Certified Tester AI Testing (CT-AI) Syllabus

Version 1.0

International Software Testing Qualifications Board

Provided by

Alliance for Qualification, Artificial Intelligence United, Chinese Software Testing Qualifications Board, and Korean Software Testing Qualifications Board

Copyright Notice

Revision History

Version	Date	Remarks
1.0	2021/10/01	Release for GA

Table of Contents

Acknowledgements

This document was formally released by the General Assembly of the ISTQB® on October 1st, 2021.

It was produced by a team from the International Software Testing Qualifications Board: Klaudia Dussa-Zieger (chair), Werner Henschelchen, Vipul Kocher, Qin Liu, Stuart Reid, Kyle Siemens, and Adam Leon Smith.

The team thanks the authors of the three contributing syllabi;

- A4Q: Rex Black, Bruno Legeard, Jeremias Rößler, Adam Leon Smith, Stephan Goericke, Werner Henschelchen
- AiU: Main authors Vipul Kocher, Saurabh Bansal, Srinivas Padmanabhuni and Sonika Bengani and co-authors Rik Marselis, José M. Diaz Delgado
- CSTQB/KSTQB: Qin Liu, Stuart Reid

The team thanks the Exam, Glossary and Marketing Working Groups for their support throughout the development of the syllabus, Graham Bath for his technical editing and the Member Boards for their suggestions and input.

The following persons participated in the reviewing and commenting of this syllabus:
Laura Albert, Reto Armuzzi, Árpád Beszédes, Armin Born, Géza Bujdosó, Renzo Cerquozzi, Sudeep Chatterjee, Seunghee Choi, Young-jae Choi, Piet de Roo, Myriam Christener, Jean-Baptiste Crouigneau, Guofu Ding,Erwin Engelsma, Hongfei Fan, Péter Földházi Jr., Tamás Gergely, Ferdinand Gramsamer, Attila Gyúri, Matthias Hamburg, Tobias Horn, Jarosław Hryszko, Beata Karpinska, Joan Killeen, Rik Kochuyt, Thomas Letzkus, Chunhui Li, Haiying Liu, Gary Mogyorodi, Rik Marselis, Imre Mészáros, Tetsu Nagata, Ingvar Nordström, Gábor Péterffy, Tal Pe'er, Ralph Pichler, Nishan Portoyan, Meile Posthuma, Adam Roman, Gerhard Runze, Andrew Rutz, Klaus Skafte, Mike Smith, Payal Sobti, Péter Sótér, Michael Stahl, Chris van Bael, Stephanie van Dijck, Robert Werkhoven, Paul Weymouth, Dong Xin, Ester Zabar, Claude Zhang.

0 Introduction

0.1 Purpose of this Syllabus

This syllabus forms the basis for the ISTQB® Certified Tester AI Testing. The ISTQB® provides this syllabus as follows:

1. To member boards, to translate into their local language and to accredit training providers. Member boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To certification bodies, to derive examination questions in their local language adapted to the learning objectives for this syllabus.
3. To training providers, to produce courseware and determine appropriate teaching methods.
4. To certification candidates, to prepare for the certification exam (either as part of a training course or independently).
5. To the international software and systems engineering community, to advance the profession of software and systems testing, and as a basis for books and articles.

0.2 The Certified Tester AI Testing

The Certified Tester AI Testing is aimed at anyone involved in testing AI-based systems and/or AI for testing. This includes people in roles such as testers, test analysts, data analysts, test engineers, test consultants, test managers, user acceptance testers and software developers. This certification is also appropriate for anyone who wants a basic understanding of testing AI-based systems and/or AI for testing, such as project managers, quality managers, software development managers, business analysts, operations team members, IT directors and management consultants.

The Certified Tester AI Testing Overview [I03] is a separate document which includes the following information:

- Business outcomes for the syllabus
- Matrix of business outcomes and connection with learning objectives
- Summary of the syllabus
- Relationships among the syllabi

0.3 Examinable Learning Objectives and Cognitive Level of Knowledge

Learning objectives support the business outcomes and are used to create the Certified Tester AI Testing exams.

Candidates may be asked to recognize, remember, or recall a keyword or concept mentioned in any of the eleven chapters. The specific learning objectives levels are shown at the beginning of each chapter, and classified as follows:

- K1: Remember
- K2: Understand
- K3: Apply
- K4: Analyze

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

0.4 Hands-on Levels of Competency

The Certified Tester Specialist AI Testing includes hands-on objectives which focus on practical skills and competencies.

The following levels apply to hands-on objectives (as shown):

- H0: Live demo of an exercise or recorded video.
- H1: Guided exercise. The students follow a sequence of steps performed by the trainer.
- H2: Exercise with hints. The student is given an exercise with relevant hints so the exercise can be solved within the given timeframe, or students take part in a discussion.

Competencies are achieved by performing hands-on exercises, such shown in the following list:

- Demonstrate underfitting and overfitting (H0).
- Perform data preparation in support of the creation of an ML model (H2).
- Identify training and test datasets and create an ML model (H2).
- Evaluate the created ML model using selected ML functional performance metrics (H2).
- Experience of the implementation of a perceptron (H1).
- Use of a tool to show how explainability can be used by testers (H2).
- Apply pairwise testing to derive and execute test cases for an AI-based system (H2).
- Apply metamorphic testing to derive and execute test cases for a given scenario (H2).
- Apply exploratory testing to an AI-based system (H2).
- Discuss, using examples, those activities in testing where AI is less likely to be used (H2).
- Implement a simple AI-based defect prediction system (H2).

0.5 The Certified Tester AI Testing Exam

The Certified Tester AI Testing exam will be based on this syllabus. Answers to exam questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable, except for the Introduction and Appendices. Standards and books are included as references, but their content is not examinable, beyond what is summarized in the syllabus itself from such standards and books.

Refer to Certified Tester Specialist AI Testing “Overview” document for further details under section “Exam Structure”.

Entry Requirement Note: The ISTQB® Foundation Level certificate shall be obtained before taking the Certified Tester Specialist AI Testing exam.

0.6 Accreditation

An ISTQB® Member Board may accredit training providers whose course material follows this syllabus. Training providers should obtain accreditation guidelines from the Member Board or body that performs the accreditation. An accredited course is recognized as conforming to this syllabus and is allowed to have an ISTQB® exam as part of the course.

The accreditation guidelines for this syllabus follow the general Accreditation Guidelines published by the Processes Management and Compliance Working Group.

0.7 Level of Detail

The level of detail in this syllabus allows internationally consistent courses and exams. In order to achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the AI Testing Certified Tester.
- A list of terms that students must be able to recall.
- Learning and hands-on objectives for each knowledge area, describing the learning outcomes to be achieved.
- A description of the key concepts, including references to sources such as accepted literature or standards.

The syllabus content is not a description of the entire knowledge area for the testing of AI-based systems; it reflects the level of detail to be covered in Certified Tester Specialist AI Testing training courses. It focuses on introducing the basic concepts of artificial intelligence (AI) and machine learning in particular, and how systems based on these technologies can be tested.

0.8 How this Syllabus is Organized

There are eleven chapters with examinable content. The top-level heading for each chapter specifies the time for the chapter; timing is not provided below chapter level. For accredited training courses, the syllabus requires a minimum of 25.1 hours of instruction, distributed across the eleven chapters as follows:

- Chapter 1: 105 minutes Introduction to AI
- Chapter 2: 105 minutes Quality Characteristics for AI-Based Systems
- Chapter 3: 145 minutes Machine Learning (ML) – Overview

Chapter 4:	230 minutes	ML – Data
Chapter 5:	120 minutes	ML Functional Performance Metrics
Chapter 6:	65 minutes	ML – Neural Networks and Testing
Chapter 7:	115 minutes	Testing AI-Based Systems Overview
Chapter 8:	150 minutes	Testing AI-Specific Quality Characteristics
Chapter 9:	245 minutes	Methods and Techniques for the Testing of AI-Based Systems
Chapter 10:	30 minutes	Test Environments for AI-Based Systems
Chapter 11:	195 minutes	Using AI for Testing

1 Introduction to AI – 105 minutes

Testing Keywords

None

AI-Specific Keywords

AI as a Service (AIaaS), AI development framework, AI effect, AI-based system, artificial intelligence (AI), neural network, deep learning (DL), deep neural network, general AI, General Data Protection Regulation (GDPR), machine learning (ML), narrow AI, pre-trained model, super AI, technological singularity, transfer learning

Learning Objectives for Chapter 1:

1.1 Definition of AI and AI Effect

AI-1.1.1	K2	Describe the AI effect and how it influences the definition of AI.

1.2 Narrow, General and Super AI

AI-1.2.1	K2	Distinguish between narrow AI, general AI, and super AI.

1.3 AI-Based and Conventional Systems.

AI-1.3.1	K2	Differentiate between AI-based systems and conventional systems.

1.4 AI Technologies

AI-1.4.1	K1	Recognize the different technologies used to implement AI.

1.5 AI Development Frameworks

AI-1.5.1	K1	Identify popular AI development frameworks.

1.6 Hardware for AI-Based Systems

AI-1.6.1	K2	Compare the choices available for hardware to implement AI-based systems.

1.7 AI as a Service (AIaaS)

AI-1.7.1	K2	Explain the concept of AI as a Service (AIaaS).

1.8 Pre-Trained Models

AI-1.8.1	K2	Explain the use of pre-trained AI models and the risks associated with them.

1.9 Standards, Regulations and AI

AI-1.9.1	K2	Describe how standards apply to AI-based systems.

1.1 Definition of AI and AI Effect

The term artificial intelligence (AI) dates back to the 1950s and refers to the objective of building and programming “intelligent” machines capable of imitating human beings. The definition today has evolved significantly, and the following definition captures the concept [S01]:

The capability of an engineered system to acquire, process and apply knowledge and skills.

The way in which people understand the meaning of AI depends on their current perception. In the 1970s the idea of a computer system that could beat a human at chess was somewhere in the future and most considered this to be AI. Now, over twenty years after the computer-based system Deep Blue beat world chess champion Garry Kasparov, the “brute force” approach implemented in that system is not considered by many to be true artificial intelligence (i.e., the system did not learn from data and was not capable of self-learning). Similarly, the expert systems of the 1970s and 1980s incorporated human expertise as rules which could be run repeatedly without the expert being present. These were considered to be AI then, but are not considered as such now.

The changing perception of what constitutes AI is known as the “AI Effect” [R01]. As the perception of AI in society changes, so does its definition. As a result, any definition made today is likely to change in the future and may not match those from the past.

1.2 Narrow, General and Super AI

At a high level, AI can be broken into three categories:

- Narrow AI (also known as weak AI) systems are programmed to carry out a specific task with limited context. Currently this form of AI is widely available. For example, game-playing systems, spam filters, test case generators and voice assistants.
- General AI (also known as strong AI) systems have general (wide-ranging) cognitive abilities similar to humans. These AI-based systems can reason and understand their environment as humans do, and act accordingly. As of 2021, no general AI systems have been realized.
- Super AI systems are capable of replicating human cognition (general AI) and make use of massive processing power, practically unlimited memory and access to all human knowledge (e.g., through access to the web). It is thought that super AI systems will quickly become wiser than humans. The point at which AI-based systems transition from general AI to super AI is commonly known as the technological singularity [B01].

1.3 AI-Based and Conventional Systems

In a typical conventional computer system, the software is programmed by humans using an imperative language, which includes constructs such as if-then-else and loops. It is relatively easy for humans to understand how the system transforms inputs into outputs. In an AI-based system using machine learning (ML), patterns in data are used by the system to determine how it should react in the future to new data (see Chapter 3 for a detailed explanation of ML). For example, an AI-based image processor designed to identify images of cats is trained with a set of images known to contain cats. The AI determines on its own what patterns or features in the data can be used to identify cats. These patterns and rules are then applied to new images in order to determine if they contain cats. In many AI-based systems, this results in the prediction-making procedure being less easy to understand by humans (see Section 2.7).

In practice, AI-based systems can be implemented by a variety of technologies (see Section 1.4), and the “AI Effect” (see Section 1.1) may determine what is currently considered to be an AI-based system and what is considered to be a conventional system.

1.4 AI Technologies

AI can be implemented using a wide range of technologies (see [B02] for more details), such as:

- Fuzzy logic
- Search algorithms
- Reasoning techniques
  - Rule engines
  - Deductive classifiers
  - Case-based reasoning
  - Procedural reasoning
- Machine learning techniques
  - Neural networks
  - Bayesian models
  - Decision trees
  - Random forest
  - Linear regression
  - Logistic regression
  - Clustering algorithms
  - Genetic algorithms
  - Support vector machine (SVM)

AI-based systems typically implement one or more of these technologies.

1.5 AI Development Frameworks

There are many AI development frameworks available, some of which are focused on specific domains. These frameworks support a range of activities, such as data preparation, algorithm selection, and compilation of models to run on various processors, such as central processing units (CPUs), graphical processing units (GPUs) or Cloud Tensor Processing Units (TPUs). The selection of a particular framework may also depend on particular aspects such as the programming language used for the implementation and its ease of use. The following frameworks are some of the most popular (as of April 2021):

- Apache MxNet: A deep learning open-source framework used by Amazon for Amazon Web Services (AWS) [R02].
- CNTK: The Microsoft Cognitive Toolkit (CNTK) is an open-source deep-learning toolkit [R03].
- IBM Watson Studio: A suite of tools that support the development of AI solutions [R04].
- Keras: A high-level open-source API, written in the Python language, capable of running on top of TensorFlow and CNTK [R06].
- PyTorch: An open-source ML library operated by Facebook and used for apps applying image processing and natural language processing (NLP). Support is provided for both Python and C++ interfaces [R07].
- Scikit-learn: An open-source machine ML library for the Python programming language [R08].
- TensorFlow: An open-source ML framework based on data flow graphs for scalable machine learning, provided by Google [R05].

Note that these development frameworks are constantly evolving, sometimes combining, and sometimes being replaced by new frameworks.

1.6 Hardware for AI-Based Systems

A variety of hardware is used for ML model training (see Chapter 3) and model implementation. For example, a model that performs speech recognition may run on a low-end smartphone, although access to the power of cloud computing may be needed to train it. A common approach used when the host device is not connected to the internet is to train the model in the cloud and then deploy it to the host device.

ML typically benefits from hardware that supports the following attributes:

- Low-precision arithmetic: This uses fewer bits for computation (e.g., 8 instead of 32 bits, which is usually all that is needed for ML).
- The ability to work with large data structures (e.g., to support matrix multiplication).
- Massively parallel (concurrent) processing.

General-purpose CPUs provide support for complex operations that are not typically required for ML applications and only provide a few cores. As a result, their architecture is less efficient for training and running ML models when compared to GPUs, which have thousands of cores and which are designed to perform the massively parallel but relatively simple processing of images. As a consequence, GPUs typically outperform CPUs for ML applications, even though CPUs typically have faster clock speeds. For small-scale ML work, GPUs generally offer the best option.

Some hardware is specially intended for AI, such as purpose-built Application-Specific Integrated Circuits (ASICs) and Systems on a Chip (SoC). These AI-specific solutions have features such as multiple cores, special data management and the ability to perform in-memory processing. They are most suitable for edge computing, while the training of the ML model is done in the cloud.

Hardware with specific AI architectures is currently (as of April 2021) under development. This includes neuromorphic processors [B03], which do not use the traditional von Neumann architecture, but rather an architecture that loosely mimics brain neurons.

Examples of AI hardware providers and their processors include (as of April 2021):

- NVIDIA: They provide a range of GPUs and AI-specific processors, such as the Volta [R09].
- Google: They have developed application-specific integrated circuits for both training and inferencing. Google TPUs (Cloud Tensor Processing Units) [R10] can be accessed by users on the Google Cloud, whereas the Edge TPU [R11] is a purpose-built ASIC designed to run AI on individual devices.
- Intel: They provide Nervana neural network processors [R12] for deep learning (both training and inferencing) and Movidius Myriad vision processing units for inferencing in computer vision and neural network applications.
- Mobileye: They produce the EyeQ family of SoC devices [R13] to support complex and computationally intense vision processing. These have low power consumption for use in vehicles.
- Apple: They produce the Bionic chip for on-device AI in iPhones [B04].
- Huawei: Their Kirin 970 chip for smartphones has built-in neural network processing for AI [B05].

1.7 AI as a Service (AIaaS)

AI components, such as ML models, can be created within an organization, downloaded from a third-party, or used as a service on the web (AIaaS). A hybrid approach is also possible in which some of the AI functionality is provided from within the system and some is provided as a service.

When ML is used as a service, accesses is provided to an ML model over the web and support can also be provided for data preparation and storage, model training, evaluation, tuning, testing and deployment.

Third-party providers (e.g., AWS, Microsoft) offer specific AI services, such as facial and speech recognition. This allows individuals and organizations to implement AI using cloud-based services even when they have insufficient resources and expertise to build their own AI services. In addition, ML models provided as part of a third-party service are likely to have been trained on a larger, more diverse training dataset than is readily available to many stakeholders, such as those who have recently moved into the AI market.

1.7.1 Contracts for AI as a Service

These AI services are typically provided with similar contracts as for non-AI cloud-based Software as a Service (SaaS). A contract for AIaaS typically includes a service-level agreement (SLA) that defines availability and security commitments. Such SLAs typically cover an uptime for the service (e.g., 99.99% uptime) and a response time to fix defects, but rarely define ML functional performance metrics, (such as accuracy), in a similar manner (see Chapter 5). AIaaS is often paid for on a subscription basis, and if the contracted availability and/or response times are not met, then the service provider typically provides credits for future services. Other than these credits, most AIaaS contracts provide limited liability (other than in terms of fees paid), meaning that AI-based systems that depend on AIaaS are typically limited to relatively low-risk applications, where loss of service would not be too damaging.

Services often come with an initial free trial period in lieu of an acceptance period. During this period the consumer of the AIaaS is expected to test whether the provided service meets their needs in terms of required functionality and performance (e.g., accuracy). This is generally necessary to cover any lack of transparency on the provided service (see Section 7.5).

1.7.2 AIaaS Examples

The following are examples of AIaaS (as of April 2021):

- IBM Watson Assistant: This is an AI chatbot which is priced according to the number of monthly active users.
- Google Cloud AI and ML Products: These provide document-based AI that includes a form parser and document OCR. Prices are based on the number of pages sent for processing.
- Amazon CodeGuru: This provides a review of ML Java code that supplies developers with recommendations for improving their code quality. Prices are based on the number of lines of source code analyzed.
- Microsoft Azure Cognitive Search: The provides AI cloud search. Prices are based on search units (defined in terms of the storage and throughput used).

1.8 Pre-Trained Models

1.8.1 Introduction to Pre-Trained Models

It can be expensive to train ML models (see Chapter 3). First, the data has to be prepared and then the model must be trained. The first activity can consume large amounts of human resources, while the latter activity can consume a lot of computing resources. Many organizations do not have access to these resources.

A cheaper and often more effective alternative is to use a pre-trained model. This provides similar functionality to the required model and is used as the basis for creating a new model that extends and/or focuses the functionality of the pre-trained model. Such models are only available for a limited number of technologies, such as neural networks and random forests.

If an image classifier is needed, it could be trained using the publicly available ImageNet dataset, which contains over 14 million images classified into over 1000 categories. This reduces the risk of consuming significant resources with no guarantee of success. Alternatively, an existing model could be reused that has already been trained on this dataset. By using such a pre-trained model, training costs are saved and the risk of it not working largely eliminated.

When a pre-trained model is used without modification, it can simply be embedded in the AI-based system, or it can be used as a service (see Section 1.7).

1.8.2 Transfer Learning

It is also possible to take a pre-trained model and modify it to perform a second, different requirement. This is known as transfer learning and is used on deep neural networks in which the early layers (see Chapter 6) of the neural network typically perform quite basic tasks (e.g., identifying the difference between straight and curved lines in an image classifier), whereas the later layers perform more specialized tasks (e.g., differentiating between building architectural types). In this example, all but the later layers of an image classifier can be reused, eliminating the need to train the early layers. The later layers are then retrained to handle the unique requirements for a new classifier. In practice, the pre-trained model may be fine-tuned with additional training on new problem-specific data.

The effectiveness of this approach largely depends on the similarity between the function performed by the original model and the function required by the new model. For example, modifying an image classifier that identifies cat species to then identify dog breeds would be far more effective than modifying it to identify people's accents.

There are many pre-trained models available, especially from academic researchers. Some examples of such pre-trained models are ImageNet models [R14] such as Inception, VGG, AlexNet, and MobileNet for image classification and pre-trained NLP models like Google's BERT [R15].

1.8.3 Risks of using Pre-Trained Models and Transfer Learning

Using pre-trained models and transfer learning are both common approaches to building AI-based systems, but there are some risks associated. These include:

- A pre-trained model may lack transparency compared to an internally generated model.
- The level of similarity between the function performed by the pre-trained model and the required functionality may be insufficient. Also, this difference may not be understood by the data scientist.
- Differences in the data preparation steps (see Section 4.1) used for the pre-trained model when originally developed and the data preparation steps used when this model is then used in a new system may impact the resulting functional performance. The shortcomings of a pre-trained model are likely to be inherited by those who reuse it and may not be documented. For example, inherited biases (see Section 2.4) may not be apparent if there is a lack of documentation about the data used to train the model. Also, if the pre-trained model is not widely used, there are likely to be more unknown (or undocumented) defects and more rigorous testing may be needed to mitigate this risk.
- Models created through transfer learning are highly likely to be sensitive to the same vulnerabilities as the pre-trained model on which it is based (e.g., adversarial attacks, as explained in 9.1.1). In addition, if an AI-based system is known to contain a specific pre-trained model (or is based on a specific pre-trained model), then vulnerabilities associated with it may already be known by potential attackers.

Note that several of the above risks can be more easily mitigated by having thorough documentation available for the pre-trained model (see Section 7.5).

1.9 Standards, Regulations and AI

The Joint Technical Committee of IEC and ISO on information technology (ISO/IEC JTC1) prepares international standards which contribute towards AI. For example, a subcommittee on AI (ISO/IEC JTC 1/SC42), was set up in 2017. In addition, ISO/IEC JTC1/SC7, which covers software and system engineering, has published a technical report on the "Testing of AI-based systems" [S01].

Standards on AI are also published at the regional level (e.g., European standards) and the national level.

The EU-wide General Data Protection Regulation (GDPR) came into effect in May 2018 and sets obligations for data controllers with regards to personal data and automated decision-making [B06]. It includes requirements to assess and improve AI system functional performance, including the mitigation of potential discrimination, and for ensuring individuals' rights to not be subjected to automated decision-making. The most important aspect of the GDPR from a testing perspective is that personal data (including predictions) should be accurate. This does not mean that every single prediction made by the system must be accurate, but that the system should be accurate enough for the purposes for which it is used.

The German national standards body (DIN) has also developed the AI Quality Metamodel ([S02], [S03]).

Standards on AI are also published by industry bodies. For example, the Institute of Electrical and Electronics Engineers (IEEE) is working on a range of standards on ethics and AI (The IEEE Global Initiative for Ethical Considerations in Artificial Intelligence and Autonomous Systems). Many of these standards are still in development at the time of writing.

Where AI is used in safety-related systems, the relevant regulatory standards are applicable, such as ISO 26262 [S04] and ISO/PAS 21448 (SOTIF) [S05] for automotive systems. Such regulatory standards are typically mandated by government bodies, and it would be illegal to sell a car in some countries if the included software did not comply with ISO 26262. Standards in isolation are voluntary documents, and their use is normally only made mandatory by legislation or contract. However, many users of standards do so to benefit from the expertise of the authors and to create products that are of higher quality.

2 Quality Characteristics for AI-Based Systems – 105 minutes

Keywords

None

AI-Specific Keywords

Adaptability, algorithmic bias, autonomy, bias, evolution, explainability, explainable AI (XAI), flexibility, inappropriate bias, interpretability, ML system, machine learning, reward hacking, robustness, sample bias, self-learning system, side effects, transparency

Learning Objectives for Chapter 2:

2.1 Flexibility and Adaptability

AI-2.1.1	K2	Explain the importance of flexibility and adaptability as characteristics of AI-based systems.

2.2 Autonomy

AI-2.2.1	K2	Explain the relationship between autonomy and AI-based systems.

2.3 Evolution

AI-2.3.1	K2	Explain the importance of managing evolution for AI-based systems.

2.4 Bias

AI-2.4.1	K2	Describe the different causes and types of bias found in AI-based systems.

2.5 Ethics

AI-2.5.1	K2	Discuss the ethical principles that should be respected in the development, deployment and use of AI-based systems.

2.6 Side Effects and Reward Hacking

AI-2.6.1	K2	Explain the occurrence of side effects and reward hacking in AI-based systems.

2.7 Transparency, Interpretability and Explainability

AI-2.7.1	K2	Explain how transparency, interpretability and explainability apply to AI-based systems.

2.8 Safety and AI

AI-2.8.1	K1	Recall the characteristics that make it difficult to use AI-based systems in safety-related applications.

2.1 Flexibility and Adaptability

Flexibility and adaptability are closely related quality characteristics. In this syllabus, flexibility is considered to be the ability of the system to be used in situations that were not part of the original system requirements, while adaptability is considered to be the ease with which the system can be modified for new situations, such as different hardware and changing operational environments.

Both flexibility and adaptability are useful if:

- the operational environment is not fully known when the system is deployed.
- the system is expected to cope with new operational environments.
- the system is expected to adapt to new situations.
- the system must determine when it should change its behavior.

Self-learning AI-based systems are expected to demonstrate all of the above characteristics. As a consequence, they must be adaptable and have the potential to be flexible.

The flexibility and adaptability requirements of an AI-based system should include details of any environment changes to which the system is expected to adapt. These requirements should also specify constraints on the time and resources that the system can use to adapt itself (e.g., how long can it take to adapt to recognizing a new type of object).

2.2 Autonomy

When defining autonomy, it is important to first recognize that a fully autonomous system would be completely independent of human oversight and control. In practice, full autonomy is not often desired. For example, fully self-driving cars, which are popularly referred to as “autonomous”, are officially classified as having “full driving automation” [B07].

Many consider autonomous systems to be “smart” or “intelligent”, which suggests they would include AI-based components to perform certain functions. For example, autonomous vehicles that need to be situationally aware typically use several sensors and image processing to gather information about the vehicle’s immediate environment. Machine learning, and especially deep learning (see Section 6.1), has been found to be the most effective approach to performing this function. Autonomous systems may also include decision-making and control functions. Both of these can be effectively performed using AI-based components.

Even though some AI-based systems are considered to be autonomous, this does not apply to all AI-based systems. In this syllabus, autonomy is considered to be the ability of the system to work independently of human oversight and control for prolonged periods of time. This can help with identifying the characteristics of an autonomous system that need to be specified and tested. For example, the length of time an autonomous system is expected to perform satisfactorily without human intervention needs to be known. In addition, it is important to identify the events for which the autonomous system must give control back to its human controllers.

2.3 Evolution

In this syllabus, evolution is considered to be the ability of the system to improve itself in response to changing external constraints. Some AI systems can be described as self-learning and successful self-learning AI-based systems need to incorporate this form of evolution.

AI-based systems often operate in an evolving environment. As with other forms of IT systems, an AI-based system needs to be flexible and adaptable enough to cope with changes in its operational environment.

Self-learning AI-based systems typically need to manage two forms of change:

- One form of change is where the system learns from its own decisions and its interactions with its environment.
- The other form of change is where the system learns from changes made to the system's operational environment.

In both cases the system will ideally evolve to improve its effectiveness and efficiency. However, this evolution must be constrained to prevent the system from developing any unwanted characteristics. Any evolution must continue to meet the original system requirements and constraints. Where these are lacking, the system must be managed to ensure that any evolution remains within limits and that it always stays aligned with human values. Section 2.6 provides examples relating to the impact of side effects and reward hacking on self-learning AI-based systems.

2.4 Bias

In the context of AI-based systems, bias is a statistical measure of the distance between the outputs provided by the system and what is considered to be "fair outputs" which show no favoritism to a particular group. Inappropriate biases can be linked to attributes such as gender, race, ethnicity, sexual orientation, income level, and age. Cases of inappropriate bias in AI-based systems have been reported, for example, in systems used for making recommendations for bank lending, in recruitment systems, and in judicial monitoring systems.

Bias can be introduced into many types of AI-based systems. For example, it is difficult to prevent the bias of experts being built-in to the rules applied by an expert system. However, the prevalence of ML systems means that much of the discussion relating to bias takes place in the context of these systems.

ML systems are used to make decisions and predictions, using algorithms which make use of collected data, and these two components can introduce bias in the results:

- Algorithmic bias can occur when the learning algorithm is incorrectly configured, for example, when it overvalues some data compared to others. This source of bias can be caused and managed by the hyperparameter tuning of the ML algorithms (see Section 3.2).
- Sample bias can occur when the training data is not fully representative of the data space to which ML is applied.

Inappropriate bias is often caused by sample bias, but occasionally it can also be caused by algorithmic bias.

2.5 Ethics

Ethics is defined in the Cambridge Dictionary as:

a system of accepted beliefs that control behavior, especially such a system based on morals

AI-based systems with enhanced capabilities are having a largely positive effect on people's lives. As these systems have become more widespread, concerns have been raised as to whether they are used in an ethical manner.

What is considered ethical can change over time and can also change among localities and cultures. Care must be taken that the deployment of an AI-based system from one location to another considers differences in stakeholder values.

National and international policies on the ethics of AI can be found in many countries and regions. The Organisation for Economic Co-operation and Development issued its principles for AI, the first international standards agreed by governments for the responsible development of AI, in 2019 [B08]. These principles were adopted by forty-two countries when they were issued and are also backed by the European Commission. They include practical policy recommendations as well as value-based principles for the "responsible stewardship of trustworthy AI". These are summarized as:

- AI should benefit people and the planet by driving inclusive growth, sustainable development and well-being.
- AI systems should respect the rule of law, human rights, democratic values and diversity, and should include appropriate safeguards to ensure a fair society.
- There should be transparency around AI to ensure that people understand outcomes and can challenge them.
- AI systems must function in a robust, secure and safe way throughout their life cycles and risks should be continually assessed.
- Organizations and individuals developing, deploying or operating AI systems should be held accountable.

2.6 Side Effects and Reward Hacking

Side effects and reward hacking can result in AI-based systems generating unexpected, and even harmful, results when the system attempts to meet its goals [B09].

Negative side effects can result when the designer of an AI-based system specifies a goal that "focuses on accomplishing some specific tasks in the environment but ignores other aspects of the (potentially very large) environment, and thus implicitly expresses indifference over environmental variables that might actually be harmful to change" [B09]. For example, a self-driving car with a goal of travelling to its destination in "as fuel-efficient and safe manner as possible" may achieve the goal, but with the side effect of the passengers becoming extremely annoyed at the excessive time taken.

Reward hacking can result from an AI-based system achieving a specified goal by using a "clever" or "easy" solution that "perverts the spirit of the designer's intent". Effectively, the goal can be gamed. A widely used example of reward hacking is where an AI-based system is teaching itself to play an arcade computer game. It is presented with the goal of achieving the "highest score" , and to do so it simply hacks the data record that stores the highest score, rather than playing the game to achieve it.

2.7 Transparency, Interpretability and Explainability

AI-based systems are typically applied in areas where users need to trust those systems. This may be for safety reasons, but also where privacy is needed and where they might provide potentially life-changing predictions and decisions.

Most users are presented with AI-based systems as "black boxes" and have little awareness of how these systems arrive at their results. In some cases, this ignorance may even apply to the data scientists who built the systems. Occasionally, users may not even be aware they are interacting with an AI-based system.

The inherent complexity of AI-based systems has led to the field of "Explainable AI" (XAI). The aim of XAI is for users to be able to understand how AI-based systems come up with their results, thus increasing users' trust in them.

According to The Royal Society [B10], there are several reasons for wanting XAI, including:

- giving users confidence in the system
- safeguarding against bias
- meeting regulatory standards or policy requirements
- improving system design
- assessing risk, robustness, and vulnerability
- understanding and verifying the outputs from a system
- autonomy, agency (making the user feel empowered), and meeting social values

This leads to the following three basic desirable XAI characteristics for AI-based systems from the perspective of a stakeholder (see also Section 8.6):

- Transparency: This is considered to be the ease with which the algorithm and training data used to generate the model can be determined.
- Interpretability: This is considered to be the understandability of the AI technology by various stakeholders, including the users.
- Explainability: This is considered to be the ease with which users can determine how the AI-based system comes up with a particular result.

2.8 Safety and AI

In this syllabus, safety is considered to be the expectancy that an AI-based system will not cause harm to people, property or the environment. AI-based systems may be used to make decisions that affect safety. For example, AI-based systems working in the fields of medicine, manufacturing, defense, security, and transportation have the potential to affect safety.

The characteristics of AI-based systems that make it more difficult to ensure they are safe (e.g., do not harm humans) include:

- complexity
- non-determinism
- probabilistic nature
- self-learning
- lack of transparency, interpretability and explainability
- lack of robustness

The challenges of testing several of these characteristics are covered in Chapter 8.

3 Machine Learning (ML) – Overview - 145 minutes

Keywords

None

AI-Specific Keywords

Association, classification, clustering, data preparation, ML algorithm, ML framework, ML functional performance criteria, ML model, ML training data, ML workflow, model evaluation, model tuning, outlier, overfitting, regression, reinforcement learning, supervised learning, underfitting, unsupervised learning

Learning Objectives for Chapter 3:

3.1 Forms of ML

AI-3.1.1	K2	Describe classification and regression as part of supervised learning.
AI-3.1.2	K2	Describe clustering and association as part of unsupervised learning.
AI-3.1.3	K2	Describe reinforcement learning.

3.2 ML Workflow

AI-3.2.1	K2	Summarize the workflow used to create an ML system.

3.3 Selecting a Form of ML

AI-3.3.1	K3	Given a project scenario, identify an appropriate form of ML (from classification, regression, clustering, association, or reinforcement learning).

3.4 Factors involved in ML Algorithm Selection

AI-3.4.1	K2	Explain the factors involved in the selection of ML algorithms.

3.5 Overfitting and Underfitting

AI-3.5.1	K2	Summarize the concepts of underfitting and overfitting.
HO-3.5.1	H0	Demonstrate underfitting and overfitting.

3.1 Forms of ML

ML algorithms can be categorized as:

- supervised learning,
- unsupervised learning, and
- reinforcement learning.

3.1.1 Supervised Learning

In this kind of learning, the algorithm creates the ML model from labeled data during the training phase. The labeled data, which typically comprises pairs of inputs (e.g., an image of a dog and the label “dog”) is used by the algorithm to infer the relationship between the input data (e.g., images of dogs) and the output labels (e.g., “dog” and “cat”) during the training. During the ML model testing phase, a new set of unseen data is applied to the trained model to predict the output. The model is deployed once the output accuracy level is satisfactory.

Problems solved by supervised learning are divided into two categories:

- Classification: This is when the problem requires an input to be classified into one of a few pre-defined classes, classification is used. Face recognition or object detection in an image are examples of problems that use classification.
- Regression: This is when the problem requires the ML model to predict a numeric output using regression. Predicting the age of a person based on input data about their habits or predicting the future prices of stocks are examples of problems that use regression.

Note that the term regression, as used in the context of a ML problem, is different to its use in other ISTQB® syllabi, such as [I01], where regression is used to describe the problem of software modifications causing change-related defects.

3.1.2 Unsupervised Learning

In this kind of learning, the algorithm creates the ML model from unlabeled data during the training phase. The unlabeled data is used by the algorithm to infer patterns in the input data during the training and assigns inputs to different classes, based on their commonalities. During the testing phase, the trained model is applied to a new set of unseen data to predict which classes the input data should be assigned to. The model is deployed once the output accuracy level is considered to be satisfactory.

Problems solved by unsupervised learning are divided into two categories:

- Clustering: This is when the problem requires the identification of similarities in input data points that allows them to be grouped based on common characteristics or attributes. For example, clustering is used to categorize different types of customers for the purpose of marketing.
- Association: This is when the problem requires interesting relationships or dependencies to be identified among data attributes. For example, a product recommendation system may identify associations based on customers’ shopping behavior.

3.1.3 Reinforcement Learning

Reinforcement learning is an approach where the system (an “intelligent agent”) learns by interacting with the environment in an iterative manner and thereby learns from experience. Reinforcement learning does not use training data. The agent is rewarded when it makes a correct decision and penalized when it makes an incorrect decision.

Setting up the environment, choosing the right strategy for the agent to meet the desired goal, and designing a reward function, are key challenges when implementing reinforcement learning. Robotics, autonomous vehicles, and chatbots are examples of applications that use reinforcement learning.

3.2 ML Workflow

The activities in the machine learning workflow are:

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 ML functional performance metrics – see Chapter 5) should be defined for the developed model.

Select a Framework

A suitable AI development framework should be selected based on the objectives, acceptance criteria, and business priorities (see Section 1.5).

Select & Build the Algorithm

An ML algorithm is selected based on various factors including the objectives, acceptance criteria, and the available data (see Section 3.4). The algorithm may be manually coded, but it is often retrieved from a library of pre-written code. The algorithm is then compiled to prepare for training the model, if required.

Prepare & Test Data

Data preparation (see Section 4.1) comprises data acquisition, data pre-processing and feature engineering. Exploratory data analysis (EDA) may be performed alongside these activities.

The data used by the algorithm and model will be based on the objectives and is used by all the activities in the “model generation and test” activity shown on Figure 1. For example, if the system is a real-time trading system, the data will come from the trading market.

The data used to train, tune and test the model must be representative of the operational data that will be used by the model. In some cases, it is possible to use pre-gathered datasets for the initial training of the model (e.g., see Kaggle datasets [R16]). Otherwise, raw data typically needs some pre-processing and feature engineering.

Testing of the data and any automated data preparation steps needs to be performed. See Section 7.2.1 for more details on input data testing.

Train the Model

The selected ML algorithm uses training data to train the model.

Some algorithms, such as those generating a neural network, read the training dataset several times. Each iteration of training on the training dataset is referred to as an epoch.

Parameters defining the model structure (e.g., the number of layers of a neural network or the depth of a decision tree) are passed to the algorithm. These parameters are known as model hyperparameters.

Parameters that control the training (e.g., how many epochs to use when training a neural network) are also passed to the algorithm. These parameters are known as algorithm hyperparameters.

Evaluate the Model

The model is evaluated against the agreed ML functional performance metrics, using the validation dataset and the results then used to improve (tune) the model. Model evaluation and tuning should resemble a scientific experiment that needs to be carefully conducted under controlled conditions with clear documentation. In practice, several models are typically created and trained using different algorithms (e.g., random forests, SVM, and neural networks), and the best one is chosen, based on the results of the evaluation and tuning.

Tune the Model

The results from evaluating the model against the agreed ML functional performance metrics are used to adjust the model settings to fit the data and thereby improve its 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 updated (e.g., the number of neurons in a neural network or the depth of a decision tree).

The three activities of training, evaluation and tuning can be considered as comprising model generation, as shown on Figure 1.

Test the Model

Once a model has been generated, (i.e., it has been trained, evaluated and tuned), it should be tested against an independent test dataset set to ensure that the agreed ML functional performance criteria are met (see Section 7.2.2). The functional performance measures from testing are also compared with those from evaluation, and if the performance of the model with independent data is significantly lower than during evaluation, it may be necessary to select a different model.

In addition to functional performance tests, non-functional tests, such as for the time to train the model, and the time and resource usage taken to provide a prediction, also need to be performed. Typically, these tests are performed by the data engineer/scientist, but testers with sufficient knowledge of the domain and access to the relevant resources can also perform these tests.

Deploy the Model

Once model development is complete, as shown on Figure 1, 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 framework. Targets might include embedded systems and the cloud, where the model can be accessed via a web API.

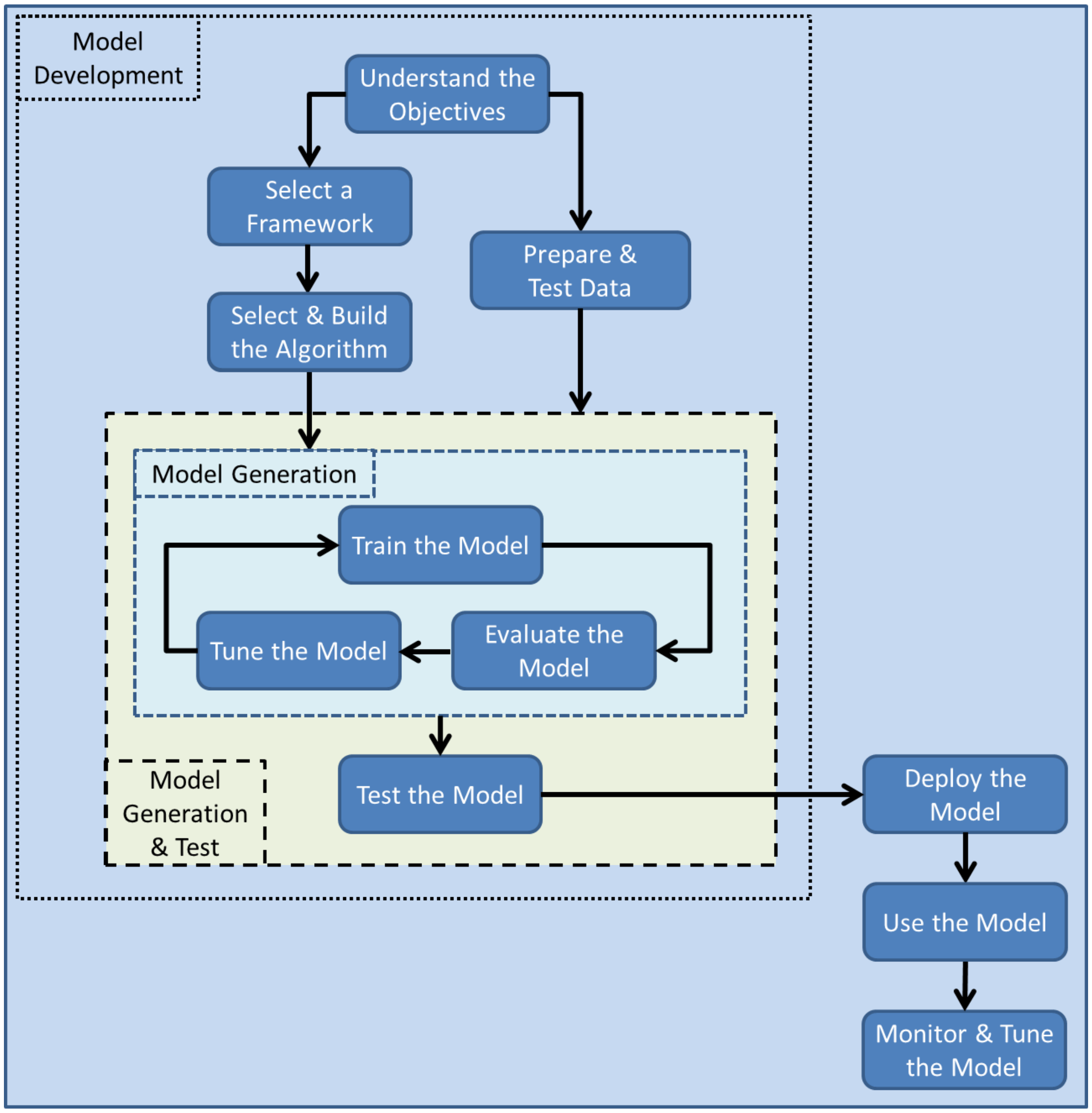

Figure 1: ML Workflow

Use the Model

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

Monitor and Tune the Model

While the model is being used, its situation may evolve and the model may drift away from its intended performance (see Sections 2.3 and 7.6). 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 is needed to create a more accurate or more robust model. In this case a new model may be created and trained with updated training data. The new model may then be compared against the existing model using a form of A/B testing (see Section 9.4).

The ML workflow shown in Figure 1 is a logical sequence. In practice, the workflow is applied in a manner where the steps are repeated iteratively (e.g., when the model is evaluated, it is often necessary to return to the training step, and sometimes to data preparation).

The steps shown in Figure 1 do not include the integration of the ML model with the non-ML parts of the overall system. Typically, ML models cannot be deployed in isolation and need to be integrated with the non-ML parts. For example, in vision applications, there is a data pipeline that cleans and modifies data before submitting it to the ML model. Where the model is part of a larger AI-based system, it will need to be integrated into this system prior to deployment. In this case, integration, system and acceptance test levels may be performed, as described in Section 7.2.

3.3 Selecting a Form of ML

When selecting an appropriate ML approach, the following guidelines apply:

- There should be sufficient training and test data available for the selected ML approach.
- For supervised learning, it is necessary to have properly labeled data.
- If there is an output label, it may be supervised learning.
- If the output is discrete and categorical, it may be classification.
- If the output is numeric and continuous in nature, it may be regression.
- If no output is provided in the given dataset, it may be unsupervised learning.
- If the problem involves grouping similar data, it may be clustering.
- If the problem involves finding co-occurring data items, it may be association.
- Reinforcement learning is better suited to contexts in which there is interaction with the environment.
- If the problem involves the notion of multiple states, and involves decisions at each state, then reinforcement learning may be applicable.

3.4 Factors Involved in ML Algorithm Selection

There is no definitive approach to selecting the optimal ML algorithm, ML model settings and ML model hyperparameters. In practice, this set is chosen based on a mix of the following factors:

- The required functionality (e.g., whether the functionality is classification or prediction of a discrete value)

- The required quality characteristics; such as
    - accuracy (e.g., some models may be more accurate, but be slower)
    - constraints on available memory (e.g., for an embedded system)
    - the speed of training (and retraining) the model
    - the speed of prediction (e.g., for real-time systems)
    - transparency, interpretability and explainability requirements
- The type of data available for training the model (e.g., some models might only work with image data)
- The amount of data available for training and testing the model (some models might, for example, have a tendency to overfit with a limited amount of data, to a greater degree than other models)
- The number of features in the input data expected to be used by the model (e.g., other factors, such as speed and accuracy, are likely to be directly affected by the number of features)
- The expected number of classes for clustering (e.g., some models may be unsuitable for problems with more than one class)
- Previous experience
- Trial and error

3.5 Overfitting and Underfitting

3.5.1 Overfitting

Overfitting occurs when the model fits too closely to a set of data points and fails to properly generalize. Such a model works very well with the data used to train it but can struggle to provide accurate predictions for new data. Overfitting can occur when the model tries to fit to every data point, including those data points that may be described as noise or outliers. It can also occur when insufficient data is provided in the training dataset.

3.5.2 Underfitting

Underfitting occurs when the model is not sophisticated enough to accurately fit to the patterns in the training data. Underfitting models tend to be too simplistic and can struggle to provide accurate predictions for both new data and data very similar to the training data. One cause of underfitting can be a training dataset that does not contain features that reflect important relationships between inputs and outputs. It can also occur when the algorithm does not correctly fit the data (e.g., creating a linear model for non-linear data).

3.5.3 Hands-On Exercise: Demonstrate Overfitting and Underfitting

Demonstrate the concepts of overfitting and underfitting on a model. This could be demonstrated by using a dataset that contains very little data (overfitting), and a dataset with poor feature correlations (underfitting).

4 ML - Data – 230 minutes

Keywords

None

AI-Specific Keywords

Annotation, augmentation, classification model, data labelling, data preparation, ML training data, supervised learning, test dataset, validation dataset

Learning Objectives for Chapter 4:

4.1 Data Preparation as part of the ML Workflow

AI-4.1.1	K2	Describe the activities and challenges related to data preparation.
HO-4.1.1	H2	Perform data preparation in support of the creation of an ML model.

4.2 Training, Validation and Test Datasets in the ML Workflow

AI-4.2.1	K2	Contrast the use of training, validation and test datasets in the development of an ML model.
HO-4.2.1	H2	Identify training and test datasets and create an ML model.

4.3 Dataset Quality Issues

AI-4.3.1	K2	Describe typical dataset quality issues.

4.4 Data quality and its effect on the ML model

AI-4.4.1	K2	Recognize how poor data quality can cause problems with the resultant ML model.

4.5 Data Labelling for Supervised Learning

AI-4.5.1	K1	Recall the different approaches to the labelling of data in datasets for supervised learning.
AI-4.5.2	K1	Recall reasons for the data in datasets being mislabeled.

4.1 Data Preparation as Part of the ML Workflow

Data preparation uses an average of 36% of the ML workflow effort and is probably the most resource-intensive activity in the ML workflow. In comparison, model selection and building uses only 14% [R17]. Data preparation forms part of the data pipeline, which takes in raw data and outputs data in a form that can be used to both train an ML model and for prediction by a trained ML model.

Data preparation can be considered to comprise the following activities:

Data acquisition

- Identification: The types of data to be used for training and predictions are identified. For example, for a self-driving car, it could include the identification of the need for radar, video and laser imaging, detection, and ranging (LiDAR) data.
- Gathering: The source of the data is identified and the means for collecting the data are determined. For example, this could include the identification of the International Monetary Fund (IMF) as a source for financial data and the channels that will be used to submit the data into the AI-based system.
- Labelling: See Section 4.5.

The acquired data can be in various forms (e.g., numerical, categorical, image, tabular, text, time-series, sensor, geospatial, video, and audio).

Data pre-processing

- Cleaning: Where incorrect data, duplicate data or outliers are identified, they are either removed or corrected. In addition, data imputation may be used to replace missing data values with estimated or guessed values (e.g., using mean, median and mode values). The removal or anonymization of personal information may also be performed.
- Transformation: The format of the given data is changed (e.g., breaking an address held as a string into its constituent parts, dropping a field holding a random identifier, converting categorical data into numerical data, changing image formats). Some of the transformations applied on numerical data include scaling to ensure that the same range is used. Standardization, for example, rescales data to ensure it takes a mean of zero and a standard deviation of one. This normalization ensures that the data has a range between zero and one.
- Augmentation: This is used to increase the number of samples in a dataset. Augmentation can also be used to include adversarial examples in the training data, providing robustness against adversarial attacks (see 9.1).
- Sampling: This involves selection of some part of the total available dataset so that patterns in the larger dataset can be observed. This is typically done to reduce costs and the time needed to create the ML model.

Note that all pre-processing carries a risk that it may change useful valid data or add invalid data.

Feature engineering

- Feature selection: A feature is an attribute/property reflected in the data. Feature selection involves the selection of those features which are most likely to contribute to model training and prediction. In practice, it often includes the removal features that are not expected (or that are not wanted) to have any effect on the resultant model. By removing irrelevant information (noise), feature selection can reduce overall training times, prevent overfitting (see Section 3.5.1), increase accuracy and make models more generalizable.

- Feature extraction: This involves the derivation of informative and non-redundant features from the existing features. The resulting data set is typically smaller and can be used to generate an ML model of equivalent accuracy more cheaply and more quickly.

In parallel to these data preparation activities, exploratory data analysis (EDA) is also typically carried out to support the overall data preparation task. This includes performing data analysis to discover trends inherent in the data and using data visualization to represent data in a visual format by plotting trends in the data.

Although the above data preparation activities and sub-activities have been shown in a logical order, different projects may re-order them or only use a subset of them. Some of the data preparation steps, such as the identification of the data source, are performed just once and can be performed manually. Other steps may be part of the operational data pipeline and normally work on live data. These tasks should be automated.

4.1.1 Challenges in Data Preparation

Some of the challenges related to data preparation include:

- The need for knowledge of:
  - the application domain.
  - the data and its properties.
  - the various techniques associated with data preparation.
- The difficulty of getting high quality data from multiple sources.
- The difficulty of automating the data pipeline, and ensuring that the production data pipeline is both scalable and has reasonable performance efficiency (e.g., time needed to complete the processing of a data item).
- The costs associated with data preparation.
- Not giving sufficient priority to checking for defects introduced into the data pipeline during data preparation.
- The introduction of data bias (see Section 2.4).

4.1.2 Hands-On Exercise: Data Preparation for ML

For a given set of raw data, perform the applicable data preparation steps as outlined in Section 4.1 to produce a dataset that will be used to create a classification model using supervised learning.

This activity forms the first step in creating an ML model that will be used for future exercises.

To perform this activity, students will be provided with appropriate (and language-specific) materials, including:

- Libraries

- ML frameworks
- Tools
- A development environment

4.2 Training, Validation and Test Datasets in the ML Workflow

Logically, three sets of equivalent data (i.e., randomly selected from a single initial dataset) are required to develop an ML model:

- A training dataset, which is used to train the model.
- A validation dataset, which used for evaluating and subsequently tuning the model.
- A test dataset, (also known as the holdout dataset), which is used for testing the tuned model.

If unlimited suitable data is available, the amount of data used in the ML workflow for training, evaluation and testing typically depends on the following factors:

- The algorithm used to train the model.
- The availability of resources, such as RAM, disk space, computing power, network bandwidth and the available time.

In practice, due to the challenge of acquiring sufficient suitable data, the training and validation datasets are often derived from a single combined dataset. The test dataset is kept separate and is not used during training. This is to ensure the developed model is not influenced by the test data, and so that test results give a true reflection of the model’s quality

There is no optimal ratio for splitting the combined dataset into the three individual datasets, but the typical ratios which may be used as a guideline range from 60:20:20 to 80:10:10 (training: validation: test). Splitting the data into these datasets it is often done randomly, unless the dataset is small or if there is a risk of the resultant datasets not being representative of the expected operational data.

If limited data is available, then splitting the available data into three datasets may result in insufficient data being available for effective training. To overcome this issue, the training and validation datasets may be combined (keeping the test dataset separate), and then used to create multiple split combinations of this dataset (e.g., 80% training / 20% validation). Data is then randomly assigned to the training and validation datasets. Training, validation and tuning are performed using these multiple split combinations to create multiple tuned models, and the overall model performance may be calculated as the average across all runs. There are various methods used for creating multiple split combinations, which include split-test, bootstrap, K-fold cross validation and leave-one-out cross validation (see [B02] for more details).

4.2.1 Hands-On Exercise: Identify Training and Test Data and Create an ML Model

Split the previously prepared data (see Section 4.1.2) into training, validation and test datasets.

Train and test a classification model using supervised learning with these datasets.

Explain the difference between evaluating/tuning and testing by comparing the accuracy achieved with the validation and test datasets.

4.3 Dataset Quality Issues

Typical quality issues relating to the data in a dataset include, but are not limited to, those shown in the following table:

Quality Aspect	**Description**
Wrong data	The captured data was incorrect (e.g., through a faulty sensor) or entered incorrectly (e.g., copy-paste errors).
Incomplete data	Data values may be missing (e.g., a field in a record may be empty, or the data for a particular time interval may have been omitted). There can be various reasons for incomplete data, including security issues, hardware issues, and human error.
Mislabeled data	There are several possible reasons for data to be mislabeled (see Section 4.5.2).
Insufficient data	Insufficient data is available for patterns to be recognized by the learning algorithms in use (note that the minimum required quantity of data will vary for different algorithms).
Data not pre-processed	Data should be pre-processed to ensure it is clean, in a consistent format and contains no unwanted outliers (see Section 4.1).
Obsolete data	Data used for both learning and prediction should be current as possible (e.g., using financial data from several years ago may well lead to inaccurate results).
Unbalanced data	Unbalanced data may result from inappropriate bias (e.g., based on race, gender, or ethnicity), poor placement of sensors (e.g., facial recognition cameras placed at ceiling height), variability in the availability of datasets, and differing motivations of data suppliers.
Unfair data	Fairness is a subjective quality characteristic but can often be identified. For example, to support diversity or gender balancing, selected data may be positively biased towards minorities or disadvantaged groups (note that such data may be considered fair but may not be balanced).
Duplicate data	Repeated data records may unduly influence the resultant ML model.
Irrelevant data	Data that is not relevant to the problem being addressed may adversely influence the results and may lead to wasting resources.

Quality Aspect	Description
Privacy issues	Any data use should respect the relevant data privacy laws (e.g., GDPR with relation to individuals' personal information in the European Union).
Security issues	Fraudulent or misleading data that has been deliberately inserted into the training data may lead to inaccuracy in the trained model.

4.4 Data Quality and its Effect on the ML Model

The quality of the ML model is highly dependent on the quality of the dataset from which it is created. Poor quality data can result in both flawed models and flawed predictions.

The following categories of defects result from data quality issues:

- Reduced accuracy: These defects are caused by data which is wrong, incomplete, mislabeled, insufficient, obsolete, irrelevant, and data which is not pre-processed. For example, if the data was used to build a model of expected house prices, but the training data contained little or no data on detached houses with conservatories, then the predicted prices for this specific house type would probably be inaccurate.
- Biased model: These defects are caused by data which is incomplete, unbalanced, unfair, lacking diversity, or duplicated. For example, if the data from a particular feature is missing (e.g., all the medical data for disease prediction is gathered from subjects of one particular gender), then this is likely to have an adverse effect on the resultant model (unless the model is only to be used to make predictions for that gender operationally).
- Compromised model: These defects are due to data privacy and security restrictions . For example, privacy issues in the data can lead to security vulnerabilities, which would enable attackers to reverse engineer information from the models and might subsequently cause leakage of personal information.

4.5 Data Labelling for Supervised Learning

Data labelling is the enrichment of unlabeled (or poorly labeled) data by adding labels, so it becomes suitable for use in supervised learning. Data labelling is a resource-intensive activity that has been reported to use, on average, 25% of the time on ML projects [B11].

In its simplest form, data labelling can consist of putting images or text files in various folders, based on their classes. For example, putting all text files of positive product reviews into one folder and all negative reviews into another folder. Labelling objects in images by drawing rectangles around them is another common labelling technique, often known as annotation. More complex annotations could be required for labelling 3D objects or for drawing bounding boxes around irregular objects. Data labelling and annotation are typically supported by tools.

4.5.1 Approaches to Data Labelling

Labelling may be performed in a number of ways:

- Internal: The labelling is performed by developers, testers or a team within the organization which is set up for the labelling.
- Outsourced: The labelling is done by an external specialist organization.
- Crowdsourced: The labelling is performed by a large group of individuals. Due to the difficulty of managing the quality of the labelling, several annotators may be asked to label the same data and a decision then taken on the label to be used.
- AI-Assisted: AI-based tools are used to recognize and annotate data or to cluster similar data. The results are then confirmed or perhaps supplemented (e.g., by modifying the bounding box) by a human, as part of a two-step process.
- Hybrid: A combination of the above labelling approaches could be used. For example, crowdsourced labelling is typically managed by an external organization which has access to specialized crowd-management tools.

Where applicable, it may be possible to reuse a pre-labeled dataset, hence avoiding the need for data labelling altogether. Many such datasets are publicly available, for example, from Kaggle [R16].

4.5.2 Mislabeled Data in Datasets

Supervised learning assumes that the data is correctly labeled by the data annotators. However, it is rare in practice for all items in a dataset to be labeled correctly. Data is mislabeled for the following reasons:

- Random errors may be made by annotators (e.g., pressing the wrong button).
- Systemic errors may be made, (e.g., where the labelers are given the wrong instructions or poor training).
- Deliberate errors may be made by malicious data annotators.
- Translation errors may take correctly labeled data in one language and mislabel it in another.
- Where the choice is open to interpretation, subjective judgements made by data annotators may lead to conflicting data labels from different annotators.
- Lack of required domain knowledge may lead to incorrect labelling.
- Complex classification tasks can result in more errors being made.
- The tools used to support data labelling have defects that lead to incorrect labels.
- ML-based approaches to labelling are probabilistic, and this can lead to some incorrect labels.

5 ML Functional Performance Metrics – 120 minutes

Keywords

None

AI-Specific Keywords

Accuracy, area under curve (AUC), confusion matrix, F1-score, inter-cluster metrics, intra-cluster metrics, mean square error (MSE), ML benchmark suites, ML functional performance metrics, precision, recall, receiver operating characteristic (ROC) curve, regression model, R-squared, silhouette coefficient

Learning Objectives for Chapter 5:

5.1 Confusion Matrix

AI-5.1.1	K3	Calculate the ML functional performance metrics from a given set of confusion matrix data.

5.2 Additional ML Functional Performance Metrics for Classification, Regression and Clustering

AI-5.2.1	K2	Contrast and compare the concepts behind the ML functional performance metrics for classification, regression and clustering methods.

5.3 Limitations of ML Functional Performance Metrics

AI-5.3.1	K2	Summarize the limitations of using ML functional performance metrics to determine the quality of the ML system.

5.4 Selecting ML Functional Performance Metrics

AI-5.4.1	K4	Select appropriate ML functional performance metrics and/or their values for a given ML model and scenario.
HO-5.4.1	H2	Evaluate the created ML model using selected ML functional performance metrics

5.5 Benchmark Suites for ML

AI-5.5.1	K2	Explain the use of benchmark suites in the context of ML

5.1 Confusion Matrix

In a classification problem, a model will rarely predict the results correctly all the time. For any such problem, a confusion matrix can be created with the following possibilities:

		Actual	
		Positive	**Negative**
Predicted	**Positive**	**True Positive (TP)**	**False Positive (FP)**
	Negative	**False Negative (FN)**	**True Negative (TN)**

Figure 2: Confusion Matrix

Note that the confusion matrix shown in Figure 2 may be presented differently but will always generate values for the four possible situations of true positive (TP), true negative (TN), false positive (FP) and false negative (FN).

Based on the confusion matrix, the following metrics are defined:

- Accuracy

  Accuracy = (TP + TN) / (TP +TN + FP + FN) * 100%

  Accuracy measures the percentage of all correct classifications.

- Precision

  Precision = TP / (TP + FP) * 100%

  Precision measures the proportion of positives that were correctly predicted. It is a measure of how sure one can be about positive predictions.

- Recall

  Recall = TP / (TP + FN) * 100%

  Recall (also known as sensitivity) measures the proportion of actual positives that were predicted correctly. It is a measure of how sure one can be about not missing any positives.

- F1-score

  F1-score = 2* (Precision * Recall) / (Precision + Recall)

  F1-score is computed as the harmonic mean of precision and recall. It will have a value between zero and one. A score close to one indicates that false data has little influence on the result. A low F1-score suggests that the model is poor at detecting positives.

5.2 Additional ML Functional Performance Metrics for Classification, Regression and Clustering

There are numerous metrics for different types of ML problems (in addition to those related to classification described in Section 5.1). Some of the most commonly used metrics are described below.

Supervised Classification Metrics

- The receiver operating characteristic (ROC) curve is a graphical plot that illustrates the ability of a binary classifier as its discrimination threshold is varied. The method was originally developed for military radars, which is why it is so named. The ROC curve is plotted with true positive rate (TPR) (also known as recall) against the false positive rate (FPR = FP / (TN + FP)), with TPR on the y axis and FPR on the x axis.
- The area under curve (AUC) is the area under the ROC curve. It represents the degree of separability of a classifier, showing how well the model distinguishes between classes. With a higher AUC, the model's predictions are better.

Supervised Regression Metrics

For supervised regression models, the metrics represent how well the regression line fits the actual data points.

- Mean Square Error (MSE) is the average of the squared differences between the actual value and the predicted value. The value of MSE is always positive, and a value closer to zero suggests a better regression model. By squaring the difference, it ensures positive and negative errors do not cancel each other out.
- R-squared (also known as the coefficient of determination) is a measure of how well the regression model fits the dependent variables.

Unsupervised Clustering Metrics

For unsupervised clustering, there are several metrics that represent the distances between the various clusters and the closeness of data points within a given cluster.

- Intra-cluster metrics measure the similarity of data points within a cluster.
- Inter-cluster metrics measure the similarity of data points in different clusters.
- The silhouette coefficient (also known as silhouette score) is a measure (between -1 and +1) based on the average inter-cluster and intra-cluster distances. A score of +1 means the clusters are well-separated, a score of zero implies random clustering, and a score of -1 means the clusters are wrongly assigned.

5.3 Limitations of ML Functional Performance Metrics

ML functional performance metrics are limited to measuring the functionality of the model, e.g., in terms of accuracy, precision, recall, MSE, AUC and the silhouette coefficient. They do not measure other non-functional quality characteristics, such as those defined in ISO 25010 [S06] (e.g., performance efficiency) and those described in Chapter 2, (e.g., explainability, flexibility, and autonomy). In this syllabus, the term "ML functional performance metrics" is used because of the widespread use of the term "performance metrics" to refer to these functional metrics. Adding "ML functional" highlights that these metrics are specific to machine learning and have no relationship to performance efficiency metrics.

ML functional performance metrics are constrained by several other factors:

- For supervised learning, the ML functional performance metrics are calculated on the basis of labeled data, and the accuracy of the resultant metrics depends on correct labelling (see Section 4.5).
- The data used for measurement may not be representative (e.g., it may be biased) and the generated ML functional performance metrics depend on this data (see Section 2.4).
- The system may comprise several components, but the ML functional performance metrics only applies to the ML model. For example, the data pipeline is not considered by the ML functional performance metrics to evaluate the model.
- Most of the ML functional performance metrics can only be measured with support from tools.

5.4 Selecting ML Functional Performance Metrics

It is not normally possible to build an ML model that achieves the highest score for all of the ML functional performance metrics generated from a confusion matrix. Instead, the most appropriate ML functional performance metrics are selected as acceptance criteria, based on the expected use of the model (e.g., to minimize false positives, a high value of precision is required, whereas to minimize false negatives, the recall metric should be high). The following criteria can be used when selecting the ML functional performance metrics described in Sections 5.1 and 5.2:

- Accuracy: This metric is likely to be applicable if the datasets are symmetric (e.g., false positive and false negative counts and costs are similar). This metric becomes a poor choice if one class of data dominates over the others, in which case the F1-score should be considered.
- Precision: This can be a suitable metric when the cost of false positives is high and confidence in positive outcomes needs to be high. A spam filter, (where classifying an email as spam is considered positive), is an example where high precision is required, as putting too many emails in the spam folder that are not actually spam will not be acceptable to most users. When the classifier deals with situations where a very large percentage of cases are positive, then using precision alone is unlikely to be a good choice.
- Recall: When it is critical that positives should not be missed, then a high recall score is important. For example, missing any true positive results in cancer detection and marking them as negative (i.e., no cancer detected) is likely to be unacceptable.
- F1-score – F1-score is most useful when there is an imbalance in the expected classes and when the precision and recall are of similar importance.

In addition to the above metrics, several metrics are described in Section 5.2. These may be applicable for given ML problems, for example:

- The AUC for the ROC curve may be used for supervised classification problems.
- MSE and R-squared may be used for supervised regression problems.

- Inter-cluster metrics, intra-cluster metrics and the silhouette coefficient may be used for unsupervised clustering problems.

5.4.1 Hands-On Exercise: Evaluate the Created ML Model

Using the classification model trained in the previous exercise, calculate and display the values for accuracy, precision, recall and F1-score. Where applicable, use the library functions provided by your development framework to perform the calculations.

5.5 Benchmark Suites for ML

New AI technologies such as new datasets, algorithms, models, and hardware are released regularly, and it can be difficult to determine the relative efficacy of each new technology.

To provide objective comparisons between these different technologies, industry-standard ML benchmark suites are available. These cover a wide range of application areas, and provide tools to evaluate hardware platforms, software frameworks and cloud platforms for AI and ML performance.

ML benchmark suites can provide various measures, including training times (e.g., how fast a framework can train an ML model using a defined training dataset to a specified target quality metric, such as 75% accuracy), and inference times (e.g., how fast a trained ML model can perform inference).

ML benchmark suites are provided by several different organizations, such as:

- MLCommons [R18]: This is a not-for-profit organization formed in 2020 and previously named ML Perf, which provides benchmarks for software frameworks, AI-specific processors and ML cloud platforms.
- DAWNBench [R19]: This is an ML benchmark suite from Stanford University.
- MLMark [R20]: This is an ML benchmark suite designed to measure the performance and accuracy of embedded inference from the Embedded Microprocessor Benchmark Consortium.

6 ML - Neural Networks and Testing – 65 minutes

Keywords

None

AI-Specific Keywords

Activation value, deep neural network (DNN), ML training data, multi-layer perceptron, neural network, neuron coverage, perceptron, sign change coverage, sign-sign coverage, supervised learning, threshold coverage, training data, value change coverage

Learning Objectives for Chapter 6:

6.1 Neural Networks

AI-6.1.1	K2	Explain the structure and function of a neural network including a DNN.
HO-6.1.1	H1	Experience the implementation of a perceptron.

6.2 Coverage Measures for Neural Networks

AI-6.2.1	K2	Describe the different coverage measures for neural networks.

6.1 Neural Networks

Artificial neural networks were initially intended to mimic the functioning of the human brain, which can be thought of as many connected biological neurons. The single-layer perceptron is one of the first examples of the implementation of an artificial neural network and comprises a neural network with just one layer (i.e., a single neuron). It can be used for supervised learning of classifiers, which decide whether an input belongs to a specific class or not.

Most current neural networks are considered to be deep neural networks because they comprise several layers and can be considered as multi-layer perceptrons (see Figure 3).

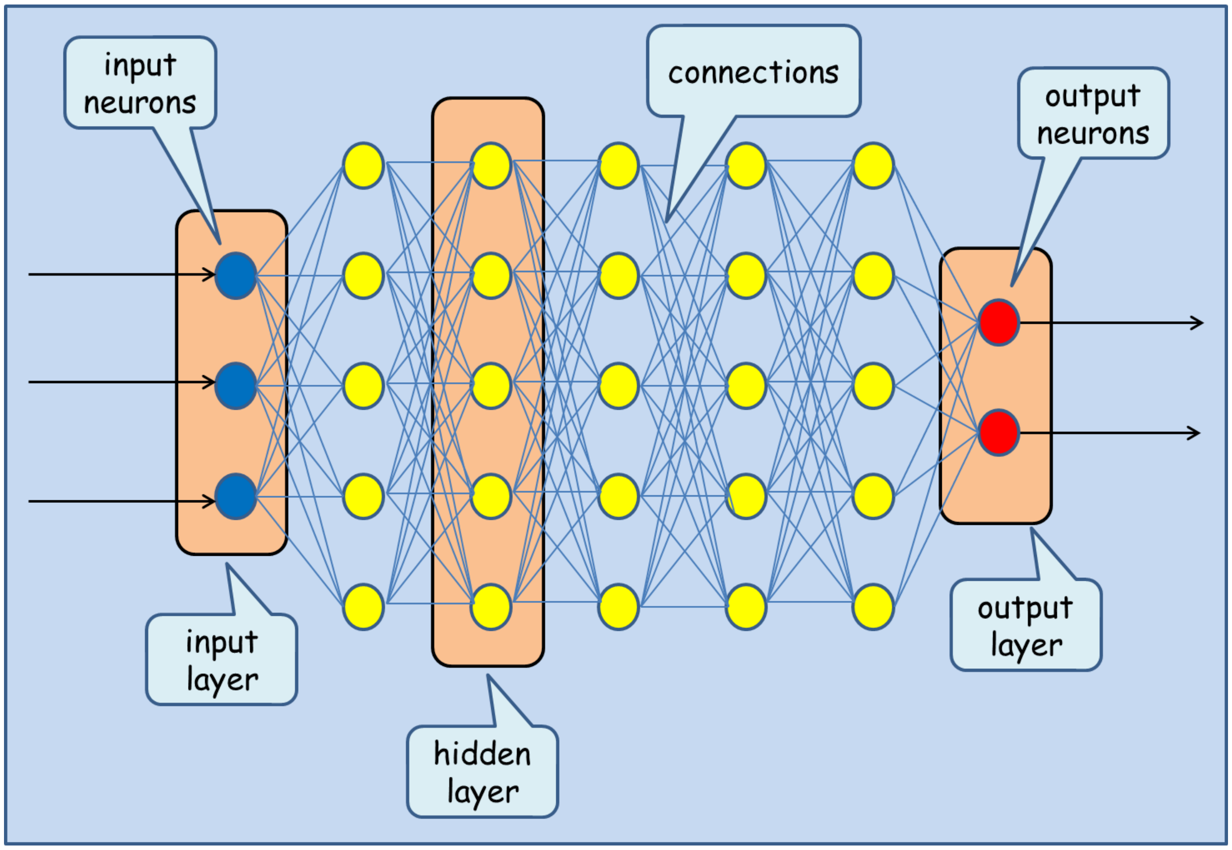

Figure 3 Structure of a deep neural network

A deep neural network comprises three types of layers. The input layer receives inputs, for example pixel values from a camera. The output layer provides results to the outside world. This might be, for example, a value signifying the likelihood that the input image is a cat. Between the input and output layers are hidden layers made up of artificial neurons, which are also known as nodes. The neurons in one layer are connected to each of the neurons in the next layer and there may be different numbers of neurons in each successive layer. The neurons perform computations and pass information across the network from the input neurons to the output neurons.

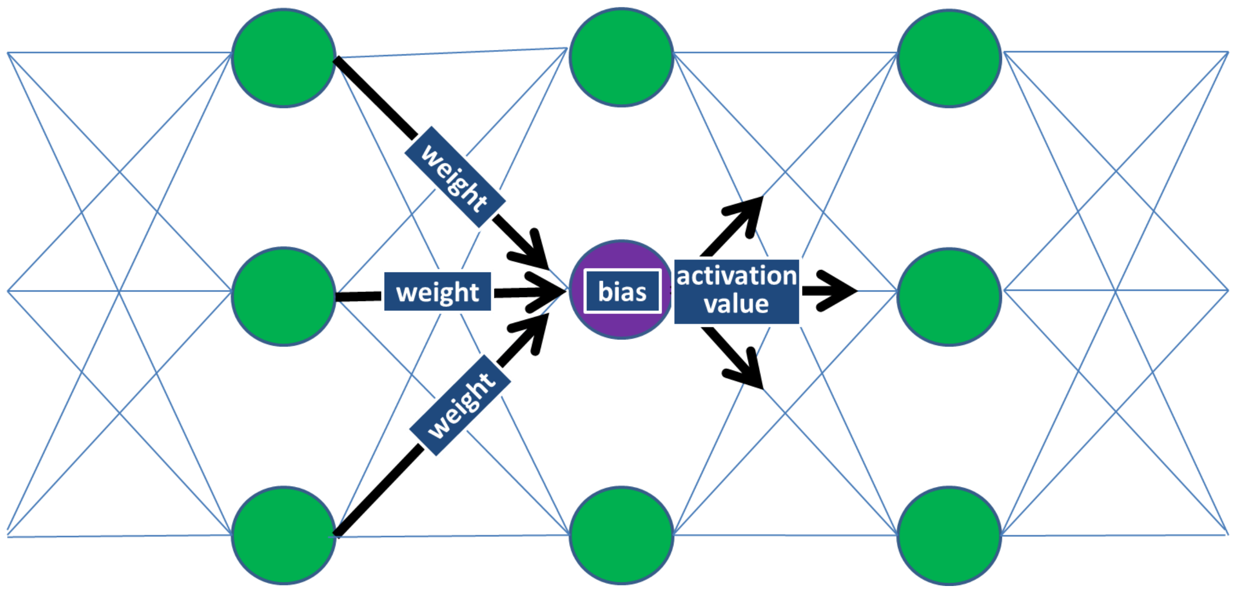

Figure 4 Computation performed by each neuron

As shown in Figure 4, the computation performed by each neuron (except those in the input layer) generates what is known as the activation value. This value is calculated by running a formula (the activation function) that receives as input the activation values from all the neurons in the previous layer, the weights assigned to the connections between the neurons (these weights change as the network learns), and the individual bias of each neuron. Note that this bias is a preset constant value and is not related to the bias considered earlier in Section 2.4). Running different activation functions can result in different activation values being calculated. These values are typically centered around zero and have a range between -1 (meaning that the neuron is “disinterested”) and +1 (meaning that the neuron is “very interested”).

When training the neural network, each neuron is preset to a bias value and the training data is passed through the network, with each neuron running the activation function, to eventually generate an output. The generated output is then compared with the known correct result (labeled data is used in this example of supervised learning). The difference between the actual output and the known correct result is then fed back through the network to modify the values of the weights on the connections between the neurons in order to minimize this difference. As more training data is fed through the network, the weights are gradually adjusted as the network learns. Ultimately, the outputs produced are considered good enough to end training.

6.1.1 Hands-On Exercise: Implement a Simple Perceptron

Students will be led through an exercise demonstrating a Perceptron learning a simple function, such as an AND function.

The exercise should cover how a Perceptron learns by modifying weights across a number of epochs until the error is zero. Various mechanisms may be used for this activity (e.g., spreadsheet, simulation).

6.2 Coverage Measures for Neural Networks

Achieving white-box test coverage criteria (e.g., statement, branch, modified condition/decision coverage (MC/DC) [I01] is mandatory for compliance with some safety-related standards [S07] when using traditional imperative source code, and is recommended by many test practitioners for other critical applications. Monitoring and improving coverage supports the design of new test cases, leading to increased confidence in the test object.

Using such measures for measuring the coverage of neural networks provides little value as the same code tends to be run each time the neural network is executed. Instead, coverage measures have been proposed based on the coverage of the structure of the neural network itself, and more specifically, the neurons within it. Most of these measures are based on the activation values of the neurons.

Coverage for neural networks is a new research area. Academic papers have only been published since 2017, and as such, there is little objective evidence available (e.g., duplicated research results) that show the proposed measures are effective. It should be noted, however, that despite statement and decision coverage having been used for over 50 years, there is also little objective evidence of their relative effectiveness, even though they have been mandated for measuring coverage of software in safety-related applications, such as medical devices and avionics systems.

The following coverage criteria for neural networks have been proposed and applied by researchers to a variety of applications:

- Neuron coverage: Full neuron coverage requires that each neuron in the neural network achieves an activation value greater than zero [B12]. This is very easy to achieve in practice and research has shown that almost 100% coverage is achieved with very few test cases on a variety of deep neural networks. This coverage measure may be most useful as an alarm signal when it is not achieved.
- Threshold coverage: Full threshold coverage requires that each neuron in the neural network achieves an activation value greater than a specified threshold. The researchers who created the DeepXplore framework actually suggested that neuron coverage should be measured based on the activation value exceeding a threshold which would change based on the situation. They performed their research with a threshold of 0.75 when they reported efficiently finding thousands of incorrect corner case behaviors using this white-box approach. This type of coverage has been renamed here to distinguish it more easily from neuron coverage with a threshold set to zero, as some other researchers use the term “neuron coverage” to mean neuron coverage with a threshold of zero.
- Sign-Change coverage: To achieve full sign-change coverage, test cases need to cause each neuron to achieve both positive and negative activation values [B13].
- Value-Change coverage: To achieve full value-change coverage, test cases need to cause each neuron to achieve two activation values, where the difference between the two values exceeds some chosen value [B13].
- Sign-Sign coverage: This coverage considers pairs of neurons in adjacent layers and the sign taken by their activation values. For a pair of neurons to be considered covered, a test case needs to show that changing the sign of a neuron in the first layer causes the neuron in the second layer to change its sign, while the signs of all other neurons in the second layer remain unchanged [B13]. This is a similar concept to MC/DC coverage for imperative source code.

Researchers have reported on further coverage measures based on layers (although simpler than sign-sign coverage), and a successful approach using nearest neighbor algorithms to identify meaningful change in neighboring sets of neurons has been implemented in the TensorFuzz tool [B14].

7 Testing AI-Based Systems Overview – 115 minutes

Keywords

Input data testing, ML model testing

AI-Specific Keywords

AI component, automation bias, big data, concept drift, data pipeline, ML functional performance metrics, training data

Learning Objectives for Chapter 7:

7.1 Specification of AI-Based Systems

AI-7.1.1	K2	Explain how system specifications for AI-based systems can create challenges in testing.

7.2 Test Levels for AI-Based Systems

AI-7.2.1	K2	Describe how AI-based systems are tested at each test level

7.3 Test Data for Testing AI-Based Systems

AI-7.3.1	K1	Recall those factors associated with test data that can make testing AI-based systems difficult.

7.4 Testing for Automation Bias in AI-Based Systems

AI-7.4.1	K2	Explain automation bias and how this affects testing.

7.5 Documenting an ML Model

AI-7.5.1	K2	Describe the documentation of an AI component and understand how documentation supports the testing of AI-based systems.

7.6 Testing for Concept Drift

AI-7.6.1	K2	Explain the need for frequently testing the trained model to handle concept drift.

7.7 Selecting a Test Approach for an ML System

AI-7.7.1	K4	For a given scenario determine a test approach to be followed when developing an ML system.

7.1 Specification of AI-Based Systems

System requirements and design specifications are equally important for both AI-based systems and conventional systems. These specifications provide the basis for testers to check whether actual system behavior aligns with the specified requirements. However, if the specifications are incomplete and lack testability, this introduces a test oracle problem (see Section 8.7).

There are several reasons why the specification of AI-based systems can be particularly challenging:

- In many AI-based systems projects, requirements are specified only in terms of high-level business goals and required predictions. A reason for this is the exploratory nature of AI-based system development. Often, AI-based systems projects start with a dataset, and the goal is to determine which predictions can be obtained from that data. This is in contrast with specifying the required logic from the start of a conventional project.
- The accuracy of the AI-based system is often unknown until the test results from independent testing are available. Along with the exploratory development approach, this often leads to inadequate specifications as implementation is already in progress by the time the desired acceptance criteria are determined.
- The probabilistic nature of many AI-based systems can make it necessary to specify tolerances for some of the expected quality requirements, such as the accuracy of predictions.
- Where the system goals call for replicating human behavior, rather than providing specific functionality, this often leads to poorly specified behavior requirements based on the system being as good as, or better than the human activities it aims to replace. This can make it difficult to define a test oracle, especially when the humans it is replacing vary widely in their capabilities.
- Where AI is used to implement user interfaces, such as by natural language recognition, computer vision, or physical interaction with humans, the systems need to demonstrate increased flexibility. However, such flexibility can also create challenges in identifying and documenting all the different ways in which such interactions might happen.
- Quality characteristics specific to AI-based systems, such as adaptability, flexibility, evolution, and autonomy, need to be considered and defined as part of requirements specification (see Chapter 2). The novelty of these characteristics can make them difficult to define and test.

7.2 Test Levels for AI-Based Systems

AI-based systems typically comprise both AI and non-AI components. Non-AI components can be tested using conventional approaches [I01], while AI components and systems containing AI components may need to be tested differently in some respects, as described below. For all test levels that include the testing of AI components, it is important for the testing to be closely supported by data engineers/scientists and domain experts.

A major difference from the test levels used for conventional software is the inclusion of two new specialized test levels to explicitly handle the testing of the input data and the models used in AI-based systems [B15]. Most of this section is applicable to all AI-based systems, although some parts are specifically focused on ML.

7.2.1 Input Data Testing

The objective of input data testing is to ensure that the data used by the system for training and prediction is of the highest quality (see Section 4.3). It includes the following:

- Reviews
- Statistical techniques (e.g., testing data for bias)
- EDA of the training data
- Static and dynamic testing of the data pipeline

The data pipeline typically comprises several components performing data preparation (see Section 4.1), and the testing of these components includes both component testing and integration testing. The data pipeline for training may be quite different from the data pipeline used to support operational prediction. For training, the data pipeline can be considered a prototype, compared to the fully engineered, automated version used operationally. For this reason, the testing of these two versions of the data pipeline may be quite distinct. However, testing the functional equivalence of the two versions should also be considered.

7.2.2 ML Model Testing

The objective of ML model testing is to ensure that the selected model meets any performance criteria that may have been specified. This includes:

- ML functional performance criteria (see Sections 5.1 and 5.2)
- ML non-functional acceptance criteria that are appropriate for the model in isolation, such as speed of training, speed of prediction, computing resources used, adaptability, and transparency.

ML model testing also aims to determine that the choice of ML framework, algorithm, model, model settings and hyperparameters is as close to optimal as possible. Where appropriate, ML model testing may also include testing to achieve white-box coverage criteria (see Section 6.2). The selected model is later integrated with other components, AI and non-AI.

7.2.3 Component Testing

Component testing is a conventional test level which is applicable to any non-model components, such as user interfaces and communication components.

7.2.4 Component Integration Testing

Component integration testing is a conventional test level which is conducted to ensure that the system components (both AI and non-AI) interact correctly. It tests that the inputs from the data pipeline are received as expected by the model, and that any predictions produced by the model are exchanged with the relevant system components (e.g., the user interface) and used correctly by them. Where AI is provided as a service (see Section 1.7), it is normal to perform API testing of the provided service as part of component integration testing.

7.2.5 System Testing

System testing is a conventional test level which is conducted to ensure that the complete system of integrated components (both AI and non-AI) performs as expected, from both functional and non-functional viewpoints, in a test environment that is closely representative of the operational environment. Depending on the system, this testing may take the form of field trials in the expected operational environment or testing within a simulator (e.g., if test scenarios are hazardous or difficult to replicate in an operational environment).

During system testing, the ML functional performance criteria are re-tested to ensure that the test results from the initial ML model testing are not adversely affected when the model is embedded within a complete system. This testing is especially important where the AI component has been deliberately changed (e.g., compressing a DNN to reduce its size).

System testing is also the test level in which many of the non-functional requirements for the system are tested. For example, adversarial testing may be performed to test for robustness, and the system may be tested for explainability. Where appropriate, interfaces with hardware components (e.g., sensors) may be tested as part of system testing.

7.2.6 Acceptance Testing

Acceptance testing is a conventional test level and is used to determine whether the complete system is acceptable to the customer. For AI-based systems, the definition of acceptance criteria can be challenging (see Section 8.8). Where AI is provided as a service (see Section 1.7), acceptance testing may be needed to determine the suitability of the service for the intended system and whether, for example, ML functional performance criteria have been sufficiently achieved.

7.3 Test Data for Testing AI-based Systems

Depending on the situation and the system under test (SUT), the acquisition of test data might present a challenge. There are several potential challenges in dealing with test data for AI-based systems, including:

- Big data (high-volume, high-velocity and high-variety data) can be difficult to create and manage. For example, it may be difficult to create representative test data for a system that consumes large volumes of images and audio at a high speed.
- Input data may need to change over time, particularly if it represents events in the real world. For example, recorded photographs to test a facial recognition system may need to be “aged” to represent the ageing of people over several years in real life.
- Personal or otherwise confidential data may need special techniques for sanitization, encryption, or redaction. Legal approval for use may also be required.
- When testers use the same implementation as the data scientists for data acquisition and data pre-processing, then defects in these steps may be masked.

7.4 Testing for Automation Bias in AI-Based Systems

One category of AI-based systems helps humans in decision-making. However, there is occasionally a tendency for humans to be too trusting of these systems. This misplaced trust may be called either automation bias or complacency bias, and takes two forms.

- The first form of automation/complacency bias is when the human accepts recommendations provided by the system and fails to consider inputs from other sources (including themselves). For example, a procedure where a human keys data into a form might be improved by using machine learning to pre-populate the form, and the human then validates this data. It has been shown that this form of automation bias typically reduces the quality of decisions made by 5%, but this could be much greater depending on the system context [B16]. Similarly, the automatic correction of typed text (e.g., in mobile phone messages) is often faulty and could change the meaning. Users often do not notice this, and do not override the mistake.
- The second form of automation/complacency bias is where the human misses a system failure because they do not adequately monitor the system. For example, semi-autonomous vehicles are becoming increasingly self-driving, but still rely on a human to take over in the event of an imminent accident. Typically, the human vehicle occupant gradually becomes too trusting of the system's abilities to control the vehicle and they begin to pay less attention. This may lead to a situation where they are unable to react appropriately when needed.

In both scenarios, testers should understand how human decision-making may be compromised, and test for both the quality of the system's recommendations and the quality of the corresponding human input provided by representative users.

7.5 Documenting an AI Component

The typical content for the documentation of an AI component includes:

- General: Identifiers, description, developer details, hardware requirements, license details, version, date and point of contact.
- Design: Assumptions and technical decisions.
- Usage: Primary and secondary use cases, typical users, approach to self-learning, known bias, ethical issues, safety issues, transparency, decision thresholds, platform and concept drift.
- Datasets: Features, collection, availability, pre-processing requirements, use, content, labelling, size, privacy, security, bias/fairness and restrictions/constraints.
- Testing: Test dataset (description and availability), independence of testing, test results, testing approach for robustness, explainability, concept drift and portability.
- Training and ML Functional Performance: ML algorithm, weights, validation dataset, selection of ML functional performance metrics, thresholds for ML functional performance metrics, and actual ML functional performance metrics.

Clear documentation helps improve the testing by providing transparency on the implementation of the AI-based system. The key areas of documentation that are important to testing are:

- The purpose of the system, and the specification of functional and non-functional requirements. These types of documentation typically form part of the test basis.
- Architectural and design information, outlining how the different AI and non-AI components interact. This supports the identification of integration testing objectives, and may provide a basis for white-box testing of the system structure.
- The specification of the operating environment. This is required when testing the autonomy, flexibility and adaptability of the system.
- The source of any input data, including associated metadata. This needs to be clearly understood when testing the following aspects:
    - Functional correctness of untrustworthy inputs
    - Explicit or implicit data bias
    - Flexibility, including the mis-learning from poor data inputs for self-learning systems
- The way in which the system is expected to adapt to changes in its operational environment. This is needed as a test basis when testing for adaptability.
- Details of expected system users. This is needed to ensure that testing can be made representative.

7.6 Testing for Concept Drift

The operational environment can change over time without the trained model changing correspondingly. This phenomenon is known as concept drift and typically causes the outputs of the model to become increasingly less accurate and less useful. For example, the impact of marketing campaigns may result in a change in the behavior of potential customers over a period of time. Such changes could be seasonal or abrupt changes due to cultural, moral or societal changes which are external to the system. An example of such an abrupt change is the impact of the COVID-19 pandemic and its effect on the accuracy of the models used for sales projections and stock markets.

Systems that may be prone to concept drift should be regularly tested against their agreed ML functional performance criteria, to ensure that any occurrences of concept drift are detected soon enough for the problem to be mitigated. Typical mitigations may include retiring the system or re-training the system. In the case of re-training, this would be performed with up-to-date training data and followed by confirmation testing, regression testing, and possibly a form of A/B testing (see Section 9.4), where the updated B-system must outperform the original A-system.

7.7 Selecting a Test Approach for an ML System

An AI-based system will typically include both AI and non-AI components. The test approach is based on a risk analysis for such a system and will include both conventional testing as well as more specialized testing to address those factors specific to AI components and AI-based systems.

The following list provides some typical risks and corresponding mitigations, specific to ML systems. Note that this list only provides a limited set of examples and that there are many more risks specific to ML systems that require mitigation through testing.

Risk Aspect	**Description and possible Mitigations**
Data quality may be lower than expected.	This risk may become an issue in several ways, each of which may be prevented in different manners (see Section 4.4). Common mitigations include the use of reviews, EDA and dynamic testing.
The operational data pipeline may be faulty.	This risk can be partly mitigated by the dynamic testing of the individual pipeline components and the integration testing of the complete pipeline
The ML workflow used to develop the model may be sub-optimal. (See Section 3.2)	This risk could be due to the following: • A lack of up-front agreement on the ML workflow to be followed • A poor choice of workflow • Data engineers failing to follow the workflow Reviews with experts may mitigate the chance of choosing the wrong workflow, while more hands-on management or audits could address the problems of agreement on and implementation of the workflow.
The choice of ML framework, algorithm, model, model settings and/or hyperparameters may be sub-optimal.	This risk could be due to a lack of expertise of the decision-makers, or to the poor implementation of the evaluation and tuning steps (or test step) of the ML workflow. Reviews with experts may mitigate the chance of taking wrong decisions, and better management may ensure that the evaluation and tuning (and test) steps of the workflow are followed.
The desired ML functional performance criteria may not be delivered operationally, despite the ML component meeting those criteria in isolation.	This risk could be due to the datasets used for training and testing the model in isolation not being representative of the data encountered operationally. Reviews of the selected datasets by experts (or users) may mitigate the chance that the selected data is not representative.
The desired ML functional performance criteria are met, but the users may be unhappy with the delivered results.	This risk could be due to the selection of the wrong performance criteria (e.g., high recall was selected when high precision was needed). Reviews with experts may mitigate the chance of choosing the wrong ML functional performance metrics, or experience-based testing could also identify inappropriate criteria. The risk could also be due

Risk Aspect	Description and possible Mitigations
	to concept drift, in which case more frequent testing of the operational system could mitigate the risk.
The desired ML functional performance criteria are met, but the users may be unhappy with the delivered service.	This risk could be due to a lack of focus on the system's non-functional requirements). Note that the range of quality characteristics for AI-based systems extends beyond those listed in ISO/IEC 25010 (see Chapter 2). Using a risk-based approach to prioritize the quality characteristics and performing the relevant non-functional testing could mitigate this risk. Alternatively, the problem could be due to a combination of factors that could be identified through experience-based testing, as part of system testing. Chapter 8 provides guidance on how to test these characteristics.
The self-learning system may not be providing the service expected by users.	This risk could be due to various reasons, for example: • The data used by the system for self-learning may be inappropriate. In this case, reviews by experts could identify the problematic data. • The system may be failing due to new self-learnt functionality being unacceptable. This could be mitigated by automated regression testing including performance comparison with the previous functionality. • The system may be learning in a way that is not expected by the users, which could be detected by experience-based testing.
Users may be frustrated by not understanding how the system determines its decisions	This risk could be due to a lack of explainability, interpretability and/or transparency. See Section 8.6 for details on how to test for these characteristics.
Users may find that the model provides excellent predictions when the data is similar to the training data but provides poor results otherwise.	This risk may be due to overfitting (see Section 3.5.1), which may be detected by testing the model with a dataset that is completely independent from the training dataset or performing experience-based testing.

8 Testing AI-Specific Quality Characteristics – 150 minutes

Keywords

Test oracle

AI-Specific Keywords

Algorithmic bias, autonomous system, autonomy, expert system, explainability, inappropriate bias, interpretability, LIME method, ML training data, non-deterministic system, probabilistic system, sample bias, self-learning system, transparency

Learning Objectives for Chapter 8:

8.1 Challenges Testing Self-Learning Systems

AI-8.1.1	K2	Explain the challenges in testing created by the self-learning of AI-based systems.

8.2 Testing Autonomous AI-Based Systems

AI-8.2.1	K2	Describe how autonomous AI-based systems are tested

8.3 Testing for Algorithmic, Sample and Inappropriate Bias

AI-8.3.1	K2	Explain how to test for bias in an AI-based system.

8.4 Challenges Testing Probabilistic and Non-Deterministic AI-Based Systems

AI-8.4.1	K2	Explain the challenges in testing created by the probabilistic and non-deterministic nature of AI-based systems.

8.5 Challenges Testing Complex AI-based Systems

AI-8.5.1	K2	Explain the challenges in testing created by the complexity of AI-based systems.

8.6 Testing the Transparency, Interpretability and Explainability of AI-based Systems

AI-8.6.1	K2	Describe how the transparency, interpretability and explainability of AI-based systems can be tested.
HO-8.6.1	H2	Use a tool to show how explainability can be used by testers.

8.7 Test Oracles for AI-Based Systems

AI-8.7.1	K2	Explain the challenges in creating test oracles resulting from the specific characteristics of AI-based systems.

8.8 Test Objectives and Acceptance Criteria

AI-8.8.1	K4	Select appropriate test objectives and acceptance criteria for the AI-specific quality characteristics of a given AI-based system.

8.1 Challenges Testing Self-Learning Systems

There are several potential challenges to overcome when testing self-learning systems (see Chapter 2 for more details on these systems), including:

- Unexpected change: The original requirements and constraints within which the system should work are generally known, but there may be little or no information available on the changes made by the system itself. It is normally possible to test against the original requirements and design (and any specified constraints), but if the system has devised an innovative implementation or gamed a solution (the implementation of which cannot be seen), it may be difficult to design tests which are appropriate for this new implementation. In addition, when systems change themselves (and their outputs), the results of previously passing tests can change. This is a test design challenge. It may be addressed by designing appropriate tests that remain relevant as the system changes its behavior, so preventing a potential regression testing problem. However, it may also require new tests to be designed based on observed new system behaviors.
- Complex acceptance criteria: It may be necessary to define expectations for improvement by the system when it self-learns. For example, it may be assumed that if the system changes itself, its overall functional performance should improve. Additionally, specifying anything other than simple “improvement” can quickly become complex. For example, a minimum improvement might be expected (rather than simply any improvement), or the required improvement may be linked to environmental factors (e.g., a minimum 10% improvement in functionality X is required if environmental factor F changes by more than Y). These problems may be addressed through the specification and testing against the more complex acceptance criteria, and by maintaining a continuous record of the current system baseline functional performance.
- Insufficient testing time: It may be necessary to know how quickly the system is expected to learn and adapt, given different scenarios. These acceptance criteria may be difficult to specify and acquire. If a system adapts quickly, there might be insufficient time to manually execute new tests after each change, so it may be necessary to write tests that can be run automatically when the system changes itself. These challenges can be addressed through the specification of appropriate acceptance criteria (see Section 8.8) and automated continuous testing.
- Resource requirements: The system requirements might include acceptance criteria for the resources which the system is permitted to use when performing self-learning or adaptation. This may include, for example, the amount of processing time and memory allowed to be used to improve. Additionally, consideration needs to be given on whether this resource usage should be linked to a measurable improvement in functionality or accuracy. This challenge affects the specification of acceptance criteria.
- Insufficient specifications of operational environment: A self-learning system may change if the environmental inputs that it receives are outside expected ranges, or if they are not reflected in the training data. These inputs may be attacks in the form of data poisoning (see Section 9.1.2). It can be difficult to predict the full range of operational environments and environmental changes, and to therefore identify the full set of representative test cases and environmental requirements. Ideally, the full scope of possible changes in the operational environment to which the system is expected to respond will be defined as acceptance criteria.

- Complex test environment: Managing the test environment to ensure it can mimic all the potential high-risk operational environment changes is a challenge and may involve the use of test tools (e.g., a fault injection tool). Depending on the nature of the operating environment, this may be tested by manipulating inputs and sensors, or by obtaining access to different physical environments in which the system can be tested.
- Undesirable behavior modifications: A self-learning system modifies its behavior based on its inputs and it may not be possible for testers to prevent this from occurring. This may arise, for example, if a third-party system is being used, or if the production system is being tested. By repeating the same tests, a self-learning system may become more effective at responding to those tests, which may then influence the long-term behavior of the system. It is therefore important to prevent a situation where the testing causes a self-learning system to adversely change its behavior. This is a challenge for test case design and test management.

8.2 Testing Autonomous AI-Based Systems

Autonomous systems must be able to determine when they require human intervention and when they do not. Therefore, testing the autonomy of AI-based systems requires that conditions are created for the system to exercise this decision-making.

Testing for autonomy may require:

- Testing whether the system requests human intervention for a specific scenario when the system should be relinquishing control. Such scenarios could include a change to the operational environment, or the system exceeding the limits of its autonomy.
- Testing whether the system requests human intervention when the system should be relinquishing control after a specified period of time.
- Testing whether the system unnecessarily requests human intervention when it should still be working autonomously.

It may be helpful to use boundary value analysis applied to the operating environment to generate the necessary conditions for this testing. It can be challenging to define how the parameters that determine autonomy manifest themselves in the operating environment, and to create the test scenarios which depend on the nature of the autonomy.

8.3 Testing for Algorithmic, Sample and Inappropriate Bias

An ML system should be evaluated against the different biases and actions taken to remove inappropriate bias. This may involve positive bias being deliberately introduced to counter the inappropriate bias.

Testing with an independent dataset can often detect bias. However, it can be difficult to identify all the data that causes bias because the ML algorithm can use combinations of seemingly unrelated features to create unwanted bias.

AI-based systems should be tested for algorithmic bias, sample bias and inappropriate bias (see Section 2.4). This may involve:

- Analysis during the model’s training, evaluation and tuning activities to identify whether algorithmic bias is present.

- Reviewing the source of the training data and the processes used to acquire it, such that the presence of sample bias can be identified.
- Reviewing the pre-processing of data as part of the ML workflow to identify whether the data has been affected in a way that could cause sample bias.
- Measuring how changes in system inputs affect system outputs over a large number of interactions, and examining the results based on the groups of people or objects that the system may be inappropriately biased towards, or against. This is similar to the LIME (Local Interpretable Model-Agnostic Explanations) method discussed in 8.6, and may be carried out in a production environment as well as part of testing prior to release.
- Obtaining additional information concerning the attributes of the input data potentially related to bias and correlating it to the results. This could relate to demographic data, for example, which might be appropriate when testing for inappropriate bias that affects groups of people, where membership of a group is relevant to assessing bias but is not an input to the model. This is because the bias can be based on “hidden” variables which are not explicitly present in the input data, but are inferred by the algorithm.

8.4 Challenges Testing Probabilistic and Non-Deterministic AI-Based Systems

Most probabilistic systems are also non-deterministic, and so the following list of testing challenges typically applies to AI-based systems with any of these attributes:

- There may be multiple, valid outcomes from a test with the same set of preconditions and inputs. This makes the definition of expected results more challenging and can cause difficulties:
  - when tests are reused for confirmation testing.
  - when tests are reused for regression testing.
  - where reproducibility of testing is important.
  - when the tests are automated.
- The tester typically requires a deeper knowledge of the required system behavior so that they can come up with reasonable checks for whether the test has passed rather than simply stating an exact value for the expected test result. For example, testers may need to define more sophisticated expected results compared with conventional systems. These expected test results may include tolerances (e.g., “is the actual result within 2% of the optimal solution?”).
- Where a single definitive output from a test is not possible due to the probabilistic nature of the system, it is often necessary for the tester to run a test several times in order to generate a statistically valid test result.

8.5 Challenges Testing Complex AI-Based Systems

AI-based systems are often used to implement tasks that are too complex for humans to perform. This can lead to a test oracle problem because testers are unable to determine the expected results as they would normally do (see Section 8.7). For example, AI-based systems are often used to identify patterns in large volumes of data. Such systems are used because they can find patterns that humans, even after much analysis, simply cannot find manually. Understanding the required behavior of such systems in sufficient depth to be able to generate expected results can be challenging.

A similar problem arises when the internal structure of an AI-based system is generated by software, making it too complex for humans to understand. This leads to the situation where the AI-based system can only be tested as a black box. Even when the internal structure is visible, this provides no additional useful information to help with the testing.

The complexity of AI-based systems increases when they provide probabilistic results and are non-deterministic in nature (see Section 8.4).

The problems with non-deterministic systems are exacerbated when an AI-based system consists of several interacting components, each providing probabilistic results. For example, a facial recognition system is likely to use one model to identify a face within an image, and a second model to recognize which face has been identified. The interactions between AI components can be complex and difficult to comprehend, making it difficult to identify all the risks, and design tests that verify the system adequately.

8.6 Testing the Transparency, Interpretability and Explainability of AI-Based Systems

Information on how the system has been implemented may be provided by the system developers. This may include the sources of training data, how labelling was conducted, and how the system components have been designed. When this information is not available, it can make the design of tests challenging. For example, if training data information is not available, then identifying potential gaps in such data and testing the impact of those gaps, becomes difficult. This situation can be compared to black-box and white-box testing, and has similar advantages and disadvantages. Transparency can be tested by comparing the information documented on the data and algorithm to the actual implementation and determining how closely they match.

With ML, it can often be more difficult to explain the link between a specific input and a specific output, than with conventional systems. This low level of explainability is primarily because the model generating the output is itself generated by code (the algorithm) and does not reflect the way humans think about a problem. Different ML models provide different levels of explainability and should be selected based on the requirements for the system, which may include explainability and testability.

One method to understand explainability is through the dynamic testing of the ML model when applying perturbations to the test data. Methods exist for quantifying explainability in this manner and for providing visual explanations of it. Some of these methods are model-agnostic, while others are specific to a particular type of model and require access to it. Exploratory testing can also be used to better understand the relationship between the inputs and outputs of a model.

The LIME method is model-agnostic and uses dynamically injected input perturbations and the analysis of outputs to provide testers with a view of the relationship between inputs and outputs. This can be an effective method for providing model explainability. However it is limited to providing possible reasons for the outputs, rather than a definitive reason, and is not applicable for all types of algorithms.

The interpretability of an AI-based system is heavily dependent on who this applies to. Different stakeholders may have different requirements in terms of how well they need to grasp the underlying technology.

Measuring and testing the level of understanding for both interpretability and explainability can be challenging as stakeholders will vary in their levels of ability and may not agree. In addition, identifying the profile of typical stakeholders may be difficult for many types of systems. Where performed, this testing typically takes the form of user surveys and/or questionnaires.

8.6.1 Hands-On Exercise: Model Explainability

Use an appropriate tool to provide explainability based on the previously created model. For example, for an image classification model or a text classification model, a model-agnostic method, such as LIME, may be appropriate.

Students should use the tool to generate explanations of model decisions; in particular, how the features in the inputs influence the outputs.

8.7 Test Oracles for AI-Based Systems

A major problem with the testing of AI-based systems can be the specification of expected results. A test oracle is the source used to determine the expected result of a test [I01]. A challenge in determining expected results is known as the test oracle problem.

With complex, non-deterministic or probabilistic systems, it can be difficult to establish a test oracle without knowing the "ground truth" (i.e., the actual result in the real world that the AI-based system is trying to predict). This "ground truth" is distinct from a test oracle, in that a test oracle may not necessarily provide an expected value, but only a mechanism with which to determine whether the system is operating correctly or not.

AI-based systems can evolve (see Section 2.3), and the testing of self-learning systems (see Section 8.1) can also suffer from test oracle problems as they modify themselves and can thereby make it necessary to frequently update the functional expectations of the system.

A further cause of difficulty in obtaining an effective test oracle is that in many cases, the correctness of the software behavior is subjective. Virtual assistants (e.g., Siri and Alexa) are an example of this problem in that different user often have quite different expectations and may experience different results depending on their choice of words and clarity of speech.

In some situations, it may be possible to define the expected result with limits or tolerances. For example, the stopping point for an autonomous car could be defined as within a maximum distance of a specific point. In the context of expert systems, the determination of the expected results may be achieved by consulting an expert (noting that the expert's opinion may still be wrong). There are several important factors to consider in such circumstances:

- Human experts vary in their levels of competence. Experts involved need to be at least as competent as the experts the system is intended to replace.
- Experts may not agree with each other, even when presented with the same information.
- Human experts may not approve of the automation of their judgement. In such cases their rating of potential outputs should be double-blind (i.e., neither the experts nor the evaluators of the outputs should know which ratings were automated).

- Humans are more likely to caveat responses (e.g., with phrases like "I'm not sure, but..."). If this kind of caveat is not available to the AI-based system, this should be considered when comparing the responses.

Test techniques exist which can alleviate the test oracle problem, such as A/B testing (see Section 9.4), back-to-back testing (see Section 9.3) and metamorphic testing (see Section 9.5).

8.8 Test Objectives and Acceptance Criteria

Test objectives and acceptance criteria for a system need to be based on the perceived product risks. These risks can often be identified from an analysis of the required quality characteristics. The quality characteristics for an AI-based system include those traditionally considered in ISO/IEC 25010 [S06] (i.e., functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability) but should also include a consideration of the following aspects:

Aspect	**Acceptance Criteria**
Adaptability	• Check the system still functions correctly and meets non-functional requirements when it adapts to a change in its environment. This may be implemented as a form of automated regression testing. • Check the time the system takes to adapt to a change in its environment. • Check the resources used when the system adapts to a change in its environment.
Flexibility	• Consider how the system copes in contexts outside the initial specification. This may be implemented as a form of automated regression testing executed in the changed operational environment. • Check the time the system takes and/or the resources used to change itself to manage a new context.
Evolution	• Check how well the system learns from its own experience. • Check how well the system copes when the profile of data changes (i.e., concept drift).
Autonomy	• Check how the system responds when it is forced outside of the operational envelope in which it is expected to be fully autonomous. • Check whether the system can be "persuaded" to request human intervention when it should be fully autonomous.
Transparency, interpretability and explainability	• Check transparency by reviewing the ease of accessing the algorithm and dataset.

Aspect	Acceptance Criteria
	• Check interpretability and explainability by questioning system users, or, if the actual system users are not available, people with a similar background.
Freedom from inappropriate bias	• Where systems are likely to be affected by bias, then this can be tested by using an independent bias-free test suite, using expert reviewers. • Compare the test results using external data such as census data in order to check for unwanted bias on inferred variables (external validity testing).
Ethics	• Check the system against a suitable checklist, such as the EC Assessment List for Trustworthy Artificial Intelligence [R21], which supports the key requirements outlined by the Ethics Guidelines for Trustworthy Artificial Intelligence (AI) [R22].
Probabilistic systems and non-deterministic systems	• This cannot be evaluated with precise acceptance criteria. When working correctly, the system may return slightly different results for the same tests.
Side-effects	• Identify potentially harmful side-effects and attempt to generate tests that cause the system to exhibit these side-effects.
Reward Hacking	• Independent tests can identify reward hacking when these tests use a different means of measuring success compared to the intelligent agent being tested.
Safety	• This needs to carefully evaluated, perhaps in a virtual test environment (see Section 10.2). This could include attempts to force a system to cause itself harm.

For ML systems, the required ML functional performance metrics for the ML model should be specified (see Chapter 5).

9 Methods and Techniques for the Testing of AI-Based Systems – 245 minutes

Keywords

A/B testing, adversarial testing, back-to-back testing, error guessing, experience-based testing, exploratory testing, metamorphic relation (MR), metamorphic testing (MT), pairwise testing, pseudo-oracle, test oracle problem, tours

AI-Specific Keywords

Adversarial attack, adversarial example, data poisoning, ML system, trained model

Learning Objectives for Chapter 9:

9.1 Adversarial Attacks and Data Poisoning

AI-9.1.1	K2	Explain how the testing of ML systems can help prevent adversarial attacks and data poisoning.

9.2 Pairwise Testing

AI-9.2.1	K2	Explain how pairwise testing is used for AI-based systems.
LO-9.2.1	H2	Apply pairwise testing to derive and execute test cases for an AI-based system.

9.3 Back-to-Back Testing

AI-9.3.1	K2	Explain how back-to-back testing is used for AI-based systems.

9.4 A/B Testing

AI-9.4.1	K2	Explain how A/B testing is applied to the testing of AI-based systems.

9.5 Metamorphic Testing

AI-9.5.1	K3	Apply metamorphic testing for the testing of AI-based systems.
HO-9.5.1	H2	Apply metamorphic testing to derive test cases for a given scenario and execute them.

9.6 Experience-Based Testing of AI-Based Systems

AI-9.6.1	K2	Explain how experience-based testing can be applied to the testing of AI-based systems.
HO-9.6.1	H2	Apply exploratory testing to an AI-based system.

9.7 Selecting Test Techniques for AI-Based Systems

AI-9.7.1	K4	For a given scenario, select appropriate test techniques when testing an AI-based system.

9.1 Adversarial Attacks and Data Poisoning

9.1.1 Adversarial Attacks

An adversarial attack is where an attacker subtly perturbs valid inputs that are passed to the trained model to cause it to provide incorrect predictions. These perturbed inputs, known as adversarial examples, were first noticed with spam filters, which could be tricked by slightly modifying a spam email without losing readability. Recently, they have become more associated with image classifiers. By simply changing a few pixels which are invisible to the human eye, it is possible to persuade a neural network to change its image classification to a very different object and with a high degree of confidence.

Adversarial examples are generally transferable [B17], which means that an adversarial example which causes one ML system to fail will often cause another ML system to fail that is trained to perform the same task. Even when the second ML system has been trained with different data and is based on different architectures, it is often still prone to failure with the same adversarial examples.

White-box adversarial attacks are where the attacker knows which algorithm was used to train the model and also which model settings and parameters were used (there is a reasonable level of transparency). The attacker uses this knowledge to generate adversarial examples by, for example, making small perturbations in inputs and monitoring which ones cause large changes to the model outputs.

Black-box adversarial attacks involve the attacker exploring the model to determine its functionality and then building a duplicate model that provides similar 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 original model.

If it is not possible to create a duplicate model, it may be possible to use high-volume automated testing to discover different adversarial examples and observe the results.

Adversarial testing simply involves performing adversarial attacks with the purpose of identifying vulnerabilities so that preventative measures can be taken to protect against future failures. Identified adversarial examples are added to the training data so that the model is trained to correctly recognize them.

9.1.2 Data Poisoning

Data poisoning attacks are where an attacker manipulates the training data to achieve one of two results. The attacker may insert backdoors or neural network trojans to facilitate future intrusions, or more often, they will use corrupted training data (e.g., mislabeled data) to induce the trained model to provide incorrect predictions.

Poisoning attacks may be targeted with the aim of causing the ML system to misclassify in specific situations. They may also be indiscriminate, such as with a denial-of-service attack. A well-known example of a poisoning attack was the corruption of the Microsoft Tay chatbot, whereby a relatively small number of harmful Twitter conversations trained the system through feedback to provide tainted conversations in the future. A commonly used form of data poisoning attack uses the false reporting of millions of spam emails as not being spam in an attempt to skew spam filtering software. An area of concern with data poisoning is the potential for public, widely used AI datasets to become poisoned.

Testing to prevent data poisoning is possible using EDA, as poisoned data may show up as outliers. In addition, data acquisition policies can be reviewed to ensure the provenance of training data. Where an operational ML system may be attacked by feeding it poisoned data, A/B testing (see Section 9.4) could be used to check that the updated version of the system is still closely aligned with the previous version. Alternatively, regression testing of an updated system using a trusted test suite may also determine if a system has been poisoned.

9.2 Pairwise Testing

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. Exhaustive testing would require all possible combinations of these parameters set to all possible values to be tested. However, since this would result in a practically infinite number of tests, test techniques are used to select a subset that can be run in the limited time available.

Where it is possible to combine numerous parameters, each of which may have many discrete values, combinatorial testing can be applied to significantly reduction the number of test cases required, ideally without compromising the defect detection capability of the test suite. There are several combinatorial testing techniques (see [I02] and [S08]). However, in practice, pairwise testing is the most widely used technique because it is easy to understand, has ample tool support. In addition, research has shown that most defects are caused by interactions involving few parameters [B33].

In practice, even the use of pairwise testing can result in extensive test suites for some systems, and the use of automation and virtual test environments (see Section 10.2) often becomes necessary to allow the necessary number of tests to be run. For example, when considering self-driving cars, high-level test scenarios for system testing need to exercise both the different environments in which the cars are expected to operate and the various vehicle functions. Thus, the parameters would need to include the range of environment constraints (e.g., road types and surfaces, weather and traffic conditions, and visibility) and the various self-driving functions (e.g., adaptive cruise control, lane keeping assistance, and lane change assistance). In addition to these parameters, inputs from sensors could be considered at varying levels of effectiveness (e.g., inputs from a video camera will degrade as a journey progress and it gets dirtier).

Research is currently unclear on the necessary level of rigor that would be required for the use of combinatorial testing with safety-critical AI-based systems such as self-driving cars. Even though pairwise testing may not be sufficient), it is known that the approach is effective at finding defects.

9.2.1 Hands-On Exercise: Pairwise Testing

For an implemented AI-based system with a minimum of five parameters and at least five hundred possible combinations, use a pairwise testing tool to identify a reduced set of pairwise combinations and execute tests for these combinations. Compare the number of pairwise combinations tested with the number required if all theoretically possible combinations were to be tested.

9.3 Back-to-Back Testing

One of the potential solutions to the test oracle problem (see Section 8.7) when testing AI-based systems is to use back-to-back testing. This is also known as differential testing. With back-to-back testing, an alternative version of the system is used as a pseudo-oracle and its outputs compared with the test results produced by the SUT. The pseudo-oracle could be an existing system, or it could be developed by a different team, possibly on a different platform, with a different architecture and with a different programming language. When testing the functional suitability (as opposed to non-functional requirements), the system used as a pseudo-oracle is not constrained to achieve the same non-functional acceptance criteria as the SUT. For example, it may not have to execute as quickly, in which case it can be far less expensive to build.

In the context of ML, it is possible to use different frameworks, algorithms and model settings to create an ML pseudo-oracle. In some situations, it may also be possible to create a pseudo-oracle using conventional, non-AI, software.

For pseudo-oracles to be effective in detecting defects, there should be no common software in both the pseudo-oracle and the SUT. Otherwise, it would be possible for the same defect in both to cause the two test results to match when both are defective. With so much immature, reusable, open-source AI software being used to develop AI-based systems, re-use of code between the pseudo-oracle and the SUT can compromise the pseudo-oracle. Poor documentation of reusable AI solutions may also make it difficult for the testers to recognize that this problem is occurring.

9.4 A/B Testing

A/B testing is a method where the response of two variants of the program (A and B) to the same inputs are compared with the purpose of determining which of the two variants is better. It is a statistical testing approach which typically requires the comparison of test results from several test runs to determine the difference between the programs.

A simple example of this method is where two promotional offers are emailed to a marketing list divided into two sets. Half of the list gets offer A, half gets offer B, and the success of each offer helps decide which to use in the future. Many e-commerce and web-based companies use A/B testing in production, diverting different consumers to different functionality, to help identify consumers' preferences.

A/B testing is one approach to solving the test oracle problem, where the existing system is used as a partial oracle. A/B testing does not generate test cases and provides no guidance on how the tests should be designed, although operational inputs are often used in tests.

A/B testing can be used to test updates to an AI-based system where there are agreed acceptance criteria, such as ML functional performance metrics, as described in Chapter 5. Whenever the system is updated, A/B testing is used to check that the updated variant performs as well as, or better than the previous variant. Such an approach can be used for a simple classifier, but can also be used for testing far more complex systems. For example, an update to improve the effectiveness of a smart city transport routing system can also be tested using A/B testing (e.g., comparing average commute times for two variants of the system on consecutive weeks).

A/B testing can also be used to test self-learning systems. When the system makes a change, automated tests are run, and the resultant system characteristics are compared with those before the change was made. If the system is improved, then the change is accepted, otherwise the system reverts to its previous state.

One major difference between A/B testing and back-to-back testing relates to the use of A/B testing to compare two variants of the same system and the use of back-to-back testing to detect defects.

9.5 Metamorphic Testing (MT)

Metamorphic testing [B18] is a technique aimed at generating test cases which are based on a source test case that has passed. One or more follow-up test cases are generated by changing (metamorphizing) the source test case based on a metamorphic relation (MR). The MR is based on a property of a required function of the test object, such that it describes how a change in a test case's test inputs are reflected in the same test case's expected results.

For example, consider a program that determines the average of a set of numbers. A source test case is generated comprising a set of numbers and an expected average, and the test case is run to confirm that it passes. It is now possible to generate follow-up test cases based on what is known about the program's average function. Initially, the order of the numbers being averaged may simply be changed. Given the average function, the expected result can be predicted to stay the same. Thus, a follow-up test case with the numbers in a different order can be generated without having to calculate the expected result. With a large set of numbers, this could lead to generating a large number of different sets of numbers in which the same numbers are used in different sequences and each of them could be used to create a separate follow-up test case. All of these test cases would be based on the same source test case and have the same expected result.

It is common to have MRs and follow-up test cases where the expected result is different from the original expected result of the source test case. For example, using the same average function, an MR can be derived in which each element of the input set is multiplied by two. The expected result for such a set is simply the original expected result multiplied by two. Similarly, any other value could be used as a multiplier to potentially generate an infinite number of follow-up test cases based on this MR.

MT can be used for most test objects and can be applied to both functional and non-functional testing (e.g., installability testing covers different target configurations where the installation parameters can be selected in different sequences). It is particularly useful where the generation of expected results is problematic, due to the lack of an inexpensive test oracle. This is the case with some AI-based systems that are based on the analysis of big data, or those where the testers are unclear on how the ML algorithm derives its predictions. In the area of AI, MT has been used for testing image recognition, search engines, route optimization and voice recognition, among others.

As explained above, MT can be based on a passed source test case, but it is also useful if it is not possible to verify that any source test case is correct. This may be the case, for example, where the program implements a function which is too complex for a human tester to replicate and use as a test oracle, such as with some AI-based systems. In this situation, MT can be used to generate one or more test cases which, when run, will create a set of outputs where the relationships between the outputs can then be checked for validity. With this form of MT, the individual tests are not known to be correct, but the relationships between them must hold true, so providing improved confidence in the program. An example could be an AI-based actuarial program that predicts an age at death based on a large set of data, where it is known, for example, that if the number of cigarettes smoked is increased, the predicted age at death should decrease (or, at least, stay the same).

MT is a relatively new test technique, first proposed in 1998. It differs from traditional test techniques in that the expected results of the follow-up test cases are not described in terms of absolute values, but are relative to the expected results in the source test case. It is based on an easily understood concept, can be applied by testers with little experience of applying the technique but who understand the application domain, and has similar costs compared to traditional techniques. It is also effective at revealing defects, with research showing that only three to six diverse MRs can reveal over 90% of the defects that could be detected using techniques based on a traditional test oracle [B19]. It is possible to automatically generate follow-up test cases from well-specified MRs and a source test case. However, commercial tools are not currently available, although Google is already applying automated MT to test Android graphics drivers using the GraphicsFuzz tool, which has been open sourced (see [R23]).

9.5.1 Hands-On Exercise: Metamorphic Testing

In this exercise, students will gain practical experience of the following:

- Deriving several metamorphic relations (MRs) for a given AI-based application or program. These MRs should include some where the expected results of the source and follow-up test cases are the same and some where they are different.
- Generating source test cases for the AI-based application or program. These do not have to be guaranteed to pass, but students should be reminded of the limitations of MT where there no such “gold standard” is available.
- Using the derived MRs and generated source test cases to derive follow-up test cases.
- Running the follow-up test cases.

9.6 Experience-Based Testing of AI-Based Systems

Experience-based testing includes error guessing, exploratory testing, and checklist-based testing [I01], all of which can be applied to the testing of AI-based systems.

Error guessing is typically based on testers knowledge, typical developer errors, and failures in similar systems (or previous versions). An example of error guessing applied to AI-based systems could be the use of knowledge about how ML systems have in the past failed due to the use of systemically biased training data.

In exploratory testing, tests are designed, generated, and executed in an iterative manner, with the opportunity for later tests to be derived, based on the test results of earlier tests. Exploratory testing is especially useful when there are poor specifications or test oracle problems, which is often the case for AI-based systems. As a result, exploratory testing is often used in this context and should be used to supplement the more systematic testing based on techniques, such as metamorphic testing (see Section 9.5).

A tour is a metaphor used for a set of strategies and goals for testers to refer to when they perform exploratory testing organized around a special focus [B20]. Typical tours for the exploratory testing of AI based systems might focus on the concepts of bias, underfitting and overfitting in ML systems. For example, a data tour might be applied to test the model. In this tour the tester could identify different types of data used for training, their distribution, their variations, their format and ranges, etc., and then use the data types to test the model.

ML systems are highly dependent on the quality of training data, and the existing field of EDA is closely related to the exploratory testing approach. EDA is where data are examined for patterns, relationships, trends and outliers. It involves the interactive, hypothesis-driven exploration of data and is described in [B21] as “We explore data with expectations. We revise our expectations based on what we see in the data. And we iterate this process.” EDA typically requires tool support in two areas; for interaction with the data, to allow analysts to better understand complex data, and for data visualization, to allow them to easily display analysis results. The use of exploratory techniques, primarily driven by data visualization, can help validate the ML algorithm being used, identify changes that result in efficient models, and leverage domain expertise [B22].

Google has a set of twenty-eight ML tests written as assertions, in the areas of data, model development, infrastructure and monitoring, which is used as a testing checklist within Google for ML systems [B23]. The Google "ML test checklist" is presented here as published by Google:

ML Data:

1. Feature expectations are captured in a schema.
2. All features are beneficial.
3. No feature's cost is too much.
4. Features adhere to metalevel requirements.
5. The data pipeline has appropriate privacy controls.
6. New features can be added quickly.
7. All input feature code is tested.

Model Development:

1. Model specs are reviewed and submitted.
2. Offline and online metrics correlate.
3. All hyperparameters have been tuned.
4. The impact of model staleness is known.
5. A simpler model is not better.
6. Model quality is sufficient on important data slices.
7. The model is tested for considerations of inclusion.

ML Infrastructure:

1. Training is reproducible.
2. Model specs are unit tested.
3. The ML pipeline is integration tested.
4. Model quality is validated before serving.
5. The model is debuggable.
6. Models are canaried before serving.
7. Serving models can be rolled back

Monitoring Tests:

1. Dependency changes result in notification.
2. Data invariants hold for inputs.
3. Training and serving are not skewed.

4. Models are not too stale.
5. Models are numerically stable.
6. Computing performance has not regressed.
7. Prediction quality has not regressed.

9.6.1 Hands-On Exercise: Exploratory Testing and Exploratory Data Analysis (EDA)

For a selected model and dataset, students will perform a data tour, considering various types of data and their distribution for various parameters.

Students will perform EDA on the data to identify missing data and/or potential bias in the data.

9.7 Selecting Test Techniques for AI-Based Systems

An AI-based system will typically include both AI and non-AI components. The selection of test techniques for testing the non-AI components is generally the same as for any conventional testing. For the AI-based components, the choice may be more constrained. For example, where a test oracle problem is perceived (i.e., generating expected results is difficult), then, based on the perceived risks, it is possible to mitigate this problem by the use of the following:

- Back-to-back testing: This requires test cases to be available or generated and an equivalent system to act as a pseudo-oracle, which for regression testing can be a previous version of the system. For effective detection of defects, an independently developed system may be required.
- A/B testing: This often uses operational inputs as test cases and is normally used to compare two variants of the same system using statistical analysis. A/B testing can be used to check for the data poisoning of a new variant, or for automated regression of a self-learning system.
- Metamorphic testing: This can be used by inexperienced testers to cost-effectively find defects although they need to understand the application domain. MT is not suitable for providing definitive results as the expected results are not absolute, but, instead, relative to the source test cases. Commercial tool support is not currently available, but many tests can be generated manually.

Adversarial testing is typically appropriate for ML models where the mishandling of adversarial examples could have a significant impact, or where the system may be attacked. Similarly, testing for data poisoning may be appropriate for ML systems where the system may be attacked.

Where the AI-based systems are complex and have multiple parameters, pairwise testing is often appropriate.

Experience-based testing is often suitable for testing AI-based systems, especially for consideration of the data used for training and operational data. EDA can be used to validate the ML algorithm being used, identify efficiency improvements, and leverage domain expertise. Google have found that their ML test checklist is an effective approach for ML systems.

In the specific area of neural networks, coverage of the network is often suitable for mission-critical systems, with some coverage criteria requiring more rigorous coverage than others.

10 Test Environments for AI-Based Systems – 30 minutes

Keywords

Virtual test environment

AI-Specific Keywords

AI-specific processor, autonomous system, big data, explainability, multi-agent system, self-learning system

Learning Objectives for Chapter 10:

10.1 Test Environments for AI-Based Systems

AI-10.1.1 K2	Describe the main factors that differentiate the test environments for AI-based systems from those required for conventional systems.

10.2 Virtual Test Environments for Testing AI-Based Systems

AI-10.2.1 K2	Describe the benefits provided by virtual test environments in the testing of AI-based systems.

10.1 Test Environments for AI-Based Systems

AI-based systems can be used in a wide variety of operational environments, which means that the test environments are similarly diverse. Characteristics of AI-based systems that can cause the test environments to differ from those for conventional systems include:

- Self-learning: Self-learning systems, and some autonomous systems, are expected to adapt to changing operational environments that may not have been fully defined when the system was initially deployed (see Section 2.1). As a result, defining test environments that can mimic these undefined environmental changes is inherently difficult and may require both imagination on the part of the testers and a level of randomness built into the test environment.
- Autonomy: Autonomous systems are expected to respond to changes in their environment without human intervention, and also recognize situations where autonomy should be ceded back to human operators (see Section 2.2). For some systems, identifying and then mimicking the circumstances for ceding autonomy may require the test environments to push the systems to extremes. For some autonomous systems, their purpose is to work in environments that are hazardous, and setting up representative, hazardous test environments can be challenging.
- Multi-agency: Where multi-agent AI-based systems are expected to work in concert with other AI-based systems, the test environment may need to incorporate a level of non-determinism so that it can mimic the non-determinism of the AI-based systems with which the SUT interacts.
- Explainability: The nature of some AI-based systems can make it difficult to determine how the system made its decisions (see Section 2.7). Where this is important to understand prior to deployment, the test environment may need to incorporate tools as a means of explaining how decisions are made.
- Hardware: Some of the hardware used to host AI-based systems is specifically designed for this purpose, such as AI-specific processors (see Section 1.6). The need to include such hardware in the test environment should be considered as part of the relevant test planning.
- Big data: Where an AI-based system is expected to consume big data (e.g., high-volume, high-velocity and/or high-variety data), then setting this up as part of a test environment needs careful planning and implementation (see Section 7.3).

10.2 Virtual Test Environments for Testing AI-Based Systems

The use of a virtual test environment when testing an AI-based system brings the following benefits:

- Dangerous scenarios: These can be tested without endangering the SUT, other interacting systems, including humans, or the operational environment (e.g., trees, buildings).
- Unusual scenarios: These can be tested when it would otherwise be very time consuming or expensive to set up these scenarios for real operations (e.g., waiting for a rare event, such as a full solar eclipse or four buses entering the same road intersection simultaneously). Similarly, edge cases, which are difficult to create in the real world, can be created more easily, repeatedly and reproducibly in a virtual test environment.
- Extreme scenarios: These can be tested when it would be expensive or impossible to set these up in reality (e.g., for a nuclear disaster or deep space exploration).

- Time-intensive scenarios: These can be tested in reduced timescales (e.g., several times per second) in a virtual environment. In contrast, these might take hours or days to set up and run in real time. A further advantage is that multiple virtual test environments can be run in parallel. This typically takes place in the cloud and allows many scenarios to be run concurrently, which may not be possible using actual system hardware.
- Observability and controllability: Virtual test environments provide far greater controllability of the test environment For example, they can ensure that an unusual set of financial trading conditions is replicated. In addition, they give far better observability, as all digitally provided parts of the environment can be continuously monitored and recorded.
- Availability: The simulation of hardware by virtual test environments allows systems to be tested with (simulated) hardware components that may otherwise not be available, perhaps because they have not been developed yet or are too expensive.

Virtual test environments may be built specifically for a given system, may be generic, or may be developed to support specific application domains. Both commercially and open-source virtual test environments are available to support the testing of AI-based systems. Examples include:

- Morse: The Modular Open Robots Simulation Engine, is a simulator for generic mobile robot simulation of single or multi robots, based on the Blender game engine [R24].
- AI Habitat: This is a simulation platform created by Facebook AI, designed to train embodied agents (such as virtual robots) in photo-realistic 3D environments [R25].
- DRIVE Constellation: This is an open and scalable platform for self-driving cars from NVIDIA. It is based on a cloud-based platform and is capable of generating billions of miles of autonomous vehicle testing [R26].
- MATLAB and Simulink: These provide the ability to prepare training data, produce ML models and simulate the execution of AI-based systems including the models using synthetic data [R27].

11 Using AI for Testing – 195 minutes

Keywords

Visual testing

AI-Specific Keywords

Bayesian techniques, classification, clustering algorithm, defect prediction, graphical user interface (GUI)

Learning Objectives for Chapter 11:

11.1 AI Technologies for Testing

AI-11.1.1	K2	Categorize the AI technologies used in software testing.
HO-11.1.1	H2	Discuss, using examples, those activities in testing where AI is less likely to be used.

11.2 Using AI to Analyze Reported Defects

AI-11.2.1	K2	Explain how AI can assist in supporting the analysis of new defects.

11.3 Using AI for Test Case Generation

AI-11.3.1	K2	Explain how AI can assist in test case generation.

11.4 Using AI for the Optimization of Regression Test Suites

AI-11.4.1	K2	Explain how AI can assist in optimization of regression test suites

11.5 Using AI for Defect Prediction

AI-11.5.1	K2	Explain how AI can assist in defect prediction.
HO-11.5.1	H2	Implement a simple AI-based defect prediction system.

11.6 Using AI for Testing User Interfaces

AI-11.6.1	K2	Explain the use of AI in testing user interfaces

11.1 AI Technologies for Testing

Several AI technologies are listed in Section1.4, all of which can be used to support some specific aspect of software testing. According to Harman [B24], the software engineering community uses three broad areas of AI technologies:

- Fuzzy logic and probabilistic methods: These involve the use of AI techniques to handle real world problems which are themselves probabilistic. For example, AI can be used to analyze and predict possible system failures using Bayesian techniques. These may estimate the likelihood of components or functions failing, or reflect the potentially random nature of human interactions with the system.
- Classification, Learning and Prediction: This can be used for various use cases such as predicting costs as part of project planning or of predicting defects. As embodied by ML, this area is used for many software testing tasks, including defect management (see Section 11.2), defect prediction (see Section 11.5), and user interface testing (see Section 11.6).
- Computational search and optimization techniques: These can be used to solve optimization problems using a computational search of potentially large and complex search spaces (e.g., using search algorithms. Examples include generating test cases (see Section 11.3), identifying the smallest number of test cases that achieves a given coverage criterion, and optimizing regression test cases (see Section 11.4).

The above categorization is necessarily broad, as there is considerable overlap between the testing tasks that can be implemented by AI and the different AI technologies. It is also just one categorization, and others could be created that may be equally valid.

11.1.1 Hands-On Exercise: The Use of AI in Testing

As part of a discussion, students will identify testing activities and tasks that are currently impractical for implementation as AI. These could include:

- specifying test oracles.
- communicating with stakeholders to clarify ambiguities and retrieve missing information.
- suggesting improvements to the user experience.
- challenging stakeholder assumptions and asking uncomfortable questions.
- understanding user needs.

A distinction should be drawn between weak AI, which could be used for some limited tasks, and general AI, which is currently not available (see Section 1.2).

11.2 Using AI to Analyze Reported Defects

Reported defects are usually categorized, prioritized, and any duplicates identified. This activity is often referred to as defect triage or analysis and is intended to optimize the elapsed time spent in defect resolution. AI can be used to support this activity in various ways, such as:

- Categorization: NLP [B25] can be used to analyze text within defect reports and extract topics, such as the area of affected functionality, that can then be provided alongside other meta data to clustering algorithms, such as k-nearest neighbors or support vector machines. These algorithms can identify suitable defect categories and highlight similar or duplicate defects. AI-based categorization is particularly useful for automated defect reporting systems (e.g., for Microsoft Windows and for Firefox) and on large projects with many software engineers.
- Criticality: ML models trained on the features of the most critical defects can be used to identify those defects most likely to cause those system failures that account for a large percentage of reported defects [B26].
- Assignment: ML models can suggest which developers are best suited to fix particular defects, based on the defect content and previous developer assignments [B26].

11.3 Using AI for Test Case Generation

The use of AI to generate tests can be a very effective technique for quickly create testing assets and maximizing coverage (e.g., code or requirements coverage). The basis for generating these tests includes the source code, the user interface, and a machine-readable test model. Some tools also base tests on the observation of the low-level behavior of the system through instrumentation or through log files [B27].

However, unless a test model that defines required behaviors is used as the basis of the tests, this form of test generation generally suffers from a test oracle problem because the AI-based tool does not know what the expected results should be for a given set of test data. One solution is to use back-to-back testing if a suitable system is available to use as a pseudo-oracle (see Section 9.3). Alternatively, tests could be run with the expected result that neither an “application not responding” nor a system crash occurred, or other similar simple failure indicators.

Research comparing AI-based test generation tools with similar non-AI fuzz testing tools shows that the AI-based tools can achieve equivalent levels of coverage and find more defects while reducing the average sequence of steps needed to cause a failure from an average of around 15,000 steps to around 100 steps. This makes debugging far easier [B27].

11.4 Using AI for the Optimization of Regression Test Suites

As changes are made to a system, new tests are created, executed and become candidates for a regression test suite. To prevent regression test suites from growing too large, they should be frequently optimized to select, prioritize and even augment test cases to create a more effective and efficient regression test suite.

An AI-based tool can perform optimization of the regression test suite by analyzing, for example, the information from previous test results, associated defects, and the latest changes that have been made, such as features which are broken more frequently and which tests exercise code impacted by recent changes.

Research shows that reductions of 50% in the size of a regression test suite can be achieved while still detecting most defects [B28], and reductions of 40% in the test execution duration can be reached without significant reduction in fault detection for continuous integration testing [B29].

11.5 Using AI for Defect Prediction

Defect prediction can be used to predict whether a defect is present, how many defects are present, or whether defects can be found. This capability depends on the sophistication of the tool used. Results are normally used to prioritize the testing (e.g., more tests for those components where more defects are predicted).

Defect prediction is typically based on source code metrics, process metrics and/or people and organizational metrics. Due to there being so many potential factors to consider, determining the relationship between these factors and the likelihood of defects is beyond human capabilities. As a result, using an AI-based approach which typically uses ML is a necessity. Defect prediction is most effective when based on prior experiences in a similar situation (e.g., with the same code base and/or the same developers).

Defect prediction using ML has been successfully used in several different situations (e.g., [B30] and [B31]). The best predictors have been found to be people and organizational measures rather than the more widely used source code metrics, such as lines of code and cyclomatic complexity [B32].

11.5.1 Hands-On Exercise: Build a Defect Prediction System

Students will use a suitable dataset (e.g., including source code measures and corresponding defect data) to build a simple defect prediction model and use it to predict the likelihood of defects using source code measures from similar code.

The model should use at least four features from the dataset, and the class should explore the results using several different features to highlight how the results change based on the selected features.

11.6 Using AI for Testing User Interfaces

11.6.1 Using AI to Test Through the Graphical User Interface (GUI)

Testing through the GUI is the typical approach for manual testing (other than for component testing) and is often the starting point for test automation initiatives. The resultant tests emulate human interaction with the test object. This scripted test automation can be implemented by applying a capture/playback approach, using either the actual coordinates of the user interface elements, or the software-defined objects/widgets of the interface. However, this approach suffers several drawbacks with object identification, including sensitivity to interface changes, code changes, and platform changes.

AI can be used to reduce the brittleness of this approach, by employing AI-based tools to identify the correct objects using various criteria (e.g., XPath, label, id, class, X/Y coordinates), and to choose the historically most stable identification criteria. For example, the ID of a button in a particular area of the application may change with each release, and so the AI-based tool may assign a lower importance to this ID over time and place more reliance on other criteria. This approach classifies the objects in the user interface as matching the test, or not matching the test.

Alternatively, visual testing uses image recognition to interact with GUI objects through the same interface as an actual user, and therefore does not need to access the underlying code and interface definitions. This makes it completely non-intrusive and independent of the underlying technology. The scripts need only work through the visible user interface. This approach allows the tester to create scripts that interact directly with the images, buttons and text fields on the screen in the same way as a human user, without being affected by the overall screen layout. The use of image recognition in test automation can become restricted by the computing resources needed. However, the availability of affordable AI that supports sophisticated image recognition now makes this approach possible for mainstream use.

11.6.2 Using AI to Test the GUI

ML models can be used to determine the acceptability of user interface screens (e.g., by using heuristics and supervised learning). Tools based on these models can identify incorrectly rendered elements, determine whether some objects are inaccessible or hard to detect, and detect various other issues with the visual appearance of the GUI.

While image recognition is one form of computer vision algorithm, other forms of AI-based computer vision can be used to compare images (e.g., screenshots) to identify unintended changes to the layout, the size, position, color, font or other visible attributes of objects. The results of these comparisons can be used to support regression testing to check that changes to the test object have not adversely affected the user interface.

The technology for checking the acceptability of screens can be combined with comparison tools to create more sophisticated AI-based regression testing tools that are capable of advising whether detected user interface changes are likely to be acceptable to users, or whether these changes should be flagged for checking by a human. Such AI-based tools can also be used to support testing for compatibility on different browsers, devices or platforms aimed at checking that the user interface for the same application works correctly on various browsers/devices/platforms.

12 References

12.1 Standards

[S01] ISO/IEC TR 29119-11:2020, Software and systems engineering — Software testing — Part 11 Guidelines on the testing of AI-based systems

[S02] DIN SPEC 92001-1, Artificial Intelligence - Life Cycle Processes and Quality Requirements - Part 1: Quality Meta Model, https://www.din.de/en/wdc-beuth:din21:303650673 (accessed May 2021).

[S03] DIN SPEC 92001-2, Artificial Intelligence - Life Cycle Processes and Quality Requirements - Part 2: Technical and Organizational Requirements, https://www.din.de/en/innovation-and-research/din-spec-en/projects/wdc-proj:din21:298702628 (accessed May 2021).

[S04] ISO 26262 - https://www.iso.org/standard/68383.html (accessed May 2021)

[S05] ISO/PAS 21448:2019, Road vehicles – Safety of the intended functionality (SOTIF) - https://www.iso.org/standard/70939.html (accessed May 2021)

[S06] ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, 2011.

[S07] ISO 26262-6:2018 - Road vehicles - Functional safety - Part 6: Product development at the software level.

[S08] ISO/IEC/IEEE 29119-4:2015, Software and systems engineering — Software testing — Part 4: Test techniques.

12.2 ISTQB® Documents

[I01] ISTQB® Certified Tester Foundation Level Syllabus, Version 2018 V3.1 https://www.istqb.org/downloads/category/2-foundation-level-documents.html (accessed May 2021).

[I02] ISTQB® Certified Tester Advanced Level Test Analyst Syllabus, Version 3.1, section 3.2.6 https://www.istqb.org/downloads/category/75-advanced-level-test-analyst-v3-1.html (accessed August 2021).

[I03] ISTQB® Certified Tester AI Testing, Overview of Syllabus, Version 1.0

12.3 Books and Articles

[B01] Cadwalladr, Carole (2014). "Are the robots about to rise? Google's new director of engineering thinks so…" The Guardian. Guardian News and Media Limited., https://www.theguardian.com/technology/2014/feb/22/robots-google-ray-kurzweil-terminator-singularity-artificial-intelligence (accessed May 2021).

[B02] Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, Pearson, 2020.

[B03] M. Davies et al., “Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook,” Proceedings of the IEEE, vol. 109, no. 5, pp. 911–934, May 2021, doi: 10.1109/JPROC.2021.3067593.

[B04] Chris Wiltz, Can Apple Use Its Latest AI Chip for More Than Photos?, Electronics & Test, Artificial Intelligence, https://www.designnews.com/electronics-test/can-apple-use-its-latest-ai-chip-more-photos/153617253461497 (accessed May 2021).

[B05] HUAWEI Reveals the Future of Mobile AI at IFA 2017, Huawei Press Release, https://consumer.huawei.com/en/press/news/2017/ifa2017-kirin970/ (accessed May 2021).

[B06] REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), April 2016,https://eur-lex.europa.eu/eli/reg/2016/679/oj (accessed May 2021)

[B07] Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles J3016_201806, SAE, ,https://www.sae.org/standards/content/j3016_201806/ (accessed May 2021).

[B08] G20 Ministerial Statement on Trade and Digital Economy: Annex. Available from: https://www.mofa.go.jp/files/000486596.pdf (accessed May 2021).

[B09] Concrete Problems in AI Safety, Dario Amodei (Google Brain), Chris Olah (Google Brain), Jacob Steinhardt (Stanford University), Paul Christiano (UC Berkeley), John Schulman (OpenAI), Dan Man´e (Google Brain), March 2016. https://arxiv.org/pdf/1606.06565 (accessed May 2021).

[B10] Explainable AI: the basics, Policy briefing, Issued: November 2019 DES6051, ISBN: 978-1-78252-433-5, The Royal Society.

[B11] The Ultimate Guide to Data Labeling for Machine Learning, www.cloudfactory.com/data-labeling-guide (accessed May 2021).

[B12] Pei et al, DeepXplore: Automated Whitebox Testing of Deep Learning Systems, Proceedings of ACM Symposium on Operating Systems Principles (SOSP '17), Jan 2017.

[B13] Sun et al, Testing Deep Neural Networks, https://www.researchgate.net/publication/323747173_Testing_Deep_Neural_Networks, accessed Nov 2019 (accessed May 2021).

[B14] A. Odena and I. Goodfellow, TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing, ArXiv e-prints, Jul. 2018, https://arxiv.org/pdf/1807.10875 (accessed May 2021)

[B15] Riccio, V. et al, Testing Machine Learning based Systems: A Systematic Mapping. Empirical Software Engineering, https://link.springer.com/article/10.1007/s10664-020-09881-0 (accessed May 2021)

[B16] Baudel, Thomas et al, Addressing Cognitive Biases in Augmented Business Decision Systems., https://arxiv.org/abs/2009.08127 (accessed May 2021)

[B17] Papernot, N. et al, Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, arXiv preprint arXiv:1605.07277, 2016. https://arxiv.org/pdf/1605.07277 (accessed May 2021).

[B18] Chen et al, Metamorphic Testing: A Review of Challenges and Opportunities, ACM Comput. Surv. 51, 1, Article 4, January 2018. https://www.researchgate.net/publication/322261865_Metamorphic_Testing_A_Review_of_Challenges_and_Opportunities (accessed May 2021).

[B19] Huai Liu, Fei-Ching Kuo, Dave Towey, and Tsong Yueh Chen., How effectively does metamorphic testing alleviate the oracle problem?, IEEE Transactions on Software Engineering 40, 1, 4–22, 2014

[B20] James Whittaker, Exploratory Software Testing: Tips, Tricks, Tours and Techniques to Guide Test Design, 1. Edition, Addison-Wesley Professional, 2009.

[B21] L. Wilkinson, A. Anand, and R. Grossman. High-dimensional visual analytics: Interactive exploration guided by pairwise views of point distributions. Visualization and Computer Graphics, IEEE Transactions on, 12(6):1363–1372, 2006, https://www.cs.uic.edu/~wilkinson/Publications/sorting.pdf (accessed May 2021).

[B22] Ryan Hafen and Terence Critchlow, EDA and ML – A Perfect Pair for Large-Scale Data Analysis, IEEE 27th International Symposium on Parallel and Distributed Processing, 2013, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6651091 (accessed May 2021).

[B23] Breck, Eric, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley., The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction, IEEE International Conference on Big Data (Big Data), 2017, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8258038 (accessed May 2021).

[B24] Harman, The Role of Artificial Intelligence in Software Engineering, In First International Workshop on Realizing AI Synergies in Software Engineering (RAISE), pp. 1-6. IEEE, June 2012, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6227961 (accessed May 2021).

[B25] Nilambri et al, A Survey on Automated Duplicate Detection in a Bug Repository, International Journal of Engineering Research & Technology (IJERT), 2014, https://www.ijert.org/research/a-survey-on-automated-duplicate-detection-in-a-bug-repository-IJERTV3IS041769 (accessed May 2021)

[B26] Kim, D.; Wang, X.; Kim, S.; Zeller, A.; Cheung, S.C.; Park, S. (2011). "Which Crashes Should I Fix First? Predicting Top Crashes at an Early Stage to Prioritize Debugging Efforts," in the IEEE Transactions on Software Engineering, volume 37, https://ieeexplore.ieee.org/document/5711013 (accessed May 2021).

[B27] Mao et al, Sapienz: multi-objective automated testing for Android applications, Proceedings of the 25th International Symposium on Software Testing and Analysis, July 2016, http://www0.cs.ucl.ac.uk/staff/K.Mao/archive/p_issta16_sapienz.pdf (accessed May 2021).

[B28] Rai et al, Regression Test Case Optimization Using Honey Bee Mating Optimization Algorithm with Fuzzy Rule Base, World Applied Sciences Journal 31 (4): 654-662, 2014, https://www.researchgate.net/publication/336133351_Regression_Test_Case_Optimization_Using_Honey_Bee_Mating_Optimization_Algorithm_with_Fuzzy_Rule_Base (accessed May 2021).

[B29] Dusica Marijan, Arnaud Gotlieb, Marius Liaaen. A learning algorithm for optimizing continuous integration development and testing practice, Journal of Software : Practice and Experience, Nov 2018.

[B30] Tosun et al, AI-Based Software Defect Predictors: Applications and Benefits in a Case Study, Proceedings of the Twenty-Second Innovative Applications of Artificial Intelligence Conference (IAAI-10), 2010.

[B31] Kim et al, Predicting Faults from Cached History, 29th International Conference on Software Engineering (ICSE'07), 2007.

[B32] Nagappan 2008 Nagappan et al, The Influence of Organizational Structure on Software Quality: An Empirical Case Study, Proceedings of the 30th international conference on Software engineering (ICSE'08), May 2008.

[B33] Kuhn et al, Software Fault Interactions and Implications for Software Testing, IEEE Transactions on Software Engineering vol. 30, no. 6, (June 2004) pp. 418-421.

12.4 Other References

The following references point to information available on the Internet. Even though these references were checked at the time of publication, the ISTQB® cannot be held responsible if the references are no longer available.

[R01] Wikipedia contributors, "AI effect," Wikipedia, https://en.wikipedia.org/wiki/AI_effect (accessed May 2021).

[R02] https://mxnet.apache.org/ (accessed May 2021).

[R03] https://docs.microsoft.com/en-us/cognitive-toolkit/ (accessed May 2021).

[R04] IBM Watson, https://www.ibm.com/watson/ai-services

[R05] https://www.tensorflow.org/ (accessed May 2021).

[R06] https://keras.io/ (accessed May 2021).

[R07] https://pytorch.org/ (accessed May 2021).

[R08] https://scikit-learn.org/stable/whats_new/v0.23.html (accessed May 2021).

[R09] NVIDIA VOLTA, https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/ (accessed May 2021).

[R10] Cloud TPU, https://cloud.google.com/tpu/ (accessed May 2021).

[R11] Edge TPU, https://cloud.google.com/edge-tpu/ (accessed May 2021).

[R12] Intel® Nervana™ Neural Network processors deliver the scale and efficiency demanded by deep learning model evolution, https://www.intel.ai/nervana-nnp/ (accessed May 2021).

[R13] The Evolution of EyeQ, https://www.mobileye.com/our-technology/evolution-eyeq-chip/ (accessed May 2021).

[R14] ImageNet - http://www.image-net.org/ (accessed May 2021).

[R15] Google’s BERT - https://github.com/google-research/bert (accessed May 2021).

[R16] https://www.kaggle.com/datasets (accessed May 2021).

[R17] https://www.kaggle.com/paultimothymooney/2018-kaggle-machine-learning-data-science-survey (accessed May 2021).

[R18] MLCommons - https://mlcommons.org/ (accessed May 2021).

[R19] DAWNBench – https://dawn.cs.stanford.edu/benchmark (accessed May 2021).

[R20] MLMark – https://www.eembc.org/mlmark (accessed May 2021).

[R21] https://digital-strategy.ec.europa.eu/en/library/assessment-list-trustworthy-artificial-intelligence-altai-self-assessment| Shaping Europe’s digital future (europa.eu) (accessed August 2021)

[R22] https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai (accessed August 2021)

[R23] Google GraphicsFuzz, https://github.com/google/graphicsfuzz (accessed May 2021).

[R24] http://www.openrobots.org/morse/doc/0.2.1/morse.html (accessed May 2021).

[R25] https://ai.facebook.com/blog/open-sourcing-ai-habitat-a-simulation-platform-for-embodied-ai-research/ (accessed May 2021).

[R26] https://www.nvidia.com/en-gb/self-driving-cars/drive-constellation/ (accessed May 2021).

[R27] https://uk.mathworks.com/discovery/artificial-intelligence.html#ai-with-matlab (accessed May 2021).

13 Appendix A – Abbreviations

Abbreviation	Description
AI	Artificial intelligence
AIaaS	AI as a service
API	Application programming interface
AUC	Area under curve
DL	Deep learning
DNN	Deep neural network
EDA	Exploratory data analysis
EU	European Union
FN	False negative
FP	False positive
GDPR	General data protection regulation
GPU	Graphical processing unit
GUI	Graphical user interface
LIME	Local interpretable model-agonistic explanations
MC/DC	Modified condition decision coverage
ML	Machine learning
MR	Metamorphic relation
MSE	Mean square error
MT	Metamorphic testing
NLP	Natural language processing
ROC	Receiver operating characteristic
SUT	System under test
SVM	Support vector machine
TN	True negative
TP	True positive
XAI	Explainable AI

14 Appendix B – AI Specific and other Terms

Term Name	Definition
accuracy	The ML functional performance metric used to evaluate a classifier, which measures the proportion of predictions that were correct (After ISO/IEC TR 29119-11)
activation function	The formula associated with a neuron in a neural network that determines the output of the neuron from the inputs to the neuron
activation value	The output of an activation function of a neuron in a neural network
adversarial attack	The deliberate use of adversarial examples to cause an ML model to fail
AI as a Service (AIaaS)	A software licensing and delivery model in which AI and AI development services are centrally hosted
AI component	A component that provides AI functionality
AI effect	The situation when a previously labelled AI system is no longer considered to be AI as technology advances (ISO/IEC TR 29119-11)
AI-based system	A system that integrates one or more AI components
AI-specific processor	A type of specialized hardware designed to accelerate AI applications
algorithmic bias	A type of bias caused by the ML algorithm
annotation	The activity of identifying objects in images with bounding boxes to provide labelled data for classification
area under curve (AUC)	A measure of how well a classifier can distinguish between two classes.
artificial intelligence (AI)	The capability of an engineered system to acquire, process, create and apply knowledge and skills (ISO/IEC TR 29119-11)
association	An unsupervised learning technique that identifies relationships and dependencies between samples
augmentation	The activity of creating new data points based on an existing dataset

Term Name	Definition
automation bias	A type of bias caused by a person favoring the recommendations of an automated decision-making system over other sources Synonym: complacency bias
autonomous system	A system capable of working without human intervention for sustained periods
autonomy	The ability of a system to work for sustained periods without human intervention (ISO/IEC TR 29119-11)
Bayesian model	A statistical model that uses probability to represent the uncertainty of both model inputs and outputs
Bayesian technique	A technique that considers before and after probability distributions as parameters of a statistical model
bias	The systematic difference in treatment of certain objects, people or groups in comparison to others (ISO/IEC DIS 22989)
big data	Extensive datasets whose characteristics in terms of volume, variety, velocity and/or variability require specialized technologies and techniques to process
case-based reasoning	The technique of solving a new problem based on the solutions of similar past problems
chatbot	An application used to conduct a conversation via text or text-to-speech
classification	A type of ML function that predicts the output class for a given input (After ISO/IEC TR 29119-11)
classifier	An ML model used for classification Synonym: classification model
clustering	A type of ML function that groups similar data points together
clustering algorithm	A type of ML algorithm used to group similar objects into clusters

Term Name	Definition
concept drift	A change in the perceived accuracy of an ML model predictions over time caused by changes in user expectations, behavior and the operational environment.
confusion matrix	A technique for summarizing the ML functional performance of a classification algorithm
data acquisition	The activity of acquiring data relevant to the business problem to be solved by an ML model
data labelling	The activity of adding meaningful tags to objects in raw data to support classification in ML
data pipeline	The implementation of data preparation activities to provide input data to support training by an ML algorithm or prediction by an ML model
data point	A set of one or more measurements comprising a single observation used as part of a dataset
data poisoning	The deliberate and malicious manipulation of training or input data to an ML model
data preparation	The activities of data acquisition, data pre-processing and feature engineering in the ML workflow
data pre-processing	The activities of data cleaning, data transformation, data augmentation, and data sampling in the ML workflow
data visualization	A technique for graphically representing data relationships, trends and patterns
dataset	A collection of data used for training, evaluation, testing and prediction in ML
decision threshold	A value that transforms the result of a prediction function into a binary outcome of either above or below the value Synonym: discrimination threshold
decision tree	A tree-like ML model whose nodes represent decisions, and whose branches represent possible outcomes

Term Name	Definition
deductive classifier	A classifier based on the application of inference and logic to input data
deep learning (DL)	ML using neural networks with multiple layers
deep neural network	A neural network comprised of several layers of neurons Synonym: multi-layer perceptron
defect prediction	A technique to predict the areas within the test object in which defects will occur or the quantity of defects that are present
deterministic system	A system which will produce the same set of outputs and final state from a given set of inputs and starting state
edge computing	The part of a distributed architecture in which information processing is performed close to where that information is used.
epoch	An iteration of ML training on the whole training dataset
evolution	The process of continuous change from a lower, simpler, or worse state to a higher, more complex, or better state
expert system	An AI-based system for solving problems in a particular domain or application area by drawing inferences from a knowledge base developed from human expertise
explainability	The level of understanding how the AI-based system came up with a given result (ISO/IEC TR 29119-11)
explainable AI (XAI)	The field of study related to understanding the factors that influence AI system outputs
exploratory data analysis (EDA)	The interactive, hypothesis-driven and visual exploration of data used to support feature engineering
F1-Score	An ML functional performance metric used to evaluate a classifier which provides a balance between recall and precision
false negative (FN)	An ML model prediction in which the model mistakenly predicts the negative class

Term Name	Definition
false positive (FP)	An ML model prediction in which the model mistakenly predicts the positive class
feature	An individual measurable attribute of the input data used for training by an ML algorithm and for prediction by an ML model
feature engineering	The activity in which those attributes in the raw data that best represent the underlying relationships that should appear in the ML model are identified for use in the training data (ISO/IEC TR 29119-11)
flexibility	The ability of a system to work in contexts outside its initial specification (After ISO/IEC TR 29119-11)
fuzzy logic	A type of logic based on the concept of partial truth represented by certainty factors between 0 and 1
general AI	AI that exhibits intelligent behaviour comparable to a human across the full range of cognitive abilities (ISO/IEC TR 29119-11) Synonym: strong AI
General Data Protection Regulation (GDPR)	The European Union (EU) regulation on data protection and privacy that applies to the data of citizens of the EU and the European Economic Area
graphical processing unit (GPU)	An 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
ground truth	The information provided by direct observation and measurement that is known to be real or true
hyperparameter	A parameter used to either control the training of an ML model or to set the configuration of an ML model
hyperparameter tuning	The activity of determining the optimal hyperparameters based on particular goals
inappropriate bias	A type of bias that causes a system to produce results that lead to adverse effects for a particular group

Term Name	Definition
intelligent agent	An autonomous program which directs its activity towards achieving goals using observations and actions
inter-cluster metric	A metric that measures the similarity of data points in different clusters
interpretability	The level of understanding how the underlying AI technology works (ISO/IEC TR 29119-11)
intra-cluster metric	A metric that measures the similarity of data points within a cluster
k-nearest neighbor	An approach to classification that estimates the likelihood of group membership for a data point dependent on the group membership of the data points nearest to it
learning algorithm	A program that produces an ML model based on the properties of the training dataset
LIME method	The Local Interpretable Model-Agnostic Explanations program for explaining the predictions from an ML model
linear regression	A statistical technique that models the relationship between variables by fitting a linear equation to the observed data when the target variable is numeric
logistic regression	A statistical technique that models the relationship between variables when the target variable is categorical rather than numeric
machine learning (ML)	The process using computational techniques to enable systems to learn from data or experience (ISO/IEC TR 29119-11)
mean square error (MSE)	The statistical measure of the average squared difference between the estimated values and the actual value
ML algorithm	An algorithm used to create an ML model from a training dataset
ML benchmark suite	A dataset used to compare ML models and ML algorithms over a range of evaluation metrics
ML framework	A tool or library that supports the creation of an ML model

Term Name	Definition
ML function	Functionality implemented by an ML model, such as classification, regression or clustering
ML model evaluation	The process of comparing achieved ML functional performance metrics with required criteria and those of other ML models
ML model training	The process of applying the ML algorithm to the training dataset to create an ML model
ML model tuning	The process of testing hyperparameters to achieve optimum performance
ML system	A system that integrates one or more ML models
ML workflow	A sequence of activities used to manage the development and deployment of an ML model
multi-agent system	A system that comprises multiple intelligent agents
narrow AI	AI focused on a single well-defined task to address a specific problem (ISO/IEC TR 29119-11) Synonym: weak AI
natural language processing (NLP)	A field of computing that provides the ability to read, understand, and derive meaning from natural languages
neural network	A 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 (ISO/IEC 2382) Synonym: artificial neural network
neural network trojan	A vulnerability injected into a neural network using a data poisoning attack with the intent of exploiting it later
neuromorphic processor	An integrated circuit designed to mimic the biological neurons of the human brain

Term Name	Definition
neuron	A node in a neural network, usually receiving multiple input values and generating an activation value
noise	A distortion or corruption in data
non-deterministic system	A system which will not always produce the same set of outputs and final state given a particular set of inputs and starting state
outlier	An observation that lies outside the overall pattern of the data distribution
overfitting	The generation of an ML model that corresponds too closely to the training dataset, resulting in a model that finds it difficult to generalize to new data (After ISO/IEC TR 29119-11)
perceptron	A neural network with just one layer and one neuron
precision	An ML functional performance metric used to evaluate a classifier, which measures the proportion of predicted positives that were correct (After ISO/IEC TR 29119-11)
pre-trained model	An ML model already trained when it was obtained
probabilistic system	A system whose behavior is described in terms of probabilities; hence its outputs cannot be perfectly predicted
procedural reasoning	AI technology used for constructing real-time reasoning systems that can perform complex tasks in dynamic environments
random forest	Ensemble ML technology for classification, regression and other tasks that operate by constructing and running many decision trees and then either outputting the mode of the class or the mean prediction of the individual trees
reasoning technique	AI that generates conclusions from available information using logical techniques (After ISO/IEC TR 29119-11)
recall	An ML functional performance metric used to evaluate a classifier, which measures the proportion of actual positives that were predicted correctly (After ISO/IEC TR 29119-11) Synonym: sensitivity

Term Name	Definition
receiver operating characteristic (ROC) curve	A graphical plot that illustrates the ability of a binary classifier as its discrimination threshold is varied
regression	A type of ML function that results in a numerical or continuous output value for a given input (After ISO/IEC TR 29119-11)
regression model	An ML model whose expected output for a given numeric input is a continuous variable (After ISO/IEC DIS 23053)
reinforcement learning	The activity of building an ML model using a process of trial and reward to achieve an objective (After ISO/IEC TR 29119-11)
reward function	A function that defines the success of reinforcement learning
reward hacking	The activity performed by an intelligent agent to maximize its reward function to the detriment of meeting the original objective (After ISO/IEC TR 29119-11)
R-squared	A statistical measure of how close the data points are to the fitted regression line. Synonym: coefficient of determination
rule engine	A set of rules that determine which actions should occur when certain conditions are satisfied
safety	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)
sample bias	A type of bias where the dataset is not fully representative of the data space to which ML is applied
search algorithm	An algorithm that systematically visits a subset of all possible states or structures until the goal state or structure is reached (After ISO/IEC TR 29119-11)
self-learning system	An adaptive system that changes its behavior based on learning through trial and error (After ISO/IEC TR 29119-11)
silhouette coefficient	A clustering measure between -1 and +1 based on the average inter-cluster and intra-cluster differences

Term Name	Definition
	Synonym: silhouette score
super AI	An artificial intelligence-based system that far exceeds human capabilities
supervised learning	Training an ML model from input data and its corresponding labels
support vector machine (SVM)	An ML technique in which the data points are viewed as vectors in multi-dimensional space separated by a hyperplane
technological singularity	A point in the future when technological advances are no longer controllable by people (After ISO/IEC TR 29119-11)
test oracle problem	The challenge of determining whether a test has passed or failed for a given set of test inputs and state
training dataset	A dataset used to train an ML model
transfer learning	A technique for modifying a pre-trained ML model to perform a different related task
transparency	The level of visibility of the algorithm and data used by the AI-based system (After ISO/IEC TR 29119-11)
true negative (TN)	A prediction in which the model correctly predicts the negative class
true positive (TP)	A prediction in which the model correctly predicts the positive class
underfitting	The generation of an ML model that does not reflect the underlying trend of the training dataset, resulting in a model that finds it difficult to make accurate predictions (ISO/IEC TR 29119-11)
unsupervised learning	Training an ML model from input data using an unlabeled dataset
validation dataset	A dataset used to evaluate a trained ML model with the purpose of tuning the model
von Neumann architecture	A computer architecture which consists of five main components: memory, a central processing unit, a control unit, input and output

Term Name	**Definition**
weight	An internal variable of a connection between neurons in a neural network that affects how it computes its outputs and that changes as the neural network is trained

15 Index