e.g. evaluating the model).

3.2.7 Evaluate the Model

The trained model is evaluated against the agreed performance metrics using validation data – and the results are then used to improve (tune) the model. Visualization of the results of the evaluation is normally required and different ML frameworks support different visualization options.

In practice several models are typically created and trained, and the best one chosen based on the results of the evaluation and tuning.

3.2.8 Tune the Model

The results from evaluating the model against the agreed performance metrics are used to adjust its settings to fit the data and so improve performance. The model may be tuned by hyperparameter tuning, where the training activity is modified (e.g. by changing the number of training steps or by changing the amount of data used for training), or attributes of the model are set (e.g. the number of neurons in a neural network or the depth of a decision tree).

3.2.9 Test the Model

Once a model has been trained, evaluated, tuned and selected it should be tested against the test data set to ensure that the agreed performance criteria are met. This test data should be completely independent of the training and validation data used up until this point in the workflow.

3.2.10 Deploy the Model

The tuned model typically needs to be re-engineered for deployment along with its related resources, including the relevant data pipeline – this is normally achieved through the ML framework. Targets include embedded systems and the cloud, where the model can be accessed via a web API.

3.2.11 Use the Model

Once deployed, the model, typically as part of a larger AI-Based system, can be used operationally. Models may perform scheduled batch predictions at set time intervals or may run on request in real-time.

3.2.12 Monitor & Tune the Model

While the model is being used, there is a danger that its situation may evolve (see section 2.1.3 on Evolution) and that the model may 'drift' away from its intended performance. To ensure that any drift is identified and managed, the operational model should be regularly evaluated against its acceptance criteria.

It may be deemed necessary to update the model to address the problem of drift or it may be decided that re-training with new data will result in a more accurate or more robust model, in which case a new model may be created and trained with updated training data. The new model may be compared against the existing model using a form of A/B Testing (see section 6.3).

3.3 Machine Learning Training and Test Data

When performed supervised ML, two separate data sets (a training data set and test set) are used to prevent overfitting (see section 3.4.1). The test set is sometimes called the holdout set.

To support the iterative evaluation and tuning of the model, the training data set is split into two – data used for training and validation data used for evaluation. However, this can mean that there is now insufficient data for training. One way to address this problem is known as cross-validation, where the training data set is split into n equal parts known as folds. The model is then trained and evaluated n times – in each case a different fold is used as the validation set and the remaining folds are used as the training set. In this way training is improved and evaluation and tuning can still be performed.

3.4 Overfitting and Underfitting in Machine Learning

3.4.1 Overfitting

Overfitting occurs when the model learns incorrect relationships from extraneous information, such as insignificant details, random fluctuations and noise in the training data (i.e. too many features have been included in the training data). In effect it's as if the model has memorized the training data and when it is used operationally it works excellently with data that is very similar to the training data but finds it difficult to generalize and handle new data. One way to identify overfitting is to ensure an independent test set is used that is completely separate from the training data.

3.4.2 Underfitting

Underfitting occurs when the model is unable to identify the relationships between inputs and outputs from the training data. Underfitting usually occurs when there is insufficient training data to provide enough information to derive the correct relationships between inputs and outputs (i.e. not enough features included in the training data), but it can also occur when the selected algorithm does not fit the data (e.g. creating a linear model to work with non-linear data). This typically leads to a simplistic model that makes many incorrect predictions.

3.5 Bias and Fairness in the Training Data

Bias and fairness were introduced in section 2.1.5. If the training data includes inherent biases, then the derived model will probably include those same biases. Therefore, care must be taken to ensure data features that would lead to unfairness in the resultant model are not included. For instance, among others, the following features should be recognized as potentially causing unwanted bias:

- Gender
- Sexual orientation
- Age
- Race
- Religion
- Country of origin
- Educational background
- Source of income
- Home address

However, simply removing the above features from the training data does not necessarily solve the problem as there could well be other features (perhaps used in combination) that could still lead to an unfair model (e.g. whether parents were divorced can lead to racial stereotyping in some locations).

3.6 Data Quality

Supervised learning assumes that the training data is correct. However, in practice, it is rare for training data sets to be labelled correctly 100% of the time. Human labellers can make simple random mistakes (e.g. pressing the wrong button), systemic mistakes (e.g. all labellers were given the wrong instructions) and there is also the possibility that they make deliberate mistakes. Labels are not always simple classifications into one of two classes and more complex labelling tasks may mean the correct label is questionable. Labels in one language may be mis-translated into a second language. Labelling may be performed a number of ways, with each approach having inherent risks to data quality:

- Labelling by internal team
- Outsourced labelling
- Crowdsourced labelling

- Synthetic data generation
- AI-Based labelling
- Hybrid approaches

Data may be of poor quality if, for example, the input sensors are low quality or badly calibrated – and this is more often a problem when sensor data comes from multiple sources (e.g. laboratories using slightly different measurement approaches or a variety of IoT devices).

Missing data can take three main forms, and each has a different effect on the resultant model. If the data is missed completely at random it should have little effect given the probabilistic nature of the model (other than to reduce accuracy due to lack of data). If the data from a particular feature is missed (e.g. all data from females) then this is more likely to have an adverse effect on the resultant model (unless the model is not used to make predictions for females operationally). Worse still, and most difficult to detect, is the third case, where a specific set of data values from a given feature are missing (e.g. data from females aged 35 to 50). Such problems often occur in medical studies due to the nature of data collection. In this case the model is likely to be severely compromised.

3.7 Machine Learning Algorithm/Model selection

There is some controversy whether the selection of the algorithm, the model settings and hyperparameters is a science or an art [Henderson, Peter, et al. "Deep reinforcement learning that matters." Thirty-Second AAAI Conference on Artificial Intelligence. 2018.]. There is no definitive approach that would allow the selection of the optimal set purely from an analysis of the problem situation – in practice this selection is nearly always partly by trial and error (as shown in the explicitly iterative part of the machine learning workflow in Figure 1).

The information needed to make this selection includes knowing what functionality the model is expected to provide, what data is available to the learning algorithm and the model, and what non-functional requirements must be met.

In terms of functionality, the model is typically going to provide classification, prediction of a value (regression), detection of anomalies or determination of structure from data (clustering). Knowing how much data is available may allow certain algorithms to be discarded (e.g. those that that rely on big data can be ignored if less data is available). If the data is labelled then supervised learning is possible, otherwise another approach is needed. The number of features that are expected to be used by the model will also point to the selection of certain algorithms, as will the number of expected classes for clustering. Non-functional requirements may include constraints on available memory (e.g. an embedded system), the speed of prediction (e.g. for real-time systems) and, for some situations, the speed of training the model with updated training data. Other non-functional areas of relevance may be the need for transparency (see section 2.1.7).

3.8 Machine Learning Testing and Quality Assurance

Testing has been mentioned in several of the preceding sections on ML. This section briefly identifies the quality assurance and testing opportunities directly related to ML.

3.8.1 Review of ML Workflow

The ML workflow that is used should be documented and followed when performing ML. Deviations from the workflow described in Figure 1 should be justified.

3.8.2 Acceptance Criteria

Acceptance criteria (including both functional and non-functional requirements) should be documented and justified for use on this application. Performance metrics should be included for the model.

3.8.3 Framework, Algorithm/Model and Parameter Selection

The choice of framework, algorithm, model, settings and hyperparameters should be documented and justified.

3.8.4 Model Updates

Whenever the deployed model is updated it should be re-tested to ensure it continues to satisfy the acceptance criteria, including tests against implicit requirements that may not be documented, such as testing for model degradation (e.g. the new model runs slower than the previous model). Where appropriate, A/B testing or back-to-back testing should be performed against the previous model.

3.8.5 Training Data Quality

ML systems are highly dependent on the training data being representative of the operational data and some ML systems have extensive operational environments (e.g. those used in autonomous vehicles). Boundary conditions (edge cases) are known to cause failures in all types of system (AI and non-AI) and should be included in the training data. The selection of training data in terms of the size of the data set and characteristics such bias, transparency and completeness should be documented and justified and confirmed by experts for more critical systems.

3.8.6 Test Data Quality

The criteria for the training data apply equally to the test data, with the caveat that the test data must be as independent of the training data as possible. The level of independence should be documented and justified. Test data should be systematically selected and/or created and should also include negative tests (e.g. inputs outside the expected input range) and adversarial tests (see section 3.8.8 for details).

3.8.7 Model Integration Testing

Integration testing should be performed to ensure the ML model is correctly integrated with the remainder of the AI-Based system of which it is a part. For instance, tests should be performed to check that the correct image file is passed to the model for object recognition and that it is in the format expected by the model. Tests should also be performed to check that the output of the model is correctly interpreted and used by the rest of the system.

3.8.8 Adversarial Examples and Testing

An adversarial example is where an extremely small change made to the input to a neural network produces an unexpected (and wrong) large change in the output (i.e. a completely different result than

for the unchanged inputs). Adversarial examples were first noticed with image classifiers. By simply changing a few pixels (not visible to the human eye) it is possible to persuade the neural network to change its image classification to a very different object (and with a high degree of confidence). Note, however, that adversarial examples are not limited to image classifiers, but are an attribute of neural networks in general, and so apply to any use made of neural networks.

Adversarial examples are generally transferable. This means that an adversarial example that causes one neural network to fail will often cause other neural networks to fail that are trained to perform the same task. Note that these other neural networks may have been trained with different data and based on different architectures, but they are still prone to failure with the same adversarial examples.

Adversarial testing is often referred to as performing adversarial attacks. By performing these attacks and identifying vulnerabilities during testing, measures can be taken to protect against future failures and so the robustness of the neural network is improved.

Attacks can be made when training the model and then on the trained model (neural network) itself. Attacks during training can include corrupting the training data (e.g. modifying labels), adding bad data to the training set (e.g. unwanted features) and corrupting the learning algorithm. Attacks on the trained model can be white box or black box and involve identifying adversarial examples that will force the model to give bad results.

With white box attacks, the attacker has full knowledge of the algorithm used to train the model and also the settings and parameters used. The attacker uses this knowledge to generate adversarial examples by, for instance, making small perturbations in inputs and monitoring which ones cause large changes to the model.

With black box attacks, the attacker has no access to the model's internal workings or knowledge of how it was trained. In this situation, the attacker initially uses the model to determine its functionality and then builds a 'duplicate' model that provides the same functionality. The attacker then uses a white box approach to identify adversarial examples for this duplicate model. As adversarial examples are generally transferable, the same adversarial examples will normally also work on the (black box) model.

4 MACHINE LEARNING PERFORMANCE METRICS AND BENCHMARKS

Machine Learning Performance Metrics and Benchmarks – 90 minutes
Keywords: Confusion Matrix, True Positive, False Positive, True Negative, False Negative, Accuracy, Precision, Recall, F1-Score, ML Benchmark Suite.
Machine Learning Performance Metrics TAI-4.1 Give examples of the selection of different ML performance metrics to achieve different objectives (K2) TAI-4.2 Calculate the accuracy, precision, recall and F1-Score from a given set of ML performance data (K3) **Benchmarks for Machine Learning** TAI-4.3 Explain the use of benchmark suites for measuring ML performance (K2)

4.1 MACHINE LEARNING PERFORMANCE METRICS

Different performance metrics are used to evaluate different Machine Learning (ML) algorithms. This document is limited to covering performance metrics for classification problems. These metrics are initially agreed at the start of the ML workflow and then typically evaluated in two places in the ML workflow. For evaluation, they are used by developers to tune their models (e.g. by the selection of parameters) until they achieve an acceptable level of performance with their evaluation data set. The metrics are subsequently used to measure the acceptability of the performance of the final model with the (independent) test set.

4.1.1 Confusion Matrix

Imagine inputs are classified by the ML model as either true or false. In an ideal world, all data would be correctly classified, however, in reality the data sets will occasionally overlap meaning that some data points, which should be classified as true will be wrongly classified as false (a false negative) and some data points, which should be classified as false will be wrongly classified as true (a false positive). The remaining data points will be correctly classified as either a true negative or a true positive. These four sets of data can be represented in a confusion matrix.

4.1.2 Accuracy

Accuracy measures the proportion of all classifications that were correct, thus:

$$Accuracy = \frac{\text{true positives} + \text{true negatives}}{\text{true positives} + \text{true negatives} + \text{false positives} + \text{false negatives}}$$

4.1.3 Precision

Precision measures the proportion of predicted positives that were correct (how sure you are of your predicted positives), thus:

$$Precision = \frac{\text{true positives}}{\text{true positives + false positives}}$$

4.1.4 Recall

Recall (or Sensitivity) measures the proportion of actual positives that were predicted correctly (how sure you have not missed any positives), thus:

$$Recall = \frac{\text{true positives}}{\text{true positives + false negatives}}$$

4.1.5 F1-Score

The F1-Score provides a balance (the harmonic average) between Recall and Precision, thus:

$$F1 - Score = 2 * \frac{\text{Precsion} * \text{Recall}}{\text{Precsion + Recall}}$$

4.1.6 Selection of Performance Metrics

Different performance metrics are appropriate depending on the situation. Only the most basic and most common are covered here. Other metrics include BLEU/ROUGE and mAP.

Accuracy is suitable when the data sets are symmetric, for instance, when the counts of false negatives and false positives are similar.

Precision is most useful when you want to be sure of your true positives (i.e. we want few or no false positives). An example of this could be a military drone attacking terrorist targets. In this scenario we want no innocent bystanders to be wrongly identified as terrorists. This means we want no (or very few) false positives – and so Precision should be high.

Recall is most useful when catching true positives is important (i.e. we need to sure of all or most negatives). An example of this could be for an autonomous car sensing people in the road ahead. If there is a pedestrian, we want to be sure to identify them and therefore we need no (or few) false negatives – so Recall needs to be very high

F1 is most useful when the data distribution is uneven.

These performance metrics will provide the average performance of a ML model, however, in most situations it is also important to assure the performance of the model in expected worst case scenarios.

4.2 Benchmarks for Machine Learning

Ideally experts would be used to evaluate each new ML system, but that's normally too expensive. Instead, "representative" industry-standard benchmark suites are available, which include diverse workloads to cover a wide range of situations (e.g. image classification, object detection, translation and recommendation).

These benchmark suites can be used to measure the performance of both hardware (using defined models) and software (e.g. to determine the fastest models). Software benchmark suites can measure training (e.g. how fast a framework can train a ML model using a defined training data set to a specified target quality metric, such as 75% accuracy) and inference (e.g. how fast a trained ML model can perform inference).

Examples of ML sets of benchmarks are provided by MLPerf, which provides benchmarks for software frameworks, hardware accelerators and ML cloud platforms, and DAWNBench, which is a benchmark suite from Stanford University. The OAEI (Ontology Alignment Evaluation Initiative) is a coordinated international initiative with the goals of:

- assessing strengths and weaknesses of alignment/matching systems;
- comparing performance of techniques;
- increasing communication among algorithm developers;
- improving evaluation techniques;
- helping to improve the work on ontology alignment/matching.

through the controlled experimental evaluation of the techniques' performances.

5 Introduction to the Testing of AI Systems

Introduction to the Testing of AI Systems – 45 minutes
Keywords: Probabilistic System, Self-Learning System, Test Oracle Problem.
Challenges in Testing AI-Based Systems TAI-5.1 Explain how system specifications for AI-Based systems can create challenges with testing (K2) TAI-5.2 Recall those factors associated with test input data that can make testing AI-Based systems difficult (K1) TAI-5.3 Compare and contrast the difficulties associated with testing probabilistic & non-deterministic AI-Based systems (K2) **The Test Oracle Problem for AI-Based Systems** TAI-5.4 Explain how AI-Based systems are affected by the test oracle problem (K2)

5.1 Challenges in Testing AI-Based Systems

AI-Based systems are typically made up of conventional components (e.g. a user interface) and AI components (e.g. a Machine Learning model). Also, even 'pure' AI components are implemented in software and so can suffer the same defects as any other software. Thus, when testing an AI-Based system, conventional software testing approaches are still required. However, AI-Based systems include a number of special attributes that can make additional testing necessary than for conventional software systems:

5.1.1 System Specifications

Despite the amount of recent academic research conducted on AI (and ML in particular), there is little coverage of how best to specify the expected behaviour of AI-Based systems with their special characteristics (see section 2.1).

In an ideal world, complete formal specifications would be available, so allowing the creation of automated test oracles. In reality, the specifications for AI-Based systems are likely to be incomplete and informal, which requires testers to determine unspecified expected results. This can be problematic if the testers are not fully cognizant of the required system behaviour and it is difficult to get this information from domain experts.

Another problem is that AI-Based systems are often specified in terms of objectives rather than required functionality, which is a more conventional approach. This is because the nature of many AI-Based systems is such that the functionality provided is opaque (e.g. it is very difficult to imagine how a deep neural network functions).

Some AI-Based systems have extensive operational environments (e.g. an autonomous drone) and fully defining the operational environment can be more challenging than for a typical conventional system.

Note that the complexity of the operational environment normally means the test environments for these systems can be equally challenging (see section 8 for more details on test environments).

The specifications for ML models should contain a set of required performance metrics (see section 4.1) to act as acceptance criteria for the ML models.

5.1.2 Test Input Data

AI-Based systems often depend on big data inputs and/or inputs from a large range of sources. This can mean that input data is often unstructured and provided in diverse formats. When developing AI-Based systems managing this data is a specialist task of a Data Scientist, but when it comes to the testing, this specialist data management task is one of several performed by the tester, often with little or no specialist training.

When there are requirements for sets of test data that could contain personal information, as is often the case for AI-Based systems, testers are required to source sanitized test data to meet data privacy regulations, such as GDPR. Care must be taken that the level of sanitization is sufficient to prevent the AI-Based system under test from inferring personal details that are only partially hidden.

5.1.3 Probabilistic & Non-Deterministic Systems

Due to the probabilistic nature of many AI-Based systems, there is not always an exact value that can be used as an expected result. For instance, when an autonomous car plots a route around a stopped bus it does not need to calculate the optimal solution, but rather a solution that works (and is safe) - and so we accept sub-optimal, but good-enough solutions.

The nature of how AI-Based systems determine their route can also mean that they do not come up with the same result each time (e.g. their calculation may be based on a random seed, which results in different, but workable, routes each time). This makes such systems non-deterministic, which results in a lack of reproducibility and means that any regression tests need to have smarter expected results that take account in the variability due to the non-determinism.

In both cases, the uncertainty in actual results requires testers to derive more sophisticated expected results, perhaps including tolerances, than for conventional systems. Probabilistic AI-Based systems may also require the tester to run the same test multiple times to provide a statistically significant assurance that the system is working correctly (like a Monte Carlo experiment).

5.1.4 Self-Learning Systems

As AI technology becomes more advanced, more AI-Based systems will become available that can change their own behaviour over time. These may be self-adapting systems (able to reconfigure and optimize themselves) or full self-learning systems that can adapt themselves by learning from their past experiences. For both situations, it is likely that some tests that ran successfully on the original system will no longer be viable on the new, improved system. Although it may be relatively easy to identify which tests are no longer valid, it is far more difficult to ensure that new tests for the new functionality have been generated.

5.1.5 Complexity

Many AI-Based systems are extremely complex and difficult for humans to understand. For instance, a 'simple' car-driving neural network that was built over just 3 days was estimated to have about 27 million connections and 250,000 parameters. Another example of a complex ML system is one used to identify patterns in big data. These systems are used because they can find patterns that humans, even after much study, simply cannot find. If these AI-Based systems are so complex, then understanding them in sufficient depth to be able to generate expected results may be beyond many testers.

5.1.6 AI-Specific Characteristics (see section 2.1)

AI-Based systems have a number of special characteristics, some of which have already been covered in this section, but others are described in section 2.1, along with suggestions on how they may be tested.

5.2 The Test Oracle Problem for AI-Based Systems

A recurring challenge when testing AI-Based systems is the test oracle problem. Poor specifications, complex, probabilistic, self-learning and non-deterministic systems make the generation of expected results problematic.

Testing approaches and techniques that address the test oracle problem are described in section 6 on black box testing.

6 BLACK BOX TESTING OF AI-BASED SYSTEMS

Black Box Testing of AI-Based Systems – 120 minutes
Keywords: Combinatorial Testing, Pairwise Testing, Back-to-Back Testing, Differential Testing, A/B Testing, Metamorphic Testing, Metamorphic Relation.
Combinatorial Testing TAI-6.1 Understand how pairwise testing is used for AI-Based systems (K2) **Back-to-Back Testing** TAI-6.2 Explain how back-to-back testing is used for AI-Based systems (K2) **A/B Testing** TAI-6.3 Explain how A/B testing is applied to the testing of AI-Based systems (K2) **Metamorphic Testing** TAI-6.4 Explain how metamorphic testing can be applied to the testing of AI-Based systems (K2) TAI-6.5 Apply metamorphic testing to derive test cases for a given scenario (K3)

6.1 COMBINATORIAL TESTING

To prove – by dynamic testing – that a specific test item meets all requirements under all given circumstances, then all possible combinations of input values in all possible states would need to be tested. This impractical activity is referred to as 'exhaustive testing'. For that reason, in practice software testing derives test suites by *sampling* from the (extremely large) set of possible input values and states. Combinatorial testing is one systematic approach to deriving a useful subset of combinations from this input space.

The combinations of interest are defined in terms of parameters (i.e. inputs and environment conditions) and the values these parameters can take. Where numerous parameters (each with numerous discrete values) can be combined, this technique enables a significant reduction in the number of test cases required, ideally without compromising the defect detection ability of the test suite.

ISO/IEC/IEEE 29119-4 defines several combinatorial testing techniques, such as All Combinations, Each Choice Testing, Base Choice Testing and Pairwise Testing. In practice pairwise testing is the most widely used, mainly due to ease of understanding, ample tool support and research showing that most defects are caused by interactions involving few parameters.

The number of parameters of interest for an AI-Based system can be extremely high, especially when the system uses big data or interacts with the outside world, such as a self-driving car. Thus, a means of systematically reducing the almost infinite number of possible combinations to a manageable subset by using a combinatorial testing technique, such as pairwise testing, is extremely useful. In practice, even

the use of pairwise testing can still result in extensive test suites for such systems and the use of automation and virtual test environments (see section 8.1) often becomes necessary.

Using self-driving cars as an example, at a high level the scenarios for system testing need to consider both different vehicle functions and the environments in which they are expected to operate. Thus, the parameters would need to include the various self-driving functions (e.g. cruise control, adaptive cruise control, lane keeping assistance, lane change assistance, traffic light assistance, etc.) along with environment constraints (e.g. road types and surfaces, geographic area, time of day, weather conditions, traffic conditions, visibility, etc.). In addition to these parameters, inputs from sensor should be considered at varying levels of effectiveness (e.g. the input from a video camera will degrade as a journey progress and it gets dirtier or the accuracy of a GPS unit will change as different numbers of satellites come into and go out of line of sight). Research is currently unclear on the necessary level of rigour that would be required for the use of combinatorial testing with safety-critical AI-Based systems such as self-driving cars (e.g. pairwise may not be sufficient), but it is known that the approach is effective at finding defects and can also be used to estimate the residual level of risk.

6.2 Back-to-Back Testing

In back-to-back testing, an alternative version of the system (e.g. already existing, developed by a different team or implemented using a different programming language) is used as a pseudo-oracle to generate expected results for comparison from the same test inputs. This is sometimes known as differential testing.

As such, back-to-back testing is not a test case generation technique as test inputs are not generated. Only the expected results are generated automatically by the pseudo oracle (the functionally equivalent system). When used in partnership with tools for generating test inputs (random or otherwise) it becomes a powerful way to perform high-volume automated testing.

When back-to-back testing is used to support functional testing, the pseudo oracle does not have to meet the same non-functional constraints as the system under test. For instance, the pseudo oracle could run far slower than is required for the system under test. It is also not always necessary for the pseudo oracle to be a complete functionally equivalent system, as back-to-back testing can be performed with a pseudo oracle that is only equivalent to part of the system under test.

In the context of ML, it is possible to use different frameworks, algorithms and settings to create pseudo oracles (in some situations it is even possible to create a pseudo oracle using conventional, non-AI, software). A known problem with the use of pseudo oracles is that for them to work well they should be completely independent of the software under test. With so much reusable, open source software being used to develop AI-Based systems, this independence can be easily compromised.

6.3 A/B Testing

A/B testing is a statistical testing approach that allows testers to determine which of two systems performs better – it is sometimes known as split-run testing. It is often used for digital marketing (e.g. finding the email that gets the best response) in client-facing situations.

As an example, A/B testing is often used to optimize user interface design. For instance, the user interface designer hypothesises that by changing the colour of the ‘buy’ button from the current red to

blue, that sales will increase. A new variant of the interface is created with a blue button and the two interfaces are assigned to different users. The sales rates for the two variants are compared and, given a statistically significant number of uses, it is possible to determine if the hypothesis was correct. IF the blue button generated more sales, then the new interface with the blue button would replace the current interface with the red button. This form of A/B testing requires a statistically significant number of uses and can be time-consuming, although tools (often using AI) can be used to support it.

A/B testing is not a test case generation technique as test inputs are not generated. A/B testing is a means of solving the test oracle problem by using the existing system as a partial oracle. By comparing the new system with the current system, it is possible to determine if the new system is better in some way. When used for digital marketing, the measure of success may by more sales, but for an AI-Based system, such as a ML classifier, the performance metrics, such as accuracy, sensitivity and recall, could be used (see section 4.1).

A/B testing can be used whenever a component of an AI-Based system is updated, as long as acceptance criteria (e.g. 'specified performance metrics must improve or stay the same') are defined and agreed. If A/B testing is automated, then it can be used for testing self-learning AI-Based systems, by comparing the new performance of the system with its previous performance and reverting to the previous version if the self-learning has not improved the system performance. In this situation, care must be taken to ensure valid acceptance criteria are set.

6.4 METAMORPHIC TESTING

Metamorphic testing is an approach to generating test cases that deals, in part, with the test oracle problem often found with AI-Based systems, where generating expected results can be problematic (e.g. because of complexity, non-determinism and probabilistic systems). The main difference between test cases generated using metamorphic testing and conventional test case design techniques is that the expected results in metamorphic testing may not be a fixed value, but, instead, are defined by a relationship with another expected result.

Metamorphic testing uses metamorphic relations to generate follow-up test cases from a source test case that is known to be correct. A metamorphic relation for the software under test describes how a change in the test inputs from the source test case to the follow-up test case affects a change (or not) in the expected outputs from the source test case to the follow-up test case.

EXAMPLE 1 A test item measures the distance between a start and end point. The source test case has test inputs A (start point) and B (end point) and an expected result C (distance) from running the test case. The metamorphic relation states that if the start and end points are swapped, then the expected result remains unchanged. Thus, a follow-up test case can be generated with B as the start point, A as the end point and C as the distance.

EXAMPLE 2 A test item predicts the age of death for an individual based on a set of lifestyle parameters. A source test case has various test inputs, including 10 cigarettes smoked per day, and an expected result of age 58 years from running the test case. The metamorphic relation states that if a person smokes more cigarettes, then their expected age of death will probably decrease (and not increase). Thus, a follow-up test case can be generated with the same input set of lifestyle parameters, except with the number of cigarettes smoked increased to 20 per day. The expected result (the predicted age of death) for this follow-up test case can now be set to less than or equal to 58 years.

The expected result for the follow-up test case will not always be an exact value but will often be described as a function of the actual result achieved by executing the source test case (e.g. expected result for follow-up test case is greater than the actual result for source test case).

A single metamorphic relation can often be used to derive multiple follow-up test cases (e.g. a metamorphic relation for a function that translates speech into text can be used to generate multiple follow-up test cases using the same speech input file at different input volume levels but with the same text as the expected result). If metamorphic relations are stated formally (or semi-formally) and source test cases are provided, then it should be possible to automate the generation of follow-up test cases, although it is not possible to automate the generation of the metamorphic relations, which requires some domain knowledge.

The process for performing metamorphic testing is:

1. Construct metamorphic relations (MRs)
    - Identify properties of the program under test and represent them as metamorphic relations between test inputs and expected outputs, together with some method to generate a follow–up test case based on a source test case.
2. Review MRs
    - Review and confirm MRs with customers and/or users.
3. Generate source test cases
    - Generate a set of source test cases (using any testing technique or random testing).
4. Generate follow-up test cases
    - Use the metamorphic relations to generate follow–up test cases.
5. Execution of metamorphic test cases
    - Execute both the source and follow–up test cases, and check that the outputs do not violate the metamorphic relation. Otherwise, the metamorphic test case has failed, indicating a bug.

Metamorphic testing has been successfully used in a wide variety of AI-Based application areas, such as bioinformatics, web services, machine learning classifiers, search engines and security. Research shows that only 3 to 6 diverse metamorphic relations can reveal over 90% of the faults that could be detected using a traditional test oracle.

7 WHITE BOX TESTING OF NEURAL NETWORKS

White Box Testing of Neural Networks – 45 minutes
Keywords: Artificial Neural Network, Deep Neural Net, Activation Value, Backward Propagation, Neuron Coverage, Threshold Coverage, Sign Change Coverage, Value Change Coverage, Sign-Sign Coverage.
Structure of a Neural Network TAI-7.1 Explain the structure and working of a neural network (K2) **Test Coverage Measures for Neural Networks** TAI-7.2 Describe the different test coverage measures for neural networks (K2) TAI-7.3 Compare the test effectiveness of different test coverage measures for neural networks (K2) **White Box Testing Tools for Neural Networks** TAI-7.4 Identify tools supporting white box testing of neural networks (K1)

7.1 STRUCTURE OF A NEURAL NETWORK

A Neural Network is a computational model inspired by the neural network in a human brain. It comprises a number of layers of connected nodes or neurons, as shown in Figure 2. Note that in this section we will use as our example a feedforward neural network, which was the first and is the simplest type of artificial neural network – the only extra complexity we will add is that we will consider a network with multiple layers – known as a multi-layer perceptron (or deep neural net as it has hidden layers).

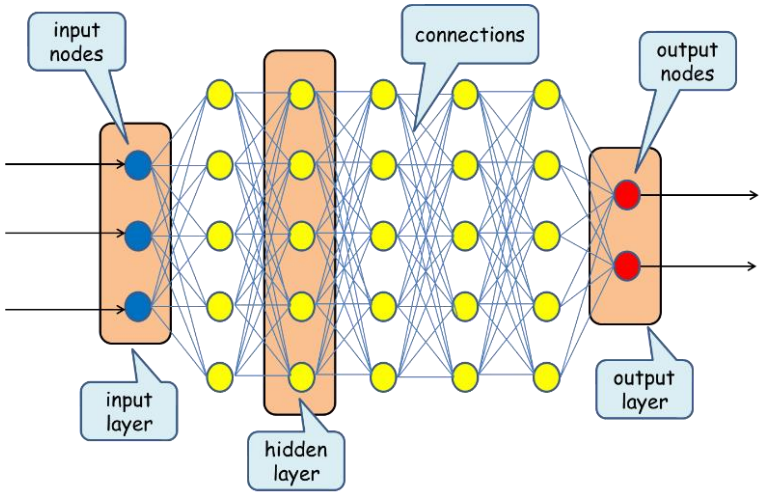

Figure 2: Deep Neural Net

The input nodes receive information from the outside world (e.g. each input may be a value for a pixel in an image), and the output nodes provide information to the outside world (e.g. a classification). The

nodes in the hidden layers have no connections to the outside world and perform the computations that pass information from the input nodes to the output nodes.

Each neuron accepts input values and generates output values, known as activation values (or output vectors), which can be positive, negative or zero (with a value of zero, a neuron has no influence on downstream neurons). Each connection has a weight and each neuron has a bias. The activation values are calculated by a formula based on the input activation values, the weights of the input connections and the bias of the neuron.

For supervised learning, the network learns by use of backward propagation. Initially all nodes are set to an initial value and the first input training data is passed into and through the network. The output is compared with the known correct answer and the difference between the calculated output and the correct answer (the error) is fed back to the previous layer of the network and is used to modify the weights. This backward error propagation goes back through the whole network and each of the connection weights is updated as appropriate. As more training data is fed into the network it gradually learns from the errors until it is considered ready for evaluation with the validation data, which will determine the performance of the trained network.

7.2 Test Coverage Measures for Neural Networks

Traditional coverage measures are not really useful for neural networks as 100% statement coverage is typically achieved with a single test case. The defects are normally hidden in the neural network itself. Thus, different coverage measures have been proposed based on the activation values of the neurons (or pairs of neurons) in the neural network when the neural network is tested.

7.2.1 Neuron Coverage

Neuron coverage for a set of tests is defined as the proportion of activated neurons divided by the total number of neurons in the neural network (normally expressed as a percentage). For neuron coverage, a neuron is considered to be activated if its activation value exceeds zero.

7.2.2 Threshold Coverage

Threshold coverage for a set of tests is defined as the proportion of neurons exceeding a threshold activation value divided by the total number of neurons in the neural network (normally expressed as a percentage). For threshold coverage, a threshold activation value between 0 and 1 must be chosen as the threshold value. Note that this threshold coverage corresponds to 'neuron coverage' in the DeepXplore tool.

7.2.3 Sign Change Coverage

Sign Change coverage for a set of tests is defined as the proportion of neurons activated with both positive and negative activation values divided by the total number of neurons in the neural network (normally expressed as a percentage). An activation value of zero is considered to be a negative activation value [Sun et al, Testing Deep Neural Nets paper].

7.2.4 Value Change Coverage

Value Change coverage for a set of tests is defined as the proportion of neurons activated where their activation values differ by more than a change amount divided by the total number of neurons in the neural network (normally expressed as a percentage). For value change coverage, a value between 0 and 1 should be chosen as the change amount.

7.2.5 Sign-Sign Coverage

Sign-Sign coverage for a set of tests is achieved if each neuron by changing sign (see 7.2.3) can be shown to individually cause one neuron in the next layer to change sign while all other neurons in the next layer stay the same (i.e. they do not change sign). In concept, this level of neuron coverage is similar to modified condition/decision coverage (MC/DC) [Sun et al, Testing Deep Neural Nets paper].

7.2.6 Layer Coverage

Coverage measures can also be defined based on whole layers of the neural network and how the activation values for the set of neurons in a whole layer change (e.g. absolutely or relative to each other). Further research is needed in this area.

7.2.7 Test Effectiveness of the White Box Measures

There is currently little data on the test effectiveness of the different white box coverage measures for the white box testing of neural networks. However, it is generally true that criteria requiring more tests will find more defects than those that require fewer tests, so allowing the relative effectiveness of the measures to be deduced. Several subsumes relationships can be derived from the coverage measures described in sections 7.2.1. to 7.2.5. All other measures subsume neuron coverage and sign-sign coverage also subsumes sign change coverage.

Although easy to understand, achieving high levels of neuron coverage can normally be achieved using only a few test cases, so limiting its test effectiveness. Early results for threshold coverage appear to show that this may be a useful measure for generating tests that cover defect-inducing corner cases, but the threshold value may need to be set individually for each neural network. For value change coverage, higher values for the change amount will naturally require more test cases. Sign-sign coverage is normally the most rigorous of the coverage criteria specified here. [Sun et al, Testing Deep Neural Nets paper]

7.3 White Box Testing Tools for Neural Networks

Commercial tools are not currently available to support the white box testing of neural networks, however there are several research tools, including:

- DeepXplore – specifically for testing deep neural nets, proposes a white-box differential testing (back-to-back) algorithm to systematically generate adversarial examples that cover all neurons in the network (threshold coverage).
- DeepTest – systematic testing tool for automatically detecting erroneous behaviours of cars driven by deep neural nets. Supports the Sign-Sign coverage for DNNs.
- DeepCover - provides all the levels of coverage defined in this section.

8 TEST ENVIRONMENTS FOR AI-BASED SYSTEMS

Test Environments for AI-Based Systems – 45 minutes
Keywords: Virtual Test Environment, Simulator, Fuzz Testing, Parameterized Test Scenario.
Test Environments for AI-Based Systems TAI-8.1 Describe the main factors that differentiate the test environments for AI-Based systems from those required for conventional systems (K2) TAI-8.2 Understand the benefits provided by virtual test environments in the testing of AI-Based systems (K2) TAI-8.3 Recall reusable simulators used for the development and testing of AI-Based systems (K1) **Test Scenario Derivation** TAI-8.4 Recall the options available for deriving test scenarios (K1) **Regulatory Test Scenarios and Test Environments** TAI-8.5 Recall the options for regulating safety-related AI-Based systems (K1)

8.1 TEST ENVIRONMENTS FOR AI-BASED SYSTEMS

The test environments for AI-Based systems have much in common with those for conventional systems: typically, the development environment at unit level and a production-like test environment at system and acceptance levels. Machine learning models, when tested in isolation, are typically tested within their development framework, as described in section 3.8.

There are two main factors that differentiate the test environments for AI-Based systems from those required for conventional systems. First, the context in which many AI-Based systems operate means their environment can be large, complex and constantly changing. This can make testing in the real world extremely expensive if the full range of possible environments are to be tested, the test environments are expected to be realistic and the testing is to be performed within a sensible timescale. Second, those AI-Based systems that can physically interact with humans have a safety component, which can make testing in the real world dangerous. Both factors indicate the need for the use of virtual test environments.

Virtual test environments provide the following benefits, among others:

- The use of a virtual environment ensures that dangerous scenarios can be tested in safety without causing damage to the system under test or any objects in its environment, such as vehicles, buildings, animals and humans. Tests in virtual environments are typically also better for the real-world environment.
- Virtual environments do not need to run in real-time – they can be run much faster with suitable processing power - meaning that many tests can be run in a short time period, potentially decreasing time-to-market by a large amount. A single system can also be tested on many virtual test environments running in parallel, perhaps in the cloud.

- Virtual environments can be cheaper to set up and run than their real-world counterparts. For instance, testing mobile phone communications across widely different urban environments is far cheaper when performed in a laboratory with virtual phones, transmitters and landscapes rather than with real phones being driven around a mix of locations, largely because only the relevant features need to be included in the virtual test environment. However, it should be noted that some virtual test environments must be truly representative and closely match the real-world in some respects. For instance, the testing of pedestrian avoidance in autonomous vehicles can require high levels of image representativeness.
- Sometimes, creating unusual (edge-case) scenarios can be very difficult in the real world and virtual environments allow the creation of such scenarios (and the ability to run multiple variants of these unusual scenarios many times). Virtual environments provide the tester with a greater level of control than they would have with real-word testing. These tests can also incorporate a level of randomness, such as by including AI-Based humans in autonomous car testing.
- By supporting the simulation of hardware, virtual environments allow systems to be tested with hardware components even when these components are not physically available (perhaps they have not been built yet) and they allow different hardware solutions to be trialled and compared inexpensively.
- Virtual environments provide excellent observability, so that all aspects of the system under test's response to a scenario can be measured and, where necessary, subsequently analysed.
- Virtual environments can be used to test systems that cannot be tested in their real operational environment, such as a robot working on the site of a nuclear accident or a system to be used for space exploration.

Virtual testing can be performed on simulators built specifically for a given system, but reusable simulators for specific domains are available both commercially and open source, for instance:

- Morse, the Modular Robots Open Simulation Engine, a simulator for generic mobile robot simulation (single or multi robots), based on the Blender game engine;
- AI Habitat, a simulation platform created by Facebook AI, designed to train embodied agents (such as virtual robots) in photo-realistic 3D environments;
- DRIVE Constellation, an open and scalable platform for self-driving cars from NVIDIA based on a cloud-based platform, capable of generating billions of miles of autonomous vehicle testing.

8.2 Test Scenario Derivation

For the systematic testing of an AI-Based system, test scenarios need to be generated to test individual AI components, the interaction of these components with the rest of the system, the complete system of interacting components, and the system interacting with its environment.

Test scenarios can be derived from several sources:

- System requirements

- User issues
- Automatically reported issues (e.g. for autonomous systems)
- Accident reports (e.g. for physical systems)
- Insurance data (e.g. for insured systems, such autonomous cars)
- Regulatory body data (e.g. collected through legislation)
- Testing at various levels (e.g. test failures or anomalies on the test track or on real roads could generate interesting test scenarios for an autonomous car at other test levels, while a sample of test scenarios run on the virtual test environment should also be run on real roads to validate representativeness of the virtual test environment)

An option using combinatorial testing for the generation of test scenarios for the system testing of autonomous cars is described in section 6.1. Metamorphic testing (see section 6.4) and fuzz testing could also be used to generate test scenarios.

8.3 Regulatory Test Scenarios and Test Environments

In the case of safety-related AI-Based systems, some level of regulation should apply to the systems. Two options are generally available to government for this regulation; it can allow the development organization to self-regulate or a regulatory body is set up to provide independent assurance that the systems meet minimum standards (a certification approach).

If a certification approach is followed, then the testing approach will need to be shared between the regulatory body and those providing the systems for certification (as it is for the crash testing of cars). A core part of the approach will be shared test environment definitions and shared test scenarios that can be run using test automation on those environments. The core set of shared test scenarios will need to be parameterized to allow new scenarios to be generated by varying the parameter values for each test to prevent overfitting and the regulatory body will also keep a set of private test scenarios that are not shared. The parameterization and the private scenarios should ensure that systems are not built just to pass known tests, and this approach also allows the regulatory body to add new scenarios as they become aware of potential problem situations from actual use of the systems.

9 USING AI FOR TESTING

Using AI for Testing - 30 minutes
Keywords: Probabilistic Software Engineering, Search Based Software Engineering, Exploratory Testing, Automated Exploratory Testing.
Introduction to AI-Driven Testing **Forms of AI used for Testing** TAI-9.1 Recall the three main forms of AI that prove useful to software testing (K1) **Test Types Supported by AI** TAI-9.2 List the test types supported by AI (K1) **Example Use Cases of AI for Testing** TAI-9.3 Describe examples of how AI is used to support testing (K2)

9.1 INTRODUCTION TO AI-DRIVEN TESTING

For many years software testers have been trying to increase the amount of automation in software testing, but, despite ongoing efforts, the rate of increase has been far slower than expected. A major problem has been that the changes that need to be made to the automation scripts when the software under test is changed are still largely performed manually, as is the initial script writing. However, with recent advances in AI technology, it should be possible to automate more of the manual testing tasks.

The main focus of this document is the testing of AI-Based systems. AI can also be used as part of the tools that perform software testing (of any form of software, including AI-Based systems). This section briefly introduces the concept and shows how AI can be used to support software testing through some example use cases.

9.2 FORMS OF AI USED FOR TESTING

According to Prof Mark Harman (of UCL and Facebook), there are three main forms of AI that prove useful to software engineering (and testing). These are:

- Probabilistic Software Engineering – AI can handle real world problems which are, by their nature, fuzzy and probabilistic. For instance, AI can be used to analyse and predict user behaviours, because of the stochastic nature of human behaviour.
- Classification, Learning and Prediction – AI can be used for predicting costs as part of project planning. For instance, machine learning techniques can be used for defect prediction.
- Search Based Software Engineering (SBSE) - AI is used to solve optimisation problems using computational search of large and complex search spaces. Search Based Software Testing may be the most successful and widely applicable areas of SBSE, and examples include

identifying the smallest set of test cases that achieve a given coverage criterion and prioritising regression test cases.

9.3 Test Types Supported by AI

There are few areas of software testing that cannot be supported by the use of AI. Here are some example uses of AI for testing:

- Specification Review – AI interprets specifications written in natural language and identifies potential anomalies and defects. Similarly, AI-Based model checkers can be used to find problems in formal and semi-formal models.
- Specification-Based Script Generation – AI interprets natural language specifications and AI can subsequently generate a test script to provide coverage of the generated model.
- Test Strategy Assistant – AI learns about risk by analysis of big data collected from multiple projects.
- Usage Profiling - AI predicts future use of systems through the analysis of big data collected from system use.
- Exploratory Testing – AI learns and applies heuristics from observation of human exploratory testing sessions.
- Crowd Testing – AI analyses responses from multiple crowd testers to identify the most useful feedback and remove duplicate feedback.
- Defect Management – AI classifies and prioritises defects objectively.
- User Interface Verification – AI compares web pages and identify perceptible differences, ignoring invisible rendering, size and position differences.
- Web App Spidering (differential testing) – AI identifies issues by repeatedly crawling around the web app collecting data, such as screenshots and load times and comparing the current data with historical data to identify potential issues.
- Element Location – AI finds web elements, such as buttons and text boxes removing the need to hard code them into test scripts (e.g. as applied in Appium).

9.4 Example Use Cases of AI for Testing

9.4.1 Bug Prediction

A bug prediction tool provides guidance on which parts of a system are most likely to contain defects (e.g. a form of classification into 'worth testing' or 'not worth testing') in order to help test prioritization. Several factors could influence the result, such as:

- Source code metrics
    - Lines of code
    - Number of comments
    - Cyclomatic complexity

    - Module dependencies
- Process metrics
    - Revisions made to module
    - Times refactored
    - Times fixed / when fixed
    - Lines of code changed (code churn)
    - Module age
- People and organizational metrics
    - Number of authors
    - Developer experience

Note that because there are so many potential factors (all the above plus more), determining the relationship between the factors and propensity to bugs is beyond human capabilities. Also, no two situations are the same, so the creation of a Machine Learning (ML) model for bug prediction that can be reused in all projects and all organizations is infeasible. Instead, ML for bug prediction needs to be performed using the available data within the local project or organization to create a model that is optimized for that use.

Bug prediction using ML has been successfully used in several different situations (e.g. Tosun, 2010 and Kim, 2007). The best predictors have been found to be people and organizational measures rather than the more widely used source code metrics.

9.4.2 Static Analysis

Static analyzers are automated tools that detect anomalies in source code by scanning programs without running them. Although not bug predictors they provide similar guidance – typically identifying where they believe there may be problems in the code. Facebook's Infer tool uses a form of Abstract Interpretation to analyse C, Objective-C and Java on both Android and iOS. It works fast compared to conventional static analysers and can analyse millions of lines of code in a few minutes, which makes it ideal for use with continuous integration. Facebook claim that approximately 80% of flagged anomalies are fixed, so the false alarm rate is relatively low. It is successfully used in a number of different organizations including Facebook, Instagram, Uber and Spotify, and is open source.

9.4.3 Regression Test Optimization

Regression testing is performed to ensure that changes to a system cause no unwanted side-effects to unchanged parts of the system – it is an exercise in re-executing previously run tests and checking that the results have not changed. As such, it is a largely repetitive task that should be automated wherever technically and economically feasible. As changes are made to a system, new tests are created and run, and the successful tests become candidates for adding to the regression test suite. In this manner, regression test suites tend to grow over time, sometimes leading to situations where they take a lot of effort to run (so costing a lot – up to 80% of the testing budget) and a lot of time to run (so delaying the release of updated software).

To prevent regression test suites growing too large, they should be frequently 'pruned' to leave only the most effective and efficient tests with no duplication. This is known as regression test optimization and should result in a regression test suite with high fault detection capability and a low cost of execution.

However, a lack of knowledge on the part of testers and poor system documentation can make this pruning activity expensive and time-consuming to perform as a manual activity.

For an AI-Based tool to perform regression test optimization, it needs to gather information on test cases, such as the following:

- tests that found defects previously
- tests that exercise recently changed code
- tests that address high risk areas
- test coverage achieved by a test
- execution time used by a test
- requirements covered by a test

Using this, and other available information, the tool can then create an optimized regression test suite that, ideally, still finds most regression faults but takes less time to run. Such AI-Based regression test optimization tools use techniques to select, prioritize and even augment test cases to create a more effective and efficient regression test suite. Results from research show that reductions of 50% in the size of a regression test suite can be achieved while still detecting most defects, and reductions of 94% can be achieved while still achieving full requirements coverage.

9.4.4 Test Input Generation

A tool that automatically generated a set of test inputs to achieve a test completion goal would be useful to many test designers. Typical completion goals for such tools tend to be structural coverage criteria, such as statements and branch coverage, mainly because these criteria are required by regulatory standards, mainly for safety-related software.

The technology used for such tools includes the use of symbolic execution and search-based AI. Several tools are available in this area, such as the open source tools AUSTIN for C and EVOSUITE for Java, and Intellitest for C#, which is part of Visual Studio.

Empirical studies show that test input generation tools are good at increasing coverage at little or no extra cost, but there appears to be no measurable improvement in fault detection by using them.

Outside of achieving coverage criteria required by regulatory standards, these tools have great potential for the automated creation of regression test suites. Often regression testing must be done, but no regression test suite is available. A ‘complete’ set of test inputs can be automatically generated for the original program using a test input generation AI-Based tool. Those test inputs can then be run through the original program to create a corresponding set of actual results. Together these inputs and actual results can be combined to form the basis of the regression test suite.

9.4.5 Automated Exploratory Testing

The testing of mobile apps is largely manual, although, if developing an Android app, Android Monkey is available to randomly generate test inputs using fuzz testing to execute tests looking for results such as an ‘application not responding’ (ANR) or a simple system crash. As these tests take little, or no, preparation most app developers run these random tests to identify serious problems with their apps. However, a problem with Android Monkey is that the sequences of inputs required to cause a problem

can be very large, meaning that many developers discount such problems as being either too difficult to debug or unlikely to correspond to real world use.

AI-Based tools can improve upon the random approach by automatically generating sequences of test inputs based on the user interface, code coverage and heuristics. Available tools for performing automated exploratory testing of Android apps include Dynodroid and Sapienz. Dynodroid employs heuristics to reduce the number of inputs and events necessary to reach comparable code coverage as that of Android Monkey. Sapienz uses a multi-objective approach of maximising code coverage and increasing fault revelation, while minimising the length of fault-revealing test sequences. Results show that the AI-Based tools can achieve equivalent levels of coverage and find more defects while reducing the average sequence of steps from an average of around 15,000 for Android Monkey to around 100.