

Certified Tester AI Testing Syllabus

Version 2.0

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

Copyright Notice © International Software Testing Qualifications Board (hereinafter called ISTQB®)

ISTQB® is a registered trademark of the International Software Testing Qualifications Board.

Copyright © 2026, the authors Klaudia Dussa-Zieger (chair), Stuart Reid, Vipul Kocher, Qin Liu, Jarosław Hryszko, Kyle Alexander Siemens, and Werner Henschelchen.

Copyright © 2021, the authors Klaudia Dussa-Zieger (chair), Vipul Kocher, Qin Liu, Stuart Reid, Kyle Alexander Siemens, Werner Henschelchen, and Adam Leon Smith.

All rights reserved. The authors hereby transfer the copyright to the ISTQB®. The authors (as current copyright holders) and ISTQB® (as the future copyright holder) have agreed to the following conditions of use:

- Extracts, for non-commercial use, from this document may be copied if the source is acknowledged. Any Accredited Training Provider may use this syllabus as the basis for a training course if the authors and the ISTQB® are acknowledged as the source and copyright owners of the syllabus, and provided that any advertisement of such a training course may mention the syllabus only after official Accreditation of the training materials has been received from an ISTQB®-recognized Member Board.
- Any individual or group of individuals may use this syllabus as the basis for articles and books, if the authors and the ISTQB® are acknowledged as the source and copyright owners of the syllabus.
- Any other use of this syllabus is prohibited without first obtaining the approval in writing of the ISTQB®.
- Any ISTQB®-recognized Member Board may translate this syllabus, provided they reproduce the above-mentioned Copyright Notice in the translated version of the syllabus.

Revision History

Version	Date	Remarks
1.0	2021/10/01	Release for GA
2.0 Alpha	2025/08/15	Alpha review
2.0 Beta	2026/01/07	Beta review
2.0	2026/04/17	Release for GA

Table of Contents

Copyright Notice.....	2
Revision History	3
Table of Contents	4
Acknowledgements	8
0 Introduction.....	9
0.1 Purpose of this Syllabus	9
0.2 The Certified Tester AI Testing.....	9
0.3 Career Path for Testers	9
0.4 Business Outcomes	9
0.5 Learning Objectives, Hands-on Objectives and Cognitive Level of Knowledge.....	10
0.6 The Certified Tester AI Testing Certificate Exam	10
0.7 Accreditation	11
0.8 Handling of Standards	11
0.9 Level of Detail	11
0.10 How this Syllabus is Organized	12
1 Introduction to Artificial Intelligence – 120 minutes	14
1.1 Introduction to AI.....	15
1.1.1 AI-Based and Conventional Systems.....	15
1.1.2 Narrow AI, General AI, and Super AI	15
1.1.3 Different Types of AI Technologies	16
1.1.4 Generative AI.....	17
1.1.5 Hardware for Machine Learning Systems	18
1.1.6 Development and Hosting of AI Models.....	18
1.1.7 Machine Learning Development Frameworks.....	19
1.1.8 Regulations and Standards for AI	20
2 Quality Characteristics for AI-Based Systems – 45 minutes.....	21
2.1 Quality Characteristics for AI-Based Systems.....	22
2.1.1 AI-Specific Quality Characteristics	22
2.1.2 AI and Safety	23
2.2 Acceptance Criteria for AI-Based Systems	23
2.2.1 Acceptance Criteria for AI-Based Systems	24

3	Machine Learning – 375 minutes	26
3.1	Introduction to Machine Learning	27
3.1.1	Different Forms of Machine Learning	27
3.1.2	Machine Learning Workflow	28
3.1.3	Hands-on Exercise: Create a Machine Learning Model	30
3.1.4	Pretrained Models, Fine-Tuning, and Retrieval-Augmented Generation	30
3.2	Data for Machine Learning	31
3.2.1	Activities in Data Preparation	31
3.2.2	Hands-on Exercise: Data Preparation in Support of the Creation of a Machine Learning Model	33
3.3	ML Functional Performance Metrics for Classification	33
3.3.1	Calculation of Machine Learning Functional Performance Metrics	33
3.3.2	Hands-on Exercise: Evaluate a Machine Learning Model using Selected ML Functional Performance Metrics	34
3.3.3	Hands-on Exercise: Show the Impact of Different Machine Learning Models and Dataset Combinations.....	35
3.4	Neural Networks	35
3.4.1	Structure and Working of a Deep Neural Network	36
3.4.2	Hands-on Exercise: Experience the Implementation of a Perceptron	37
3.4.3	Coverage Measures for Neural Networks	37
4	Testing AI-Based Systems – 195 minutes	38
4.1	Introduction to Testing AI-Based Systems	39
4.1.1	Locked and Adaptive AI-Based Systems	39
4.1.2	Rationale for a Statistical Approach to Testing AI-Based Systems	40
4.1.3	Test Oracles for AI-Based Systems	40
4.2	Testing Generative AI and Large Language Models.....	41
4.2.1	Testing Generative AI.....	41
4.2.2	Red Teaming.....	42
4.2.3	Hands-on Exercise: Exploratory Testing of a Large Language Model.....	43
4.3	Test Levels and Machine Learning Systems.....	43
4.3.1	Test Levels for Machine Learning Systems	43
4.3.2	Risk-Based Testing of Machine Learning Systems.....	44
5	Input Data Testing for Machine Learning Systems – 180 minutes	46
5.1	Input Data Testing for Machine Learning Systems	47

5.1.1	Input Data Risks and Mitigations.....	47
5.1.2	Testing for Bias.....	48
5.1.3	Data Pipeline Testing	49
5.1.4	Testing for Data Representativeness	50
5.1.5	Dataset Constraint Testing	51
5.1.6	Label Correctness Testing	52
5.1.7	Hands-on Exercise: Input Data Testing	53
6	Model Testing for Machine Learning Systems – 225 minutes	54
6.1	Model Testing for Machine Learning Systems	55
6.1.1	Machine Learning Model Risks and Mitigations.....	55
6.1.2	Machine Learning Model Documentation and Review.....	56
6.1.3	ML Functional Performance Testing of Probabilistic Machine Learning Systems	57
6.1.4	Adversarial Testing of Machine Learning Systems	58
6.1.5	Metamorphic Testing.....	59
6.1.6	Hands-on Exercise: Apply Metamorphic Testing	60
6.1.7	Drift Testing	60
6.1.8	Testing for Overfitting and Underfitting	60
6.1.9	A/B Testing.....	61
6.1.10	Back-to-Back Testing	62
7	Machine Learning Development Testing – 30 minutes.....	63
7.1	Machine Learning Development Testing.....	64
7.1.1	Machine Learning Development Risks and Mitigations	64
7.1.2	Machine Learning System Deployment Testing.....	65
8	List of Abbreviations	67
9	AI-Specific Terms	69
10	References	78
10.1	Standards.....	78
10.2	ISTQB® Documents	78
10.3	Glossary References	78
10.4	Books, Articles and Web Pages	79
11	Trademarks	80
12	Appendix A – Learning Objectives/Cognitive Level of Knowledge	81
	Level 1: Remember (K1).....	81

Level 2: Understand (K2).....	81
Level 3: Apply (K3)	82
Level 4: Analyze (K4).....	82
13 Appendix B – Business Outcomes Traceability Matrix with Learning Objectives	83
14 Appendix C – Release Notes	91
15 Index.....	92

Acknowledgements

The General Assembly of the ISTQB® formally released this document on April 17th, 2026.

ISTQB taskforce AI (v2.0): Klaudia Dussa-Zieger (chair), Stuart Reid, Vipul Kocher, Qin Liu, Jaroslaw Hryszko, Kaye Alexander Siemens, and Werner Henschelchen

The following persons participated in reviewing and commenting on the syllabus: Marina Abratis, Tom Adams, Laura Albert, Abhishek Alladi, Menno van den Berg, Earl Burba, Simeone Chiumarulo, Marco Ciarlito, Alessandro Collino, Jean-Baptiste Crouigneau, Yara Dalgamoni, Taz Daughtrey, Wim Decoutere, Dmitrii Degtiarenko, Iuliia Emelianova, Lozana Enbah, Tamás Gergely, David Hendrickx, David Janota, Sagar Joshi, Norbert Juhász, Willem Keesman, John Kurowski, Ine Lutterman, Niranjana Maharajh, Rik Marselis, Judy McKay, Gary Mogyorodi, Markus Niehammer, Tauhida Parveen, Arnd Pehl, Lukas Piska, Daniel Polan, Andrew Pollner, Nishan Portoyan, Meile Posthuma, Miroslav Renda, Randall Rice, Piet de Roo, Nicola de Rosa, Mark Rutz, Salvatore Sarno, Klaus Skafte, Giancarlo Tomasig, Yaron Tsubery, Rahul Verma, André Verschelling, Linda Vreeswijk, Mario Winter

ISTQB taskforce AI (v1.0): Klaudia Dussa-Zieger (chair), Vipul Kocher, Qin Liu, Stuart Reid, Adam Leon Smith, Kyle Alexander Siemens, and Werner Henschelchen

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 syllabus development, and the Member Boards for their suggestions and input.

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:

- To member boards, to translate into their local language and to accredit training providers. Member boards may adapt the syllabus to their specific language needs and modify the references to align with their local publications.
- To certification bodies, to derive examination questions in their local language, adapted to the learning objectives for this syllabus.
- To training providers, to produce courseware and determine appropriate teaching methods.
- To certification candidates, to prepare for the certification exam (either as part of a training course or independently).
- 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 (CT-AI) is designed for individuals involved in testing AI-based systems. This includes individuals in various roles, such as testers, test analysts, data analysts, test engineers, test consultants, test managers, user acceptance testers, and software developers. This certification is also suitable for individuals seeking a fundamental understanding of testing AI-based systems, including project managers, quality managers, software development managers, business analysts, operations team members, IT directors, and management consultants.

0.3 Career Path for Testers

The ISTQB® scheme provides support for testing professionals at all stages of their careers. Individuals who achieve the ISTQB® Certified Tester Foundation Level certification may also be interested in the Core Advanced Levels (Test Analyst, Technical Test Analyst, and Test Manager) and, thereafter, the Expert Level (Test Management or Improving the Test Process). The Specialist stream offers a deep dive into specific test approaches and activities, e.g., Agile Testing, Test Automation, AI Testing, Testing with Generative AI, or Mobile App Testing, or into group testing know-how for certain industry domains, e.g., Automotive or Gaming. Please visit www.istqb.org for the latest information on ISTQB's Certified Tester Scheme.

0.4 Business Outcomes

This section lists the Business Outcomes expected of a candidate who has achieved the CT-AI certification.

A Certified Tester in AI Testing can:

BO1	Understand the current state of AI, including generative AI.
BO2	Experience the implementation and testing of machine learning models.
BO3	Understand the working and testing of simple neural networks.
BO4	Understand the specific AI quality characteristics defined by ISO/IEC 25059.
BO5	Calculate and interpret ML functional performance metrics for machine learning models.
BO6	Recognize the scope and importance of the two test levels that are specific to the testing of machine learning systems.
BO7	Contribute to the development of an effective test strategy for a machine learning system.
BO8	Design and execute test cases for machine learning systems.

0.5 Learning Objectives, Hands-on Objectives and Cognitive Level of Knowledge

Learning Objectives (LO) support the business outcomes and are used to create the CT-AI exams. 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

Further details and examples of learning objectives are given in Appendix A.

For all terms listed as keywords just below chapter headings, the correct name and definition from the ISTQB® glossary or Chapter 9 shall be remembered (K1), even if not explicitly mentioned in the learning objective.

Hands-on Objectives (HO) focus on the practical application of the Learning Objectives and are shown at the beginning of each chapter. The level of a HO is classified as follows:

- H0: This can include a live demo of an exercise or a recorded video. Since the trainee does not perform this, it is not strictly an exercise.
- H1: Guided exercise. The trainees follow a sequence of steps performed by the trainer.
- H2: Exercise with hints. The trainee is given an exercise with relevant hints to enable the exercise to be solved within the given timeframe.

0.6 The Certified Tester AI Testing Certificate Exam

The CT-AI exam will be based on the Learning Objectives described in this syllabus. All syllabus sections are examinable except for the Introduction, Hands-On Objectives, References, and Appendices. Answering exam questions may require using material based on more than one section of this syllabus. Standards and books are included as references, but their content is not examinable beyond what is summarized in the syllabus itself.

Refer to the Exam Structures and Rules document for the CT-AI v2.0 for further details.

The main entry criterion for anyone interested in taking the CT-AI exam is holding the ISTQB® Certified Tester Foundation Level [CTFL] certificate.

It is strongly recommended that candidates also:

- Have a minimal background in either software development or software testing, such as six months of experience as a system or user acceptance tester, data scientist, or software developer.
- Take a course accredited to ISTQB® standards (by one of the ISTQB-recognized member boards).

0.7 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 allows an ISTQB® exam to be included in the course.

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

0.8 Handling of Standards

International standardization organizations like IEEE and ISO have issued standards associated with quality characteristics and software testing. The purpose of these references is to provide a framework (as in the references to ISO/IEC 25059 and ISO/IEC 25010 regarding a quality model for AI-based systems) or to provide a source of additional information if desired by the reader. Please note that syllabi are using the standard documents as a reference. Standards documents are not intended for examination.

0.9 Level of Detail

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

- General instructional objectives describing the intention of the CT-AI syllabus
- A list of keywords that students must be able to recall
- Learning objectives for each knowledge area, describing the cognitive learning outcome to be achieved
- A description of the key concepts, including references to sources such as accepted literature or standards

The syllabus content does not describe the entire knowledge area of software testing; it reflects the level of detail to be covered in CT-AI training courses. It focuses on introducing the basic concepts of AI and machine learning (ML) in particular, and how systems based on these technologies can be tested.

The syllabus uses the terminology (i.e., the name and meaning) of the terms used in software testing and quality assurance according to the ISTQB® Glossary.

0.10 How this Syllabus is Organized

There are seven chapters with examinable content. The top-level heading for each chapter specifies the chapter duration; timing is not provided at the chapter level. *For accredited training courses, the syllabus requires a minimum of 19.5 hours of instruction, distributed across the seven chapters as follows:*

- Chapter 1: 120 minutes - Introduction to Artificial Intelligence (AI)
 - Understand the key differences between AI-based systems and conventional systems, and explore the spectrum of AI capabilities, ranging from narrow AI to super AI.
 - Gain a foundational understanding of AI technologies, including generative AI (GenAI), and the hardware, hosting, and development frameworks used to implement machine learning systems (MLS).
 - Learn how regulations and standards influence the development and testing of AI-based solutions.
- Chapter 2: 45 minutes - Quality Characteristics for AI-Based Systems
 - Learn about quality characteristics specific to AI-based systems, including those defined in ISO/IEC 25059, and understand safety-related considerations when using AI in critical systems.
 - Explore how to define appropriate acceptance criteria tailored to the unique behavior and performance of AI-based solutions.
- Chapter 3: 375 minutes - Machine Learning
 - Understand the types of ML, key steps in the ML development workflow, and how pretrained models, fine-tuning, and retrieval-augmented generation contribute to modern AI-based systems.
 - Learn about data preparation, the roles of training, validation, and test datasets, and how these influence ML model development and performance.
 - Explore neural networks, including their structure and coverage measures, and gain hands-on experience with performance metrics and using a confusion matrix.
- Chapter 4: 195 minutes - Testing AI-Based Systems
 - Understand the unique testing challenges of AI-based systems, including differences in testability between locked and adaptive systems, the need for statistical testing, and the difficulties of defining test oracles.
 - Learn how to test GenAI and large language models (LLM), using techniques such as red teaming and exploratory testing for AI performing test tasks.
 - Explore test strategies for MLS, covering various test levels and the application of risk-based testing.
- Chapter 5: 180 minutes - Input Data Testing for Machine Learning Systems

- Learn how to test and validate input data for MLS, including techniques for detecting bias, verifying label correctness, assessing data representativeness, and testing the data pipeline.
- Chapter 6: 225 minutes - Model Testing for Machine Learning Systems
 - Discover test approaches for mitigating risks in ML models, including documentation reviews, ML functional performance testing, and detecting overfitting, underfitting, and drift.
 - Learn advanced test approaches, including adversarial testing, A/B testing, back-to-back testing, and the use of attacks to uncover model weaknesses.
 - Gain hands-on understanding of metamorphic testing, including how to derive and apply test cases where traditional test oracles are insufficient.
- Chapter 7: 30 minutes - Machine Learning Development Testing
 - Learn test approaches and test strategies for mitigating risks during MLS development and deployment, to support robust system behavior in production environments.

1 Introduction to Artificial Intelligence – 120 minutes

Keywords

None

AI-Specific Keywords

AI-based system, artificial intelligence, general AI, machine learning, ML development framework, narrow AI, super AI

Learning Objectives for Chapter 1:

1.1 Introduction to AI

- AI-1.1.1 (K2) Differentiate between AI-based systems and conventional systems
- AI-1.1.2 (K2) Distinguish between narrow AI, general AI, and super AI
- AI-1.1.3 (K2) Explain the different types of AI technologies
- AI-1.1.4 (K2) Explain generative AI
- AI-1.1.5 (K2) Compare the choices available for hardware to implement machine learning systems
- AI-1.1.6 (K2) Compare the options for the development and hosting of AI models
- AI-1.1.7 (K2) Summarize the functionality provided by ML development frameworks
- AI-1.1.8 (K2) Explain how regulations and standards affect the development and testing of AI-based systems

1.1 Introduction to AI

This chapter introduces key distinctions between conventional and AI-based systems, focusing on their design approaches, adaptability, and explainability. It describes the spectrum of AI capabilities, ranging from narrow AI to general AI and super AI, and outlines core technologies such as ML, deep learning, and GenAI. The chapter describes common hardware and development environments for AI-based systems, as well as popular ML frameworks. Finally, it considers essential regulatory and technical standards that guide responsible AI development and deployment within various domains.

1.1.1 AI-Based and Conventional Systems

Conventional computer systems are typically programmed using imperative languages, where human developers explicitly define step-by-step instructions, including constructs such as if-then-else statements and loops. This deterministic approach helps make the system's behavior predictable and transparent, making it easier for humans to understand how inputs produce different outputs. In contrast, most AI-based systems, particularly those leveraging ML, do not follow predefined rules. Instead, they analyze patterns in the data to determine how to respond to new inputs. For example, an AI-based image recognition system trained to identify cats does not rely on explicitly coded rules. Instead, it learns from a dataset of cat images, extracting patterns that it later applies to unseen images to classify them accurately.

One fundamental difference between conventional and AI-based systems is how they approach problem-solving. Many AI-based systems rely on probabilistic reasoning, statistical inference, and pattern recognition to generate results. This allows AI models to handle complex forms of uncertainty and ambiguity more effectively, resulting in outputs that are not always predictable.

A key challenge in AI-based systems is explainability. Many AI models, particularly deep learning architectures, can contain billions of parameters, making their internal workings difficult for humans to interpret. This “black-box” nature raises concerns in critical domains such as healthcare, finance, defence, and transportation, where understanding why an AI model made a particular decision is crucial. Achieving transparency (see 2.1.1) and explainability in AI-driven decision-making has become a critical focus in AI regulation.

Another significant distinction is adaptability. Conventional systems are static and typically require manual updates to incorporate new knowledge or respond to environmental changes. AI-based systems, in contrast, can be self-learning, continuously improving their performance as they encounter new data. This adaptability makes AI particularly powerful in dynamic environments. Adaptability also requires continuous monitoring to maintain ongoing alignment with core requirements.

1.1.2 Narrow AI, General AI, and Super AI

Narrow AI, also known as weak AI, is designed to perform specific tasks and represents all deployed AI systems in use today. Narrow AI-based systems operate within a limited domain and can be highly efficient at solving specialized problems, such as image recognition, speech processing, and language translation. However, they lack the ability to generalize beyond the functions they have learned. For instance, an AI-based system that excels at recognizing faces cannot perform language translation unless it is explicitly retrained to do so. Frontier AI, a subset of narrow AI, represents the most advanced form of these systems, pushing the boundaries of current capabilities with GenAI models. Frontier AI includes large-scale systems with highly autonomous decision-making capabilities; however, they remain task-specific and have not yet achieved the versatility of general AI.

General AI, also known as strong AI, refers to an AI-based system that possesses the ability to perform most intellectual tasks that a human can. General AI would be able to understand, learn, and apply knowledge across a broad range of tasks without needing to be retrained for each new task. This type of AI would exhibit human-like reasoning and adaptability, capable of solving unfamiliar problems in various domains, much like humans do. Despite significant progress in AI, no AI-based system today possesses general intelligence.

Super AI (artificial superintelligence) is the form of AI in which an AI-based system continuously improves itself without the need for human intervention or control. For Super AI to be possible, access to the Internet is not strictly required, but such access could significantly expand its capabilities and influence. It would surpass human intelligence and general AI, which many believe would pose an existential risk to humanity. The point at which AI-based systems transition from general AI to super AI, if it were to occur, is commonly known as the technological singularity.

1.1.3 Different Types of AI Technologies

Artificial Intelligence encompasses a range of technologies, each suited to specific tasks and challenges. One of the core branches of AI is ML, which enables systems to learn from data and build models without explicit programming. While some MLs can adapt and improve continuously with new data throughout their lifetimes, others operate on the knowledge they initially learned and require explicit retraining to update their capabilities. ML includes several approaches:

- Supervised learning utilizes labeled data and algorithms, such as linear regression and decision trees, for tasks like prediction and classification.
- Unsupervised learning uncovers patterns in unlabeled data through techniques like clustering (using clustering algorithms).
- Reinforcement learning enables intelligent agents to learn optimal behaviors through trial-and-error interactions with their environment.

Key ML technologies include neural networks, Bayesian models, support vector machines (SVM) and random forests. See Chapter 3 for more on ML.

Deep Learning (DL), a subset of ML methods, uses deep neural networks to solve complex problems. For example:

- Convolutional neural networks (CNN) are highly effective for image recognition and object detection.
- Recurrent neural networks (RNN) specialize in processing sequential data, such as text or time series.
- Transformers handle long-range dependencies in sequences, powering models for natural language processing and vision transformers for images.

GenAI builds on these technologies to create new content, including text, images, and audio (see 1.1.4). This is driven by models such as LLM, which combine deep neural networks (DNN) with natural language processing (NLP) to analyze and generate human-like language.

Other specialized AI technologies include:

- NLP for language analysis, including tasks like sentiment analysis and machine translation.
- Computer vision for analyzing visual data, supporting applications like facial recognition and robotics.

- Fuzzy logic for reasoning under uncertainty.
- Search algorithms for solving optimization problems, such as navigation, and strategic decision-making.
- Rule-based reasoning systems, or expert systems, for structured decision support.

Although the full integration of these AI technologies remains limited, new developments such as LLM demonstrate the potential to combine different AI technologies into unified, intelligent systems. Agentic AI extends these technologies through autonomous agents that plan, reason, and act independently to achieve goals in dynamic environments.

1.1.4 Generative AI

Generative AI (GenAI) refers to AI-based systems specialized in creating new content, such as text, images, video, music, or complex data, while many also support classification and prediction. These systems learn from vast amounts of data to produce outputs that resemble their training data, enabling a wide range of creative and practical applications.

The main technologies behind GenAI include generative adversarial networks (GANs), diffusion models, and transformers. GANs use two neural networks in competition to create highly realistic synthetic data. Diffusion models generate content by gradually adding and then removing noise from data, resulting in high-quality outputs. Transformer models, which underpin LLM, utilize self-attention mechanisms to generate coherent and contextually relevant text, and are increasingly being adapted for multimodal tasks.

Beyond their technical foundations, GenAI systems raise significant societal and ethical concerns. Misuse is a major concern: these models can be exploited to create deepfakes, spread misinformation, or generate convincing fraudulent content, thereby undermining trust in digital media and public discourse. The ease of producing synthetic content amplifies risks related to privacy, security, and manipulation, but it can bring notable benefits and opens opportunities for innovation in entertainment, marketing, education, and research.

The impact on employment is also notable, particularly among white-collar professions. As GenAI automates tasks such as writing, design, coding, and even legal or medical documentation, there is increasing debate about potential job displacement. While these technologies can boost productivity and foster new creative opportunities, they may also lead to workforce disruption and necessitate widespread reskilling.

Sustainability is another pressing issue. Training and running large GenAI models involve substantial computational resources, resulting in high energy consumption and a significant carbon footprint. This environmental impact has prompted calls for more efficient model designs and greener infrastructure.

Most practical GenAI tools today are based on foundation models, which are then fine-tuned for specific applications. The field is also advancing toward multimodal models capable of processing and generating content across text, images, and audio, enabling richer, more flexible AI-based systems. Regulatory frameworks, such as the EU AI Act [EU AI Act], are emerging to guide the responsible development and use of these technologies.

1.1.5 Hardware for Machine Learning Systems

A variety of hardware is used for MLS. Different types of hardware may be used for training and inference. For example, a speech recognition model may run on a low-end smartphone, although access to the power of cloud computing may be needed to train it.

ML typically benefits from hardware that supports the following:

- The ability to work with large data structures.
- Massively parallel (concurrent) processing, for example, to support matrix multiplication.
- Low-precision arithmetic (quantization): This approach uses fewer bits for computation (e.g., 4 bits instead of 32 bits), resulting in faster processing, lower power consumption, smaller and more cost-effective chips, and reduced bandwidth requirements.

General-purpose central processing units (CPU) provide support for complex operations with high precision that are not typically required for ML applications, but they typically provide only a few cores. As a result, their architecture is less efficient for training and running ML models compared to graphics processing units (GPU), which have thousands of cores and are designed to perform massively parallel, yet relatively simple, graphics processing. Consequently, GPUs typically outperform CPUs in ML applications, even though CPUs usually run at higher clock speeds. For small-scale ML work, GPUs generally offer the best option.

Some hardware is specifically designed for AI, such as purpose-built Application-Specific Integrated Circuits (ASIC) and System-on-a-Chip devices. These AI-specific solutions feature multiple cores, specialized data management, and the capability to perform in-memory processing. They are best suited for edge computing, while the training of the ML model is performed in the cloud using specialized hardware.

AI-specific hardware architectures continue to be developed. This includes neuromorphic processors, which do not use the traditional von Neumann architecture but rather brain-inspired designs mimicking neuronal structures.

1.1.6 Development and Hosting of AI Models

AI-based systems can be acquired from third-party vendors or developed privately within an organization. AI-based systems typically rely on pretrained models and can be deployed on-premises or in the cloud, with the models themselves hosted either on-premises or in the cloud, where cloud-based options are often accessed as a service (AlaaS). Third-party AI-based systems typically come as pretrained models or AI as a Service (AlaaS), enabling faster deployment and quicker time-to-market. Alternatively, private AI-based systems (on-premises or customized cloud setups) can be better tailored to specific requirements, but their development will likely require specialized skills, either through in-house experts or outsourced teams.

Local development (coding and training locally) enables direct control and privacy. Small models, such as decision trees or compact neural networks, can be developed on personal computers, while mid-sized models may require dedicated GPUs. For large-scale models, high-performance on-premises server clusters become necessary with their associated energy, cooling, and hardware costs.

Cloud development offers significant flexibility. Public clouds, in particular, provide pre-configured environments with pay-as-you-go pricing, which limits initial hardware investment and scales easily. In contrast, private clouds can provide enhanced security and privacy for applications that require it, but this control necessitates a greater upfront infrastructure investment.

Many organizations adopt hybrid approaches, including developing prototypes locally before scaling to cloud infrastructure, maintaining sensitive components on-premises, such as the preparation of private data, and leveraging cloud resources for compute-intensive tasks.

AI models can be hosted in various environments, ranging from local setups to cloud-based platforms. Local hosting involves running smaller models on personal computers or smartphones, offering privacy and eliminating cloud licensing costs, although this provides limited hardware capabilities. For larger AI models, organizations may establish dedicated servers, which require a significant upfront investment but provide enhanced control.

Cloud hosting of AI models can be on public or private clouds. Public cloud services provide scalable access to robust, powerful infrastructure, eliminating maintenance concerns and making them ideal for fluctuating workloads. Private clouds offer similar benefits, with enhanced security and customization options, and are either managed in-house or through dedicated providers, albeit at a higher cost.

Hybrid approaches combine these methods, allowing organizations to run some operations locally while leveraging cloud elasticity for intensive tasks.

The optimal development and hosting solutions, which are typically decided upon separately, depend on factors such as model size, complexity, performance requirements, budget constraints, security and data privacy considerations, deployment needs and regulatory requirements. Some organizations adopt multi-tiered strategies to balance efficiency and control.

1.1.7 Machine Learning Development Frameworks

ML development frameworks provide a toolkit for building and training ML models. Typical functionality provided by these frameworks includes:

- **Data Handling:** They assist with loading, preprocessing, and managing the data used to train and test the model. This might involve cleaning, formatting, and transforming the data into a suitable format for the chosen model.
- **Model Building:** These frameworks offer libraries of ML algorithms and tools to design the architecture of the constructed ML model. This includes specifying the type of model (e.g., neural network, decision tree), the number of layers and connections, and the mathematical operations performed within the model.
- **Training and Optimization:** Frameworks provide algorithms that iteratively adjust the model's internal parameters based on the training data and on the desired result. The goal is to optimize the model's performance in accomplishing the desired task (e.g., classification, ML regression). Some frameworks may support distributed training and enhance or fine-tune pretrained models.
- **Evaluation:** They offer tools to evaluate how well the trained model performs on unseen data. This might involve measuring accuracy, precision, and recall for classification tasks, or error rates for ML regression tasks (see 3.1.1).
- **Deployment:** Some frameworks provide capabilities for deploying the trained model for real-world use. This could involve converting the model into a format suitable for integration with web applications, mobile devices, edge devices or embedded systems.

These frameworks can operate at different levels of abstraction. Some offer a lower-level application programming interface (API), providing developers with more control over model building but requiring more coding expertise. Others offer a higher-level API, simplifying model creation but offering fewer customization options.

Different frameworks can focus on different application domains. Some are general purpose and support a wide range of application areas. In contrast, others are more specialized, focusing on specific areas such as image recognition, speech recognition, and language translation.

Selecting the most appropriate framework can depend on several factors, such as:

- the application area;
- the need for a user-friendly interface for rapid prototyping;
- configurability for complex models;
- the expertise of the users;
- deployment considerations, as some frameworks are better suited for resource-constrained environments;
- level of (community) support;
- ecosystem maturity.

1.1.8 Regulations and Standards for AI

AI regulations and standards are crucial for the responsible development, deployment, and use of AI. The core aim is to foster trust in AI and help promote realization of AI's benefits while mitigating potential harms. Ideally, compliance with such regulations and standards should guarantee that AI-based systems are safe, fair, transparent, sustainable, accountable, ethical, and used responsibly.

Internationally, the OECD AI Principles [OECD AI] and the UN report on Governing AI for Humanity [UN Gov AI] serve as influential soft law instruments that foster a shared understanding of responsible AI stewardship. They act as a compass for national governments and organizations as they formulate their own AI strategies. These principles emphasize human-centric AI, ethical considerations, and the importance of international cooperation.

The EU AI Act represents a landmark regulatory step, demonstrating a risk-based approach to regulating AI. By categorizing AI-based systems by risk, from minimal to unacceptable, regulations are tailored accordingly. High-risk systems, particularly those that impact fundamental rights or safety, face stringent requirements that encompass rigorous testing, data governance, and human oversight. The substantial financial penalties for non-compliance, based on a percentage of global turnover, underscore the EU's commitment to enforcement. In contrast, many nations outside the EU are adopting a more permissive approach, favoring lighter-touch regulations to encourage innovation.

Technical standards, developed by organizations such as ISO and IEEE, are crucial for translating high-level aspirations into practical implementations. They provide concrete technical specifications and best practices, bridging the gap between policy and practice. For example, ISO/IEC TR 29119-11 provides detailed guidance on testing AI-based systems, a critical element in demonstrating regulatory compliance. Meanwhile, the ISO/IEC 42119 series is being developed to cover various aspects of AI-based system testing. Furthermore, sector-specific regulations are emerging in areas such as healthcare and finance, recognizing the unique risks posed by AI in these domains.

To effectively navigate this evolving landscape of AI governance, continuous dialogue and collaboration are paramount. Governments, industry, academia, and civil society must actively engage to achieve a harmonized and effective approach to AI governance worldwide. Moreover, given the dynamic nature of AI, regulations and standards must be regularly reviewed and updated to remain relevant and effective in guiding responsible AI development, deployment, and use.

2 Quality Characteristics for AI-Based Systems – 45 minutes

Keywords

Functional adaptability, AI functional correctness, intervenability, AI robustness, safety, societal and ethical risk mitigation, transparency, user controllability

AI-Specific Keywords

None

Learning Objectives for Chapter 2:

2.1 Quality Characteristics for AI-Based Systems

- AI-2.1.1 (K2) Classify behaviors of AI-based systems according to the quality characteristics defined in ISO/IEC 25059
- AI-2.1.2 (K2) Explain the special considerations that arise when AI is used in safety-related systems

2.2 Acceptance Criteria for AI-Based Systems

- AI-2.2.1 (K2) Give examples of acceptance criteria for AI-based systems

2.1 Quality Characteristics for AI-Based Systems

This section covers AI-specific quality characteristics outlined in ISO/IEC 25059, which extends traditional software quality models to address unique aspects of AI-based systems. It introduces new and adapted characteristics, including AI functional correctness, functional adaptability, user controllability, transparency, AI robustness, and intervenability, alongside societal and ethical risk mitigation. Major AI safety challenges, such as vague specifications, non-determinism, self-learning, limited explainability, and evolving standards, are also addressed, with an emphasis on their impact on testing and regulation.

2.1.1 AI-Specific Quality Characteristics

ISO/IEC 25059 extends the quality model from ISO/IEC 25010 to address AI-specific considerations. This extension evaluates AI-based systems from two perspectives: product quality and quality in use. From a testing perspective, these quality characteristics directly influence how test objectives are defined, how acceptance criteria are formulated, and how test results are interpreted for AI-based systems. New and modified characteristics, compared to ISO/IEC 25010, include:

- **AI functional correctness (product quality):** AI-based systems, especially those using probabilistic ML, cannot guarantee perfect accuracy. Since a certain error rate is expected in AI outputs, the concept of functional correctness has been adjusted accordingly. ISO/IEC 25059 evaluates functional correctness by considering both correct and incorrect outputs and by defining acceptable thresholds for incorrect results, reflecting the inherent variability in AI-based system outputs (see 3.3).
- **Functional adaptability (product quality):** a new sub-characteristic of functional suitability. The ability of the AI-based system to autonomously adapt to changes in its operational environment after it is deployed.
- **User controllability (product quality):** a new sub-characteristic of interaction capability (note that interaction capability is itself a new term that replaces usability in the 2023 version of ISO/IEC 25010). A property of an AI-based system such that a human or another external agent can intervene in its functioning in a timely manner.
- **Transparency (product quality and quality in use):** a new sub-characteristic of interaction capability and a new sub-characteristic of satisfaction. It relates to the degree to which appropriate information about the AI-based system is communicated to stakeholders (see 6.1.2).
- **AI robustness (product quality):** a new sub-characteristic of reliability. It describes the ability of an AI-based system to maintain its level of AI functional correctness regardless of circumstances, such as the presence of biased, adversarial, or invalid data inputs, external interference, adverse environmental conditions, and operator misuse.
- **Intervenability (product quality):** a new sub-characteristic of security. The degree to which an operator can intervene in an AI-based system's functioning in a timely manner to prevent harm or hazard.
- **Societal and ethical risk mitigation (quality in use):** a new sub-characteristic of 'Freedom from risk'. Considers many areas to mitigate societal and ethical risk, including accountability, fairness and non-discrimination, professional responsibility, promotion of human values, privacy, safety and security, human control of technology, community involvement and development, human-

centered design, respect for the rule of law, respect for international norms of behavior, environmental sustainability, and labor practices.

2.1.2 AI and Safety

Safety-related systems have the potential to cause injury or harm to people, property or the environment. Developing and testing non-AI safety-related systems can take a lot of effort, but is feasible; however, for AI-based systems, there are several additional challenges:

- **Specifications:** In traditional safety-related systems, requirements are defined for the complete system and refined until the developer can transform them into code. The requirements for many AI-based systems often begin with vague goals and are then implicitly provided via the training data that encodes patterns, rules, and objectives, without fully formalizing every detail upfront. This can mean that the necessary traceability from requirements to implementation is inadequate for AI-based systems.
- **Non-determinism:** This characteristic of many AI-based systems makes it inherently challenging to guarantee the precise behavior of these systems. Even rigorously tested models can exhibit unexpected behavior due to factors such as random number generation or slight variations in input values.
- **Self-learning:** Rigorous testing is used to demonstrate the safety integrity of a system before deployment. For self-learning AI-based systems, this is undermined as the system's behavior progressively moves away from the originally tested behavior. Managing how the model learns and the data it uses can sometimes help to avoid the emergence of new problematic behaviors. Alternatively, safety guards can be implemented to help prevent the model from learning or making decisions that could compromise safety (e.g., a content moderation component to filter prompts).
- **Explainability and Transparency:** For safety-related systems, it is essential to understand how and why the system makes decisions. However, the decision-making processes of AI-based systems are often not transparent. Explainable AI techniques, such as LIME (Local interpretable model-agnostic explanations), can provide insights into the AI-based system's reasoning; however, they are not widely available and may compromise system performance.
- **Evolving Regulations:** The regulatory landscape for safety-related AI-based systems is constantly evolving. The use of AI is currently not included in mature functional safety international standards, and some of these standards even prohibit its use in such systems. The EU AI Act [EU AI Act] (see 1.1.8) classifies AI systems used as safety components (such as in aviation, medical devices, or automotive) as high-risk and imposes strict requirements on their development and testing.

2.2 Acceptance Criteria for AI-Based Systems

This section outlines acceptance criteria related to the quality characteristics specified in the ISO/IEC 25059 standard, as well as safety. For AI-based systems, acceptance criteria often need to be statistical, probabilistic, or threshold-based rather than binary, which introduces additional testing challenges.

2.2.1 Acceptance Criteria for AI-Based Systems

When evaluating the quality of an AI-based system, it is essential to consider both functional and non-functional quality characteristics. This helps confirm that the AI-based system functions as intended and satisfies broader quality requirements. The ISO/IEC 25010 and ISO/IEC 25059 standards provide a comprehensive framework for defining software quality. In this section, the focus is on the acceptance criteria associated with the quality characteristics specific to AI (i.e., those defined in ISO/IEC 25059) and safety (see 2.1.2).

The following table lists example acceptance criteria for safety and each of the quality characteristics defined in the ISO/IEC 25059 standard.

Characteristic	Example Acceptance Criteria
AI functional correctness (see 3.3)	<ul style="list-style-type: none"> • Accuracy of 95% for an image recognition system. • Recall of 90% for a defect prediction system.
Functional adaptability	<ul style="list-style-type: none"> • A maximum of 20 seconds for the engine management system to adapt when it crosses a specified altitude threshold. • A video streaming service shall adjust its homepage to recommend at least 40% of documentaries after a user watches three full-length documentaries in a single session.
User controllability	<ul style="list-style-type: none"> • A supervisor can take control of an autonomous drone within 0.5 of a second when it sends a distress signal due to loss of its GPS location. • The farm control system notifies the farmer when the sensor's visual performance degrades by more than 30%, allowing immediate manual override; it fully deactivates if degradation exceeds 50% without user response.
Transparency	<ul style="list-style-type: none"> • Sufficient information is provided about the third-party ML model and the provenance of its training data to meet the requirements of the relevant company standard. • The system's operational dashboard and API must provide an endpoint that returns the unique version identifier of the currently deployed prediction model and a link to its corresponding documentation.
AI robustness	<ul style="list-style-type: none"> • The response time for the AI-based security penetration alert system predictions remains below 1 second when access to the central vulnerabilities database is disrupted for 30 seconds. • The edge AI device shall automatically transition to a lower-fidelity, reduced-power inference mode (instead of crashing) when its internal operating temperature exceeds 85°C for a continuous period of 10 seconds.

Intervenability	<ul style="list-style-type: none">• If a robot breaches its safety zone, the production line is capable of being shut down within 0.5 seconds after a shutdown is initiated.• To prevent potential blackouts, the power grid management system shall provide a 30-second confirmation window, during which an engineer can veto any AI-proposed action classified as 'critical' before it is automatically executed.
Societal and ethical risk mitigation	<ul style="list-style-type: none">• The automated prison sentencing system does not discriminate between racial groups based on the specified fairness metric.• The chatbot must pass an internal "red teaming" assessment with a score of 95% or higher, demonstrating its refusal to generate content that promotes violence, self-harm, or hate speech.
Safety	<ul style="list-style-type: none">• Non-AI components of the AI-based steering control system are compliant with ISO 26262-6 at ASIL (automotive safety integrity level) C.• 100% of the relationships between inputs and outputs of the ML model in the nuclear power plant control system can be mapped with an average accuracy of no lower than 99.9% by an explainability tool.• Control signals that exceed specified safety limits by more than 10% are analyzed and regulated within 0.15 seconds after being detected by the safety monitoring sub-system.

3 Machine Learning – 375 minutes

Keywords

K-multisection neuron coverage, ML functional performance criteria, ML functional performance metric, ML model, neuron boundary coverage, neuron coverage, perceptron

AI-Specific Keywords

Association, classification, clustering, data preparation, machine learning, ML algorithm, ML development framework, ML workflow, pretrained model, ML regression, reinforcement learning, supervised learning, unsupervised learning

Learning Objectives for Chapter 3:

3.1 Introduction to Machine Learning

- AI-3.1.1 (K2) Distinguish between the different forms of ML
- AI-3.1.2 (K2) Summarize the workflow used to create an ML system
- HO-3.1.3 (H2) Create an ML model
- AI-3.1.4 (K2) Summarize the use of pretrained models, fine-tuning, and retrieval-augmented generation

3.2 Data for Machine Learning

- AI-3.2.1 (K2) Explain the activities related to data preparation
- HO-3.2.2 (H2) Perform data preparation to support the creation of an ML model
- AI-3.2.3 (K2) Contrast the use of training, validation, and test datasets in the development of an ML model

3.3 ML Functional Performance Metrics for Classification

- AI-3.3.1 (K3) Calculate common ML functional performance metrics from a given set of confusion matrix data
- HO-3.3.2 (H2) Evaluate an ML model using selected ML functional performance metrics
- HO-3.3.3 (H2) Show the impact of different ML models and dataset combinations on the training and behavior of the models

3.4 Neural Networks

- AI-3.4.1 (K2) Explain the structure and working of a deep neural network
- HO-3.4.2 (H1) Experience the implementation of a perceptron
- AI-3.4.3 (K2) Describe the different coverage measures for neural networks

3.1 Introduction to Machine Learning

This section introduces the main categories of ML algorithms: supervised, unsupervised, and reinforcement learning, distinguishing their respective problem types and typical applications. It outlines the standard workflow for developing ML models, from defining objectives and preparing data to training, evaluating, and deploying models, with attention to iteration and system integration. Practical aspects such as hands-on model creation, the use of pretrained models, fine-tuning, and retrieval-augmented generation are also covered, highlighting approaches to efficiently adapt and enhance AI models for new tasks while managing inherited limitations.

Understanding this workflow is essential for testers, as different test activities apply at different stages, and failures often originate in earlier steps such as data preparation or model selection.

3.1.1 Different Forms of Machine Learning

ML algorithms are categorized into supervised learning, unsupervised learning, and reinforcement learning.

In supervised learning, algorithms train models using labeled data, where each set of inputs has a corresponding output label (e.g., images labeled as “dog” or “cat”). The model learns to map inputs to outputs by identifying patterns in the training data. Supervised learning is typically divided into:

- **Classification:** This involves assigning inputs to predefined classes, such as classifying emails as spam or not, or image recognition in images.
- **ML regression:** This involves predicting continuous numerical values, such as estimating a person’s age based on lifestyle data or forecasting stock prices.

Note that the term ML regression, when used in the context of ML, differs from its use in other ISTQB® syllabi, where regression describes the problem of software modifications causing change-related defects.

In unsupervised learning, the algorithm trains models using unlabeled data, inferring patterns or structures without explicit output labels. The model groups similar inputs together based on shared features. Unsupervised learning is typically categorized into:

- **Clustering:** This involves grouping data points based on similarities, such as segmenting customers into different groups for targeted marketing.
- **Association:** This involves identifying relationships or dependencies among data attributes, such as finding patterns in customer purchasing behavior to recommend products.

In reinforcement learning, the AI-based system (an “intelligent agent”) learns by interacting with its environment. The agent receives positive feedback (rewards) or negative feedback (penalties) based on the outcome of its actions, enabling it to learn from experience rather than from a dataset. Challenges in reinforcement learning include setting up the environment, designing the reward function, and selecting the best strategy to meet the desired goal. Applications include robotics, autonomous vehicles, and adaptive systems like chatbots.

Each ML approach addresses different types of problems, with the choice depending on the nature of the available data and the specific task at hand.

3.1.2 Machine Learning Workflow

The activities in the ML workflow, shown in Figure 1, are:

Understand the Objectives

The purpose of the ML model is understood and agreed to by the stakeholders to verify alignment with business priorities. Acceptance criteria (including ML functional performance metrics – see 3.3) are defined for the developed model.

Select a Framework

A suitable ML development framework (see 1.1.7 for details on provided functionality) is selected based on the objectives, acceptance criteria (see 2.2), and business priorities.

Select & Build the Algorithm

An ML algorithm is selected based on various factors, including the objectives, acceptance criteria, and the available data (see 3.2). The algorithm may be manually coded, but it is often retrieved from a software library. The algorithm is then compiled, if required.

Prepare & Test Data

Data preparation (see 3.2) comprises data acquisition, data preprocessing, and feature engineering. Exploratory data analysis (EDA) may be performed alongside these activities.

The data used by the algorithm and model are based on the objectives and are utilized by all activities in the 'model generation and test' box shown in Figure 1.

The data used to train, evaluate, tune, and test the model must be representative of the data that will be used by the model operationally.

Testing of the data and any automated data preparation steps is performed (see Chapter 5).

Train the Model

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

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., the number of iterations 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 (see 3.3), using the validation dataset, and the results are used to improve the model in the 'tune the model' activity. In practice, several models are typically created and trained using different algorithms (e.g., random forests, SVM, and neural networks) and various training datasets, and the best combination is chosen based on the evaluation results.

Tune the Model

The results from the evaluation are used to adjust the model hyperparameters and the algorithm hyperparameters. The model is then retrained with these adjusted settings to improve its ML functional performance.

The three activities of training, evaluation, and tuning comprise 'model generation', as shown on Figure 1.

Test the Model

Once an acceptable model has been generated by the 'model generation' activities, it is tested using an independent test dataset to verify that the agreed ML functional performance criteria are met. The test results are also compared with those from the evaluation. If the performance of the model with independent test data is significantly lower than during evaluation, it may be necessary to return to the 'model generation' activities, or even to the 'prepare & test data' activity to train a new model.

In addition to ML functional performance tests, non-functional tests, such as for the time to provide a prediction, may also be performed. Typically, testing in this activity is performed by data engineers or scientists; however, testers with sufficient knowledge of the domain and access to the relevant resources can also perform this testing.

Deploy the Model

Once 'model generation & test' is complete, the tuned model is typically re-engineered for deployment along with its data pipeline. This is generally achieved through the ML development framework. Target platforms might include embedded systems and the cloud, where the model can be accessed via a web API. The re-engineered deployed model is tested to verify it still meets its acceptance criteria.

Use the Model

Once deployed, the model is typically integrated into a larger operational AI-based system. Models may perform scheduled batch predictions at set time intervals or run in real-time upon request.

Monitor & Tune the Model

While the model is being used, its situation may evolve, and the model may drift away from its intended performance (see 6.1.7). To verify that any drift is identified and managed, the operational model is regularly evaluated against its acceptance criteria.

It may be deemed necessary to create a new model by retraining with new data, retraining with new hyperparameters, or both. The latest model may then be compared to the existing model using a form of A/B testing (see 6.1.9).

The ML workflow shown in Figure 1 is a logical sequence; however, in practice, the workflow is applied iteratively, with steps repeated.

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 must be integrated with non-ML components (e.g., in vision applications, a data pipeline is used to clean and modify data before submitting it to the ML model). When the model is part of a large system, it must be integrated into this system before deployment. In this case, integration, system, and acceptance test levels may be performed.

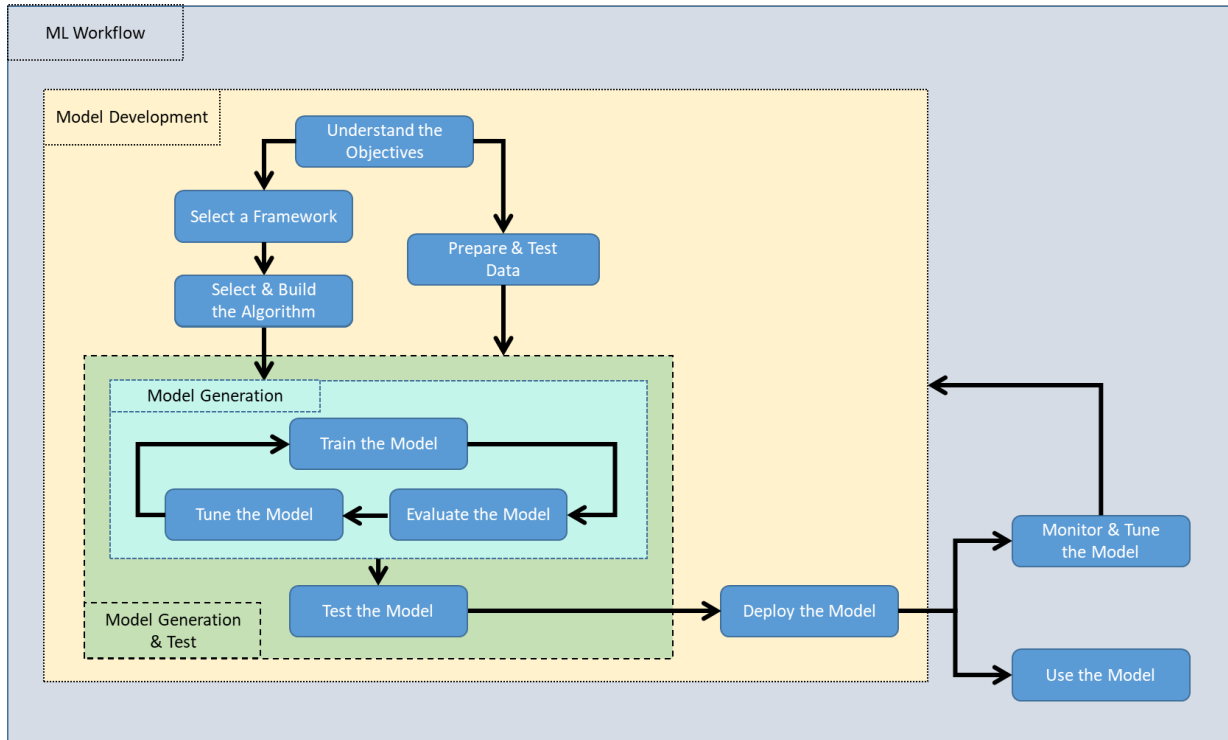


Figure 1: ML Workflow

3.1.3 Hands-on Exercise: Create a Machine Learning Model

Select, train, and test a classification model using supervised learning.

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

3.1.4 Pretrained Models, Fine-Tuning, and Retrieval-Augmented Generation

Training a new AI model from scratch is both costly and time-consuming. To address this, a common solution is fine-tuning, which involves taking a pretrained neural network and adapting it to perform a new, different task. One of the key benefits is that it requires much less training data (and training effort) compared to building a model from scratch.

The pretrained model is fine-tuned by performing additional training with data specific to the new task. The tuning can be applied to the entire neural network, only to specific layers (typically near the output end of the neural network), or to additional layers. After training, the model's ML functional performance is evaluated, and based on these results, further fine-tuning may be performed until the model meets the necessary acceptance criteria.

Fine-tuning success depends on the similarity between the original and new tasks. Small differences can lead to highly effective fine-tuning. For example, adapting a cat breed image classifier to identify dog breeds is likely to work well. However, adapting it for spoken accents is less effective due to the larger

difference. Similarly, fine-tuning an LLM for ISTQB-defined boundary value analysis requires a small change for the LLM and is readily achievable with good training data.

An alternative to fine-tuning is Retrieval-Augmented Generation (RAG), which involves providing data sources to the LLM that are specific to the required task. These data sources are transformed into a searchable format that allows them to be compared to the subject of the prompt. Once relevant documents are identified, these are incorporated into an enhanced prompt, which is passed to the LLM. As more pertinent information is now provided to the LLM, its corresponding response is likely to be more precise. With RAG, no change is made to the pretrained model.

A pretrained model can use RAG, fine-tuning, or both together to enhance performance.

Typically, any biases or vulnerabilities in the pretrained model will carry over to the new model, so testing is necessary to confirm that it performs reliably and fairly in the new task.

3.2 Data for Machine Learning

Data preparation is recognized as one of the most crucial and resource-intensive activities in the ML workflow. If operational data differs significantly from training data, ML functional performance and safety assumptions may no longer hold. Data preparation typically consumes a significantly larger proportion of the overall effort compared to other stages, such as model selection and building. Data preparation is intrinsically linked to the data pipeline, which processes raw data and transforms it into a usable format for both training and prediction by ML models.

3.2.1 Activities in Data Preparation

Data preparation supports the achievement of the quality and suitability of data for model training. It involves several key activities:

- Data acquisition:
 - Identifying relevant data types (e.g., numerical, categorical, images, text).
 - Gathering data from diverse sources, such as databases, APIs, or real-time sensors.
 - Labeling data for supervised learning tasks, verifying accuracy and consistency.

The acquired data can take various forms (e.g., numerical, categorical, image, tabular, text, time series, sensor, geospatial, video, and audio).
- Data preprocessing:
 - Cleaning data, including:
 - removing defects, duplicates, and outliers to verify data accuracy and consistency;
 - imputing missing values using techniques like mean, median, or mode to maintain data completeness;
 - anonymizing or removing personal information to protect privacy and comply with regulations.
 - Transforming data formats, scaling, and normalizing to achieve consistency.
 - Augmenting data to increase sample size, incorporating adversarial examples to enhance robustness against adversarial attacks, and generating synthetic data.

- Sampling subsets to reduce training times and computational costs.
- Feature engineering:
 - Selecting relevant features based on their contribution to ML model performance.
 - Extracting a subset of informative and non-redundant features from existing features to reduce training times and computational costs.

In parallel to these data preparation activities, exploratory data analysis (EDA) is also typically carried out to provide insights into the data. This includes:

- discovering trends, patterns, and anomalies in the data;
- visualizing data using plots and charts for better understanding.

Preparing training data is typically an iterative process, often performed manually, and individual data preparation activities may be reordered or omitted based on specific project requirements. The operational data should match the characteristics of the training data (e.g., data distributions and feature ranges) so that the model performs as expected in production. However, the preparation steps themselves may be adjusted for production efficiency and scalability.

3.2.2 Hands-on Exercise: Data Preparation in Support of the Creation of a Machine Learning Model

For a given set of data, perform the applicable data preparation steps as outlined in Section 3.2.1 to produce a dataset that will be used to create a classification model using supervised learning.

This activity serves as the first step in creating an ML model that will be utilized in future exercises.

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

- libraries;
- ML development framework;
- tools.

3.2.3 Training, Validation, and Test Datasets

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

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

If an abundance of 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 expected complexity of the model.
- The algorithm used to train the model.

- The availability of resources, such as RAM, disk space, computing power, network bandwidth, and the available time.
- The desired confidence in the resultant model.

When data is limited, a three-way split (train, validation, and test) can result in insufficient data for effective model training, thereby increasing the risk of underfitting. To address this, a common strategy is to set aside a small, final hold-out test set if feasible. The remaining data (the combined training and validation pool) is then used for techniques like k-fold cross-validation (where k is a user-specified integer, commonly 5 or 10).

In k-fold cross-validation, this data is divided into k 'folds.' For each fold, the model is trained on k-1 folds and validated on the held-out fold. This process is repeated k times, with each fold serving as the validation set once. This allows for robust hyperparameter tuning and ML functional performance estimation. Data is typically randomly assigned to folds, often using stratified sampling to make each fold representative, especially with imbalanced data or small datasets.

The performance metrics (e.g., accuracy, F1-score – see 3.3.1) from each fold's validation are then averaged to provide a more reliable estimate of the model's generalization ability. After identifying the optimal hyperparameters through cross-validation, a final model is typically trained on the entire training and validation pool (all data except the hold-out test set) using these hyperparameters. This final model is then evaluated once on the hold-out test set for a final, unbiased performance assessment. If a hold-out test set is not feasible due to extreme data scarcity, the average cross-validation performance is optimistically biased and cannot serve as an unbiased estimate.

Other resampling methods for limited data include leave-one-out cross-validation (a special case of k-fold where k equals the number of samples) and bootstrap techniques.

3.3 ML Functional Performance Metrics for Classification

In classification tasks (see 3.1.1), a confusion matrix can be used to evaluate a model's predictions, categorizing them as true positives, true negatives, false positives, or false negatives. Key metrics, such as accuracy, precision, recall, and F1-score, are derived from this. These metrics measure classification quality, highlighting the ML model's strengths and weaknesses. This section explores the calculation and interpretation of these metrics to assess ML functional performance.

3.3.1 Calculation of Machine Learning Functional Performance Metrics

In a classification problem, a model will rarely predict the results correctly all the time, partly due to the probabilistic nature of ML models and data noise. 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 (e.g., predicted and actual swapped), but it will always yield 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 = $(TP + TN) / (TP + TN + FP + FN) * 100\%$

Accuracy measures the percentage of all correct classifications.

- 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 = $TP / (TP + FN) * 100\%$

Recall (also known as sensitivity) measures the proportion of actual positives that are correctly predicted. It is a measure of how confident one can be that no positives will be missed.

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

F1-score is calculated as the harmonic mean of precision and recall, with values ranging from 0 to 100. A score close to 100 means the model achieves both high precision and high recall, indicating that classification errors (false positives and false negatives) have minimal impact. Conversely, a low F1-score indicates the model struggles to accurately identify positives, either missing true cases or generating many false alarms.

3.3.2 Hands-on Exercise: Evaluate a Machine Learning Model using Selected ML Functional Performance Metrics

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 ML development framework to perform the calculations.

3.3.3 Hands-on Exercise: Show the Impact of Different Machine Learning Models and Dataset Combinations

Following the previous exercise, use different ML models and dataset combinations to observe their effect on the training of the ML model, as well as the final behavior of the model. Observe the training times and the ML functional performance metrics.

3.4 Neural Networks

Artificial neural networks were initially designed to mimic the functioning of the human brain, which can be thought of as a network of interconnected biological neurons.

The single-layer perceptron is one of the earliest examples of implementing an artificial neural network, comprising just one layer. It can be used for supervised learning of binary classifiers for linearly separable problems, which determine whether an input belongs to a specific class or not. For example, a perceptron can distinguish between emails that are spam and those that are not spam, by learning to separate the features of the two categories with a straight line in the input space.

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

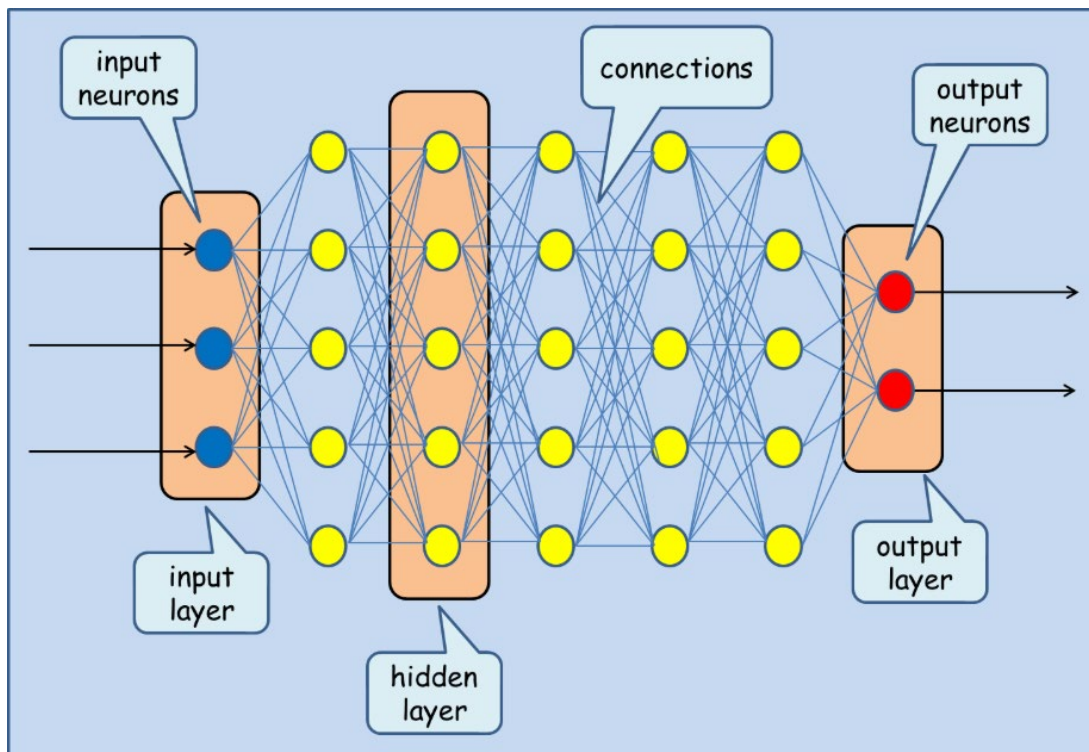


Figure 3: Structure of a deep neural network

3.4.1 Structure and Working of a Deep Neural Network

A deep neural network is typically described as comprising three main 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, for example, be a value indicating the likelihood that the input image is of a cat. Between the input and output layers are hidden layers composed of artificial neurons, also known as nodes. In many common architectures, such as fully connected networks, 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, gradually transforming the input data into increasingly abstract representations until reaching the output.

The computation performed by each neuron (after those in the input layer) generates what is known as the activation value. This value is calculated by first computing a weighted sum of the activation values from all connected neurons in the previous layer, where each such connection has its own independent weight, and adding the neuron's individual bias. This sum is then passed through a non-linear formula called the activation function. Note that this bias is unrelated to the bias considered in 5.1.2. Using different activation functions yields different activation values.

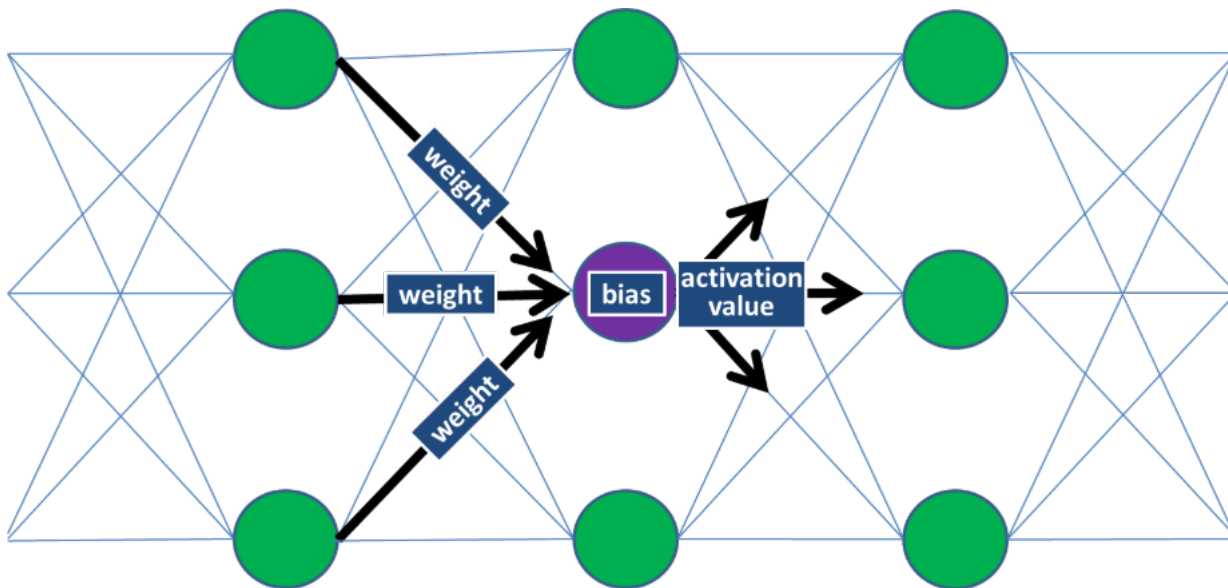


Figure 4: Computation performed by each neuron

The weights connecting neurons and each neuron's bias value are typically initialized to small random values (biases sometimes to zero) at the start of training. The training data is passed through the network, with each neuron running the activation function, to generate an output ultimately. The generated output is then compared with the known correct result. The resulting error (or loss), which quantifies this difference, is then fed back through the network to adjust the values of the weights and biases, thereby minimizing this difference. As more training data is fed through the network (each pass through the training dataset is called an epoch), the weights and bias values are gradually adjusted as the network learns. After some time, ideally, the output produced is considered good enough to end training.

3.4.2 Hands-on Exercise: Experience the Implementation of a 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 its weights and bias values across multiple epochs until the error is minimized to zero. Various mechanisms (e.g., spreadsheet, simulation) may be used for this activity.

3.4.3 Coverage Measures for Neural Networks

Neural network structural coverage metrics have emerged to assess how thoroughly test inputs exercise a model's internal mechanisms. As neural networks do not follow explicitly coded paths, but their behavior is dictated by learned weights, bias values, and activations, this creates a need for specialized coverage measures to evaluate the extent to which different parts of the network have been activated under test conditions.

Typical approaches include [COV_REF]:

- **Neuron Coverage:** Measures the proportion of neurons in the network where their output exceeds a specified threshold during testing.
- **k-Multisection Neuron Coverage (kMNC):** The possible output range of each neuron is divided into k sections. kMNC is the proportion of these sections activated during testing.
- **Neuron Boundary Coverage (NBC):** Measures the proportion of neurons in the network where their output either exceeds the maximum achieved in training, or is less than the minimum achieved in training during testing.

These coverage metrics can be helpful in AI testing, primarily by revealing areas of the model that remain untested, which may potentially hide defects or unlearned behaviors. For example, if certain neurons or layers never activate during testing, the model's performance in related decision boundaries might be questionable. Such insights help testers design additional test inputs or test conditions to exercise underexplored parts of the network. At present, commercial tools supporting these specific coverage measures are limited.

However, structural coverage alone does not guarantee that a neural network will generalize well or handle real-world variations. Neural networks can learn spurious correlations, leading to correct activations for incorrect reasons. Consequently, testers should also use other test techniques, like adversarial testing and metamorphic testing, as described in chapters 5, 6 and 7.

4 Testing AI-Based Systems – 195 minutes

Keywords

Attack, exploratory testing, risk-based testing, test oracle

AI-Specific Keywords

Adaptive AI-based system, AI-based system, generative AI, large language model, locked AI-based system

Learning Objectives for Chapter 4:

4.1 Introduction to Testing AI-Based Systems

- AI-4.1.1 (K2) Compare the testability of locked and adaptive AI-based systems
- AI-4.1.2 (K2) Explain why a statistical approach is often needed when testing AI-based systems
- AI-4.1.3 (K2) Explain the challenges and solutions relating to test oracles for AI-based systems

4.2 Testing Generative AI and LLM

- AI-4.2.1 (K2) Explain how generative AI can be tested
- AI-4.2.2 (K3) Implement red teaming for GenAI systems
- HO-4.2.3 (H2) Apply exploratory testing to an LLM performing boundary value analysis

4.3 Test Levels and Machine Learning Systems

- AI-4.3.1 (K2) Summarize the test levels used to develop machine learning systems
- AI-4.3.2 (K2) Explain how risk-based testing is applied to machine learning systems

4.1 Introduction to Testing AI-Based Systems

Testing AI-based systems presents unique challenges compared to testing conventional software. Firstly, AI-based systems can be broadly categorized as locked or adaptive. Locked AI-based systems, with fixed behavior post-deployment, are easier to test due to their mostly deterministic nature, while adaptive systems, which evolve and learn, introduce complexity as their behavior can change unpredictably. Additionally, the probabilistic nature of many AI models often necessitates statistical testing, as deterministic methods may be insufficient for evaluating outputs influenced by data and probabilities. This requires assessment of performance distributions and confidence levels across diverse scenarios.

A central challenge in testing, known as the 'test oracle problem', arises when determining whether the output produced by the system under test is correct for a given input. In traditional software, well-defined specifications make it relatively straightforward to specify expected results and verify correctness. However, with AI-based systems, especially those tackling complex or subjective tasks, clearly defining expected results can be difficult or even impossible. This uncertainty is compounded for tasks that exceed human capabilities, involve fuzziness, or lack objective 'ground truth', making it hard to implement automated test oracles and sometimes requiring expert judgment or statistical reasoning.

Vague or incomplete system requirements further exacerbate the test oracle problem in AI testing. Solutions may include using statistical evaluations, consulting domain experts, or defining a 'ground truth' against which to assess outputs. These approaches aim to establish reliable expectations and effectively evaluate AI-based systems.

4.1.1 Locked and Adaptive AI-Based Systems

Many AI-based systems in use today are locked AI-based systems, which do not change their behavior once they are deployed. An example of a locked AI-based system is a deployed ML model based on a DNN, where the DNN's weights and biases are fixed after development and can only be modified if the DNN is retrained. Safety-related self-driving car technology often relies on locked AI-based systems for specific tasks, such as lane detection or traffic sign recognition.

In contrast, an adaptive AI-based system, such as a reinforcement learning system, can adapt its behavior once it is deployed. The changes might be based on a reward function or to adapt to a new operational environment, but the specific details of such changes cannot be predicted in advance. For example, an e-commerce platform could use adaptive AI to recommend products to users based on their past behavior and evolving preferences.

Many GenAI systems, such as LLM-based chatbots, are deployed as locked models at runtime but are updated periodically, placing them between fully locked and fully adaptive systems.

In practice, AI-based systems span a continuum from fully deterministic, locked-down systems that produce the same output for a given input, to deliberately non-deterministic, self-learning systems that adapt and evolve their behavior.

Locked AI-based systems are far easier to test than adaptive AI-based systems, as they are largely deterministic and thus the expected results do not change. An updated, locked AI-based system is typically considered a new system, and a fresh round of testing is required. Note, however, that while locked AI-based systems are generally considered deterministic, in practice, large neural networks can exhibit non-deterministic behavior due to factors such as floating-point precision limits and variations in hardware execution, particularly when using parallel computation or GPUs.

An adaptive AI-based system can be rigorously tested before deployment (e.g., by simulating changes in environmental conditions, testing the learning mechanism itself, or testing its ability to adapt appropriately

in controlled scenarios). However, this testing is more complex than for a locked AI-based system, as an adaptive system may change its behavior during the testing or as a result of the testing.

As new behaviors for an adaptive AI-based system cannot always be predicted, such unpredictable behaviors cannot be tested in advance, nor can test cases be prepared for them. An automated test suite, focused on the system's core functionality, can be built and executed whenever the system undergoes significant changes to verify that adaptations are safe.

Testing can also be used to check that the system's performance has not degraded beyond a certain threshold. This testing may be in response to significant system changes or as a part of ongoing monitoring.

4.1.2 Rationale for a Statistical Approach to Testing AI-Based Systems

Testing AI-based systems presents unique challenges due to their data-driven and probabilistic nature.

The reasons a statistical approach to testing is needed for testing AI-based systems include:

- **Non-Determinism** - AI-based systems are fundamentally probabilistic and therefore often exhibit non-deterministic behavior, meaning the same inputs may not always yield the same output. This can be due to stochastic elements in their architecture or to their implementation of probabilistic mappings learned from training data. Consequently, a single test, such as an instance where a model incorrectly classifies a cat as a dog, cannot accurately reflect the model's overall AI functional correctness. To confirm that test results reliably overcome this uncertainty, the test suite must be sufficiently large to provide statistically significant results.
- **Distributional Performance Evaluation** - AI models are trained on specific data distributions that do not exactly match their operational environment. To assess how a model will perform under real-world conditions, a statistically significant sample of scenarios from relevant operational data distributions must be tested. This confirms that the testing captures the operational variability in data and subsequent model behaviors.
- **Handling Uncertainty and Bias** - AI-based systems are susceptible to data biases and can produce confident but incorrect predictions. Statistical testing allows practitioners to quantify and analyze model accuracy, fairness, and AI robustness through performance metrics such as confidence intervals, hypothesis testing, and error analysis.
- **Regulatory and Safety Context** - In regulated industries (e.g., healthcare, transport), demonstrating that an AI-based system meets safety or fairness thresholds with high confidence is often required. Statistical methods support claims about a model's reliability across a broad range of scenarios, rather than just for specific examples.

A statistical approach to the testing of probabilistic MLS is provided in 6.1.3.

4.1.3 Test Oracles for AI-Based Systems

Testing AI-based systems can be challenging, particularly when determining expected results (the test oracle problem). These difficulties arise from various factors inherent to AI:

- **Probabilistic & Non-deterministic Nature**: AI outputs can vary even with identical inputs. While many systems (e.g., in supervised learning) have a single correct target value, the model's outputs are probabilistic and defining a strict passed/failed oracle often requires setting thresholds or tolerance ranges.

- **Exploratory Development & Incomplete Specifications:** AI development is frequently exploratory. System requirements may evolve, be incomplete, or simply missing, lacking the detailed specifications needed to generate precise expected results.
- **Complexity of Tasks:** AI-based systems often tackle tasks that are too complex for straightforward human verification, making manual checks of expected outputs impractical.
- **Subjectivity of Behavior:** The correctness of an AI-based system's behavior can be subjective. For instance, user expectations for virtual assistants can vary widely, complicating the establishment of universally agreed-upon expected results.
- **Self-Learning Systems:** These AI-based systems continuously update their internal models based on new data encountered post-deployment. This causes the AI-based system's 'correct' behavior to change over time so that system responses remain effective and appropriate even as the definition of 'correct' evolves over time. As a result, an initial set of expected results can quickly become invalid.

The test oracle problem with AI-based systems can be addressed in various ways:

- **Defining Output Boundaries:** Testers can use agreed acceptable ranges, distributions, specified limits, and tolerances, such as an autonomous car stopping within a maximum distance.
- **Defining Environmental Boundaries:** Testers must specify values for test environment conditions (e.g., lighting levels, temperature, network latency) to ensure outputs are predictable and repeatable.
- **Expert Consultation:** Domain experts can help define expected results, though their opinions might differ or be fallible.
- **Specialized Testing:** A/B testing, back-to-back testing, and metamorphic testing, among others, can assess AI-based systems by comparing behaviors or verifying properties, often without requiring explicit expected outputs for every case.
- **Proxy Oracles:** Use secondary systems or models (including other AI systems) to assess or validate outputs when direct expected results are unavailable, such as training a proxy model on labeled data to predict for unlabeled tests.

4.2 Testing Generative AI and Large Language Models

This section outlines practical methods for validating GenAI systems, covering black-box evaluation, red teaming, and hands-on techniques. It highlights challenges from diverse inputs, tunable parameters, and context windows, and addresses both functional and non-functional quality measures using benchmarks and targeted exercises. See CT-GenAI syllabus [CT-GenAI] for more details on using GenAI systems for testing.

4.2.1 Testing Generative AI

Testing GenAI involves evaluating the correctness, coherence, and creativity of its outputs, including the originality and novelty of results generated, and verifying that specified requirements, both functional and non-functional, are met. Since GenAI systems can produce text, images, video, and audio, test strategies must be adapted to assess quality characteristics of such content.

A common approach is black-box testing, where testers feed various inputs (prompts, images, partial data) into the system and assess the test results (see red teaming, 4.2.2). Factors such as clarity, originality, and adherence to domain-specific rules are considered. This approach, whether manual or automated, is particularly useful for end-user applications, such as chatbots or design tools, where user satisfaction depends on the usefulness and plausibility of the generated content.

A significant challenge in testing a GenAI model is the ‘input explosion problem’, in which inputs can be highly diverse and complex to control. For instance, there are optional system prompts and a user prompt, which can include enormous amounts of disparate data, and GenAI can also be accessed via an API. There are also multiple parameters to consider, such as temperature and maximum tokens, which will also affect the output. Additionally, the context window, which retains previous parts of a conversation, also affects the generated output.

In many situations, assessing the output may involve manual review; however, determining whether a test has passed or failed depends on qualitative evaluation criteria defined in the requirements. Alternatively, a second GenAI system can often be used to automatically determine the test result. For instance, the correctness of an image generated by a GenAI system can be checked by an image recognition system. Such approaches should be used carefully, as they may reproduce similar biases or errors.

Non-functional testing can be as necessary as functional testing for GenAI systems. This includes evaluating resource utilization during both inference and training to verify efficient operation and cost-effectiveness. Measures include CPU and GPU usage, memory consumption, network bandwidth, and response times.

Benchmark suites provide standardized evaluation frameworks for assessing the capabilities of GenAI. These curated datasets and associated tasks enable consistent comparison across different models, measuring various aspects, from language understanding to reasoning and coding abilities. By measuring GenAI against these benchmarks, areas for improvement can be identified in a systematic, reproducible manner.

4.2.2 Red Teaming

Red Teaming (RT) is a systematic, often black-box form of fault attack that probes an AI-based system to identify harmful capabilities, particularly in its outputs. Drawing inspiration from historical practices like military wargaming and NASA's tiger teams, AI RT involves ‘attacking’ an AI-based system, with the goal of deliberately causing the system to produce harmful or undesirable results, such as privacy violations, the expression of racist views or providing guidance on performing chemical, biological, radiological, or nuclear (CBRN) attacks. Once such capabilities are identified, they are used to update and strengthen the system, making it less prone to such outputs in the future. While RT can be applied to any AI-based system, it is especially critical for GenAI due to the vast range of possible inputs and outputs, which create an extensive ‘attack space’. RT typically covers the complete end-to-end AI-based system, but can be applied to only the AI model.

Many organizations focus RT on security and safety vulnerabilities; however, it can also be used to detect harmful capabilities in other areas, such as reliability, privacy, fairness, bias and the generation of misinformation. RT is increasingly becoming a regulatory expectation for some types of AI-based systems, as seen in frameworks such as the EU AI Act [EU AI Act]. As an adaptive, dynamic evaluation approach in which prompts can immediately be updated in response to outputs, RT complements static approaches like benchmarking by testing systems under extreme and unexpected conditions.

When RT is used for security evaluations, the system is tested to identify vulnerabilities to external attacks, including both non-AI and AI-specific security factors, such as indirect prompt injection attacks and the hiding of malicious content in documents used by RAG. While security RT tends to focus on

malicious inputs, for safety and other evaluations, the aim is often to identify how the system might create harmful outputs under ordinary use, such as generating unsafe medical advice, without any adversarial intent from the user.

RT is most effective when conducted before a system is deployed, but after initial internal quality assessments are complete. The core activity is often interactive prompting, where red teamers engage in multi-turn dialogues (e.g., 15-20 turns) to elicit defective or policy-violating behaviors. The approach typically involves:

- 1) Assembling a diverse team of testers to cover a wide range of perspectives and attack vectors.
- 2) Providing access to the AI-based system in a safe test environment.
- 3) Prompting the system to identify vulnerabilities, through open-ended exploration or using checklists.
- 4) Analyzing the identified failures to understand threats.
- 5) Creating datasets from these threats to support mitigations and system improvements.

To achieve comprehensive coverage by RT, organizations employ several strategies beyond small expert teams. These include manual approaches like crowd-sourced prompt generation, and automated approaches, such as where an LLM is used to generate numerous attack prompts, and the outputs are then checked by another LLM. Hybrid approaches combine the creativity of human testers with the scalability of automation.

RT, which is proactive and pre-deployment focused, complements Blue Teaming, which involves ongoing, real-time defensive monitoring and filtering of inputs to an operational AI-based system to protect it against attacks. Insights from RT can be used to enhance the monitoring and filters used in Blue Teaming.

4.2.3 Hands-on Exercise: Exploratory Testing of a Large Language Model

Students will perform exploratory testing of an LLM. Students will be provided with an exploratory testing session sheet focused on testing the LLM's ability to generate test cases using 2-value and 3-value boundary value analysis. As part of the session debrief, they will verify the correctness and completeness of the output.

4.3 Test Levels and Machine Learning Systems

MLS require specialized test levels to address unique ML risks. These are input data testing and ML model testing. Additionally, conventional test levels are still relevant, including component, component integration, system, and acceptance testing.

4.3.1 Test Levels for Machine Learning Systems

The non-AI components of an MLS can be tested using conventional test levels. In addition, ML models and MLS require further testing to address the specific risks associated with ML.

Two specialized test levels are used to address ML-specific risks:

- Input data testing (see Chapter 5): Concerned with the training data used for training an ML model and the production data used by an MLS to generate a prediction in the operational environment.
- ML Model testing (see Chapter 6): Concerned with the testing of the ML models, which are the ultimate output of the ML workflow (see 3.1.2).

The risks associated with ML cannot all be addressed using these two ML-specific test levels. The following test levels are also typically required, depending on the perceived risks:

- Component Testing: Applicable to any non-AI components, such as user interface, data pipeline and communication components.
- Component Integration Testing: Includes testing that the inputs from the data pipeline are received as expected by the model and that any predictions generated by the model are exchanged with the relevant system components (e.g., the user interface) and used correctly. Where AI is provided as a service (see 1.1.6), API testing of the provided service is performed as part of component integration testing.
- System Testing: Includes confirmation testing that the ML functional performance from the initial model testing is not adversely affected when the model is embedded within a complete system. This testing is especially important when the ML model has been deliberately changed (e.g., by compressing a DNN to reduce its size). Non-functional testing is also covered, for example, testing the performance efficiency of the time required to deliver a prediction from a complete AI-based system.
- System Integration Testing: Focuses on verifying the interfaces and data exchanges between the AI-based system and external systems or services, using an environment representative of operational conditions.
- Acceptance Testing: Where AI is used as a service, 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 achieved.

4.3.2 Risk-Based Testing of Machine Learning Systems

Risk-based testing should be applied to all systems, regardless of whether they contain AI components or not. Most regulatory frameworks, whether published or in development, require a risk-based approach to the development and management of AI-based systems. As AI-based systems pose unique risks, testing them differs from that of non-AI-based systems.

There is no standard way to categorize ML risks; however, ML-specific risks are associated with both the development of the MLS (project risks) and with the MLS itself (product risks). One way of categorizing these risks is to use the ML workflow and divide it into three main areas:

- Development - concerned with the ML algorithm, the development of the model, and the ML development framework. Example project risks include sub-optimal algorithm selection, poor selection of evaluation approach, and framework security vulnerabilities. See Chapter 7 for more details.
- Input data - concerned with providing training data to support ML and the provision of production data used by the model in its operational environment. Examples of product risks include biased

training data, data-pipeline defects, and unrepresentative training data. See Chapter 5 for more details.

- Model - concerned with the generated ML model. Example product risks include failure to achieve required ML functional performance measures, an overfitted model, and susceptibility to adversarial examples. See Chapter 6 for more details.

Another way of categorizing risks associated with AI is according to the quality characteristics defined in ISO/IEC 25059.

Several forms of testing are available that address these risks and are specifically designed for the testing of MLS. For instance, data pipeline testing, adversarial testing, and review of algorithm/model suitability. This syllabus covers several of these in chapters 5, 6, and 7.

5 Input Data Testing for Machine Learning Systems – 180 minutes

Keywords

Data pipeline testing, data representativeness testing, dataset constraint testing, input data testing, label correctness testing, review, testing for bias

AI-Specific Keywords

Disparate impact analysis, multiple annotation

Learning Objectives for Chapter 5:

5.1 Input Data Testing for Machine Learning Systems

- AI-5.1.1 (K2) Give examples of test approaches used for the risk mitigation of input data for a machine learning system
- AI-5.1.2 (K2) Explain how to test for bias
- AI-5.1.3 (K2) Summarize the various forms of data pipeline testing
- AI-5.1.4 (K2) Explain how to test for data representativeness
- AI-5.1.5 (K3) Apply dataset constraint testing
- AI-5.1.6 (K2) Explain label correctness testing
- HO-5.1.7 (H2) Perform input data testing for ML datasets

5.1 Input Data Testing for Machine Learning Systems

The objective of input data testing is to confirm that the data used by the MLS for training, testing, and prediction is of sufficient quality (see 3.2). It includes reviews, statistical techniques (e.g., testing data for bias), EDA of the training data, and static and dynamic testing of the data pipeline.

5.1.1 Input Data Risks and Mitigations

The following table lists examples of input data risks and corresponding testing that could be used for risk mitigation:

Potential Risks	Possible Risk Mitigations
Defects in the training data leading to bias Problems within the algorithm, model, or ML development framework that introduce systemic unfairness	Testing for bias – see 5.1.2
Training data acquired from untrustworthy sources Data is poorly managed	Data provenance testing
Poisoned training data	A/B testing – see 6.1.9 Data provenance testing EDA – see 3.2.1 Attacks as part of red teaming – see 4.2.2
Internally inconsistent dataset Out of range data Wrong data types	Dataset constraint testing – see 5.1.5
Sub-optimal feature selection	Feature testing
Imbalanced dataset due to insufficient coverage of all target classes Skewed dataset through data augmentation Missing data	Data representativeness testing – see 5.1.4

Training data focused on a subset of all use cases Full range of values not covered in the dataset	
Poor labelling guidelines Ambiguous data Poor annotation leading to inaccurate or inconsistent labels	Label correctness testing – see 5.1.66
Poor design or integration leading to data pipeline failures Data quality defects compromising data pipeline outputs Performance degradation in the operational use of the data pipeline Security breaches or uncontrolled changes impacting the data pipeline	Data pipeline testing – see 5.1.3

5.1.2 Testing for Bias

Bias in an MLS refers to non-random, unfair differences in treatment based on sensitive attributes like gender, age, or race, which is often unlawful and renders the system discriminatory.

Testing for bias in an MLS involves understanding its potential sources, which primarily include:

- Defects in the training data, such as unrepresentativeness, historical skews, or deliberate poisoning (data bias).
- Defects within the algorithm, model, or development framework that introduce systemic unfairness, such as an algorithm using a decision threshold for credit scores in a loan approval system (algorithmic bias).

Test approaches for detecting bias include the following:

- Reviews of the overall ML workflow, and especially the data preparation, to identify their potential for introducing bias.
- Reviews of dataset documentation to identify and mitigate potential sources of unfairness by examining how the data was collected, annotated, and what populations are represented.
- Static analysis of both data preparation programs and the model implementation code to identify anti-patterns or the mishandling of sensitive attributes.
- EDA of training data by using visualization and clustering methods to reveal imbalanced data, skewed distributions, or anomalous groupings across different sensitive attributes.

- Dynamic testing to help detect bias in an ML model by feeding a known unbiased and representative dataset through the system and analyzing its predictions for statistically significant differences in outcomes across various sensitive groups. This identifies bias in model outputs, regardless of whether the bias was introduced during data training or during model development.
- Label correctness testing (see 5.1.6) to identify mislabeling that causes incorrect associations to be learned between attributes and outcomes for specific sensitive attributes.
- Disparate impact analysis:
 1. Identify sensitive attributes for the model.
 2. Generate counterfactuals for the sensitive attributes. (e.g., example scenarios where gender is changed from male to female in a loan application).
 3. Generate model output by presenting it with the counterfactuals.
 4. Analyze the results of multiple tests to achieve a statistically significant result, which determines if changing a sensitive attribute causes the model results to change, signaling the presence of bias.

Note: Disparate impact analysis can also be applied to combinations of sensitive attributes, allowing hidden bias associated with attribute combinations to be identified. However, take care to confirm that counterfactual examples are not unrealistic, as the model may then respond to them rather than to any underlying bias.

5.1.3 Data Pipeline Testing

Effective data pipeline testing is essential not only for confirming the reliability and performance of data-driven systems but also for maintaining high data quality throughout the entire ML workflow (see 3.1.2).

A layered approach is employed, beginning with data-pipeline design reviews during the design phase.

Component testing includes testing of data ingestion components, transformation scripts, and sensor interfaces. This testing uses code reviews, static analysis, and hardware-specific tests to verify reliable data capture. Component tests validate data transformation logic, test the implementation of data validation rules, confirm robust error handling, and test for vulnerabilities that could be exploited to introduce malware or poisoned data.

Component integration testing verifies the seamless flow of data across internal interfaces and the correct interpretation of data as it moves through the pipeline. It detects defects arising from mismatched interfaces or incorrect assumptions between components.

System testing assesses the fully assembled pipeline, beginning with smoke testing to confirm basic pipeline functionality. Functional testing verifies the pipeline's compliance with specified requirements, including data transformations and data routing. Non-functional testing evaluates performance under load, scalability to handle increasing data volumes, and security measures to protect data integrity. Fault injection testing measures pipeline robustness by simulating defective data inputs, assessing the system's ability to maintain data integrity when faced with unexpected or corrupted data. Back-to-back testing (see 6.1.10) compares the operational pipeline against the training pipeline to verify consistent functionality.

System integration testing verifies correct interaction between the data pipeline and external systems or services, including data sources, storage platforms, monitoring tools, and downstream consumers such as ML models.

Testing in production is performed on the operational system. Back-to-back testing verifies that performance is consistent or improved compared to previous versions. A/B testing (see 6.1.9) enables comparisons of new pipeline iterations against baselines, validating improvements while confirming no degradation in the live data stream. Tools can be integrated to continuously monitor and observe model behavior, performance, and potential defects in real time.

Configuration management reviews help verify that the correct pipeline code versions, configurations, and datasets are used across training, testing, and production.

Finally, the test strategies must align with the pipeline's purpose. Training pipelines, often exploratory prototypes, have different priorities than robust operational pipelines. The testing of training pipelines may focus on data integrity, while operational pipeline testing prioritizes reliability, performance, and maintainability.

5.1.4 Testing for Data Representativeness

Data representativeness testing determines how closely the characteristics of the datasets used for training, validation, and testing ML models match the real-world data that the ML model will encounter in operation. This testing addresses various forms of misrepresentation, including skewed datasets, missing data, disproportionate feature distributions, inadequate coverage of operational scenarios, and imbalanced class representation (see 5.1.1).

This testing typically includes the following steps:

1. Define the target population:

- Understand the intended use cases and operational context of the MLS
- Analyze the characteristics of end users and operational environments
- Identify the expected operational data distributions and critical edge cases by:
 - consulting domain experts to understand real-world data patterns
 - analyzing data from existing systems or similar applications
 - using benchmark datasets from trusted sources (e.g., NIST, industry databases)
- Apply stratified sampling to the representative reference data to create a baseline that covers all relevant subgroups in the target domain

2. Analyze data characteristics:

- Apply EDA (see 3.2.1) to:
 - training/test datasets being evaluated for representativeness
 - reference dataset that represents the expected operational data
 - visualize distributions through histograms, scatter plots, and other graphical techniques

- Examine feature relationships, and correlations in particular, to identify patterns that should be preserved
- Identify potential anomalies, gaps, or unusual concentrations in the data

3. Apply statistical assessment techniques:

- Use formal statistical tests such as Chi-squared and Kolmogorov-Smirnov to compare distributions [STATS]
- Check for data imbalances, particularly in classification problems
- Verify adequate coverage of both typical scenarios and edge/boundary cases

Testing for data representativeness should be performed before model training to prevent building models on unrepresentative data. After deployment, the properties of operational input data should be continuously monitored to detect changes that might indicate data drift from the original training data distributions (see 6.1.7 on drift testing).

5.1.5 Dataset Constraint Testing

Dataset constraint testing verifies whether the data in a dataset adheres to predefined rules or constraints. The goal is to confirm the integrity and consistency of the data used in ML. For instance, a test of the consistency of values in a dataset could be carried out.

Constraints on data are typically found in database schemas. A database schema defines the structure, types, and relationships of the data. Similarly, for ML datasets, a set of constraints can be defined that act as a logical model of the data in the dataset, which should be satisfied if the data are correct. There are several ways of categorizing dataset constraints. A constraint can be applied to a single value for an attribute, which is found in a single instance (a single-value constraint), for example:

- Missing – tests for missing values and missing attributes.
- Range – tests that a value is in a given range.
- Type – tests that an attribute value matches the specified type (e.g., that if an attribute is specified as an integer, the provided value is not a string or real number).

Alternatively, a constraint can apply across multiple values, typically considering the values for a single attribute across several instances (a multi-value constraint), for example:

- Sum – tests that the sum of all values equals, exceeds, or does not exceed a specified value (e.g., the total value of all points awarded for a Formula 1 race cannot exceed 102, and must exceed 50.5 points).
- Count – tests that the count of all non-null values for attributes or instances equals, exceeds, or does not exceed a specified value.
- Duplicate – tests for identical or near identical attribute values or instances in a dataset and enforces a limit on how many are allowed (often zero).
- Useful – tests that an attribute contains some repeated values, as if every entry is unique (like an ID or timestamp), typically provides no useful patterns to be learned for an ML model.
- Outlier – identifies any values that could be considered as statistical outliers.

A special form of multi-value constraint can compare different values (a comparison constraint), for example:

- Greater Than – tests that one value for an attribute is greater than a value for a second attribute (e.g., the count of lines of code for a program exceeds its count of lines of code with defects).
- Correlate – tests that the values for one attribute correlate with the values for a second attribute (e.g., all students with a marks attribute value at least 1.33 standard deviations above the mean also have a value of 'A' for their grade attribute).

Testing data against the defined constraints can be performed manually. Still, the scale of the activity and the typically large size of the dataset would require this to be automated as part of the data pipeline. When implemented as part of the pipeline, the tool that implements dataset constraint testing can provide reports to data scientists (for training data anomalies) or to operations staff (for operational data problems).

5.1.6 Label Correctness Testing

Data label correctness is essential in supervised learning. Inaccurate or inconsistent labels directly undermine the performance and generalization of ML models.

Common approaches to data label correctness testing include:

- Expert Review: Domain experts or trained annotators manually review a sample of labelled data. They evaluate label accuracy using their domain knowledge and guidelines.
- Multiple Annotation: Data points are independently labelled by multiple annotators and compared using a form of back-to-back testing (see 6.1.10). Disagreements highlight defects that need investigation and resolution. The inter-annotator agreement (IAA) can be measured using metrics such as Cohen's Kappa or simple percentage agreement [STATS]. Low IAA scores can indicate defects in labelling guidelines, ambiguous data, or poor annotation.
- Risk-Based Prioritization for Review and Annotation: Both expert reviews and multiple annotations can be focused using a risk-based approach to identify samples for review and multiple annotations. Prioritization can be based on the likelihood of mislabeling, such as with ambiguous data points (e.g., it is unclear to which category they belong or are near a boundary between classes), and the data that is most likely to impact the application's success or safety.
- Data Distribution Analysis: When comparable datasets exist, comparing the label distribution of the dataset under test to similar datasets can reveal anomalies and potentially incorrect labels.
- Automated Rule-Based Tests: Automated tests can be implemented based on predefined label rules or constraints for certain tasks. For instance, verifying that bounding boxes (rectangles used to locate objects in images) do not overlap or extend beyond image boundaries.
- Model Loss Analysis: Data points exhibiting high loss during model training – meaning the model's predictions for these points deviate substantially from their true labels – can indicate mislabeling. High loss reflects significant error, indicating the model struggles to learn the assigned label and pointing to a potential defect.
- Model Confidence Score Analysis: Data points with low prediction confidence from a trained model may be mislabeled, ambiguous, or lie outside the model's training data distribution.

Using a combination of approaches is often most effective. For example, expert reviews are valuable in establishing clear labelling guidelines early on. IAA scores from multiple annotations can then refine the

labelling process. Subsequently, model-based approaches can further identify potential label defects as the model continues to develop.

5.1.7 Hands-on Exercise: Input Data Testing

For a given dataset (e.g., structured, tabular data), perform input data testing for things like missing data, duplicate data, and outliers.

6 Model Testing for Machine Learning Systems – 225 minutes

Keywords

A/B testing, adversarial testing, back-to-back testing, concept drift, data drift, drift testing, metamorphic testing, ML functional performance, ML model testing, review

AI-Specific Keywords

Overfitting, underfitting

Learning Objectives for Chapter 6:

6.1 Model Testing for Machine Learning Systems

- AI-6.1.1 (K2) Give examples of test approaches used for risk mitigation of ML models
- AI-6.1.2 (K2) Explain the purpose and focus of reviewing ML model documentation
- AI-6.1.3 (K2) Explain how ML functional performance testing is carried out for probabilistic machine learning systems
- AI-6.1.4 (K2) Summarize adversarial testing of machine learning systems
- AI-6.1.5 (K3) Use metamorphic testing to derive test cases for a given scenario
- HO-6.1.6 (H2) Apply metamorphic testing
- AI-6.1.7 (K2) Explain how drift testing is used on operational machine learning systems
- AI-6.1.8 (K2) Explain how overfitting and underfitting are detected by testing
- AI-6.1.9 (K2) Explain how A/B testing is used in the context of machine learning systems
- AI-6.1.10 (K2) Explain how back-to-back testing is used in the context of machine learning systems

6.1 Model Testing for Machine Learning Systems

ML model testing involves addressing specific risks. These include functional risks such as bias, overfitting, and adversarial vulnerabilities, as well as non-functional risks such as lack of AI robustness and performance efficiency, and deployment risks.

6.1.1 Machine Learning Model Risks and Mitigations

The following table lists examples of ML model risks and corresponding testing that could be used for risk mitigation:

Potential Risk	Possible Risk Mitigation
Biased or unfair ML model	Testing for bias – see 5.1.2
Unethical model	Ethical system testing
Adversarial examples	Adversarial testing – see 6.1.4
Overfitted model	Testing for overfitting – see 6.1.8
Underfitted model	Testing for underfitting – see 6.1.8
Unacceptable data drift Unacceptable concept drift	Drift testing – see 6.1.7
Model causes side-effects	Side-effects testing
Model exhibits reward hacking	Reward hacking testing
Model API defect	API testing – see 7.1.2
Failure to achieve required ML model performance measures (e.g., lack of accuracy, recall)	ML functional performance testing – see 6.1.3
Functional incorrectness Non-functional defects	Metamorphic testing – see 6.1.5
Test oracle problem	Metamorphic testing – see 6.1.5 Back-to-back testing – see 6.1.10 A/B testing – see 6.1.9
Poor system requirements	Requirements review Red teaming – see 4.2.2

	Exploratory testing
Lack of model AI robustness due to unexpected inputs	Adversarial testing – see 6.1.4 Fuzz testing
Inadequate ML model performance efficiency	Performance testing
Poor model documentation (e.g., function, accuracy, interface)	Model documentation review – see 6.1.2
Model updates introduce defects	Back-to-back testing – see 6.1.10
Model updates decrease ML model functional performance	A/B testing – see 6.1.9
Deployment of updated model causes immediate failure	Smoke testing
Deployment of updated model causes regression	Regression testing
Security vulnerabilities Safety vulnerabilities Privacy violations Harmful or undesirable outputs (e.g., racist views, dangerous guidance)	Red teaming – see 4.2.2

6.1.2 Machine Learning Model Documentation and Review

Comprehensive documentation for ML models is not just a formality; it is a critical resource. Unlike systems where source code can be inspected directly, MLs present unique challenges due to the limited understandability of machine-generated code, the inherent black-box nature of models, and their dependency on data. Models are also frequently updated, so documentation becomes the main tool for developers, testers, and regulators to understand, evaluate, and trust AI-based systems. Transparency plays a central role in enabling this understanding, allowing stakeholders to trace model behavior, decision logic, and data lineage throughout the AI lifecycle.

Standardized documentation improves communication, supports informed decision-making, and verifies the quality and maintainability of ML models. It is increasingly important for regulatory compliance, as evidenced by frameworks such as the EU AI Act, which emphasizes transparency obligations that require clear documentation of model decisions, limitations, and interpretability measures. For high-risk systems, passing a documentation audit is often a prerequisite for deployment, while for all systems, a thorough review helps verify quality.

Several documentation frameworks exist, including Model Cards, which provide concise overviews of a model’s intended uses, evaluation results, and ethical considerations [MODEL_DOC], and Datasheets for Datasets, which offer standardized formats for describing datasets, including their motivation, composition, collection process, and uses [DATA_DOC].

The following list outlines the typical contents expected for comprehensive ML model documentation, structured so that it can be used as a practical checklist by both developers and testers to help achieve completeness, clarity, and testability:

- General: Identifiers, description, developer, version, date, contact, license, hardware needs
- Design: Assumptions, technical decisions, ML algorithm
- Usage: Intended purpose, primary/secondary uses, users, self-learning approach, bias, ethics, safety, transparency, thresholds, platform, data drift, concept drift
- Datasets: Features, source, collection, availability, preprocessing, use, content, labels, size, privacy, security, bias/fairness, restrictions
- Testing: Test dataset details, independence of testing, test results, test activities (e.g., functional, adversarial)
- Functional: Measures, validation dataset, thresholds, actual performance.
- Non-Functional: Scalability, reliability, availability, performance efficiency (e.g., latency, resource utilization), maintainability, AI robustness.
- Operational: Deployment plan, deployment environment, computational resources, monitoring metrics/alerts, retraining strategy, model update/rollback plan, deprecation plan, security (adversarial risks), explainability methods

Reviewing documentation against such checklists is a core test activity, aiming to:

- Identify missing information, inaccuracies, and inconsistencies.
- Improve clarity and readability.
- Enhance maintainability by identifying where documentation needs improvement.
- Provide sufficient information for test and deployment activities.
- Verify that all relevant regulatory requirements have been met.

6.1.3 ML Functional Performance Testing of Probabilistic Machine Learning Systems

ML functional performance testing evaluates how well an ML model performs its intended functions by measuring metrics such as accuracy, recall, precision, and F1-score (see 3.3) and comparing the results against defined acceptance criteria.

This testing for probabilistic MLS moves beyond a single passed/failed status to statistically measure a model's performance against its acceptance criteria [STATS]. This approach is necessary to address the non-determinism inherent in MLS by evaluating behavior across a large, representative dataset (see 4.1.2).

A precondition for this testing is having acceptance criteria defined in statistical terms. Instead of a simple target, a requirement might specify a performance metric, a margin of error (MoE), and a confidence level (CL). Such a requirement can be used to determine the minimum number of tests required. As the number of tests increases, there are two options:

- Fix the CL and the MoE will decrease. This means the test result becomes more precise.

- Fix the MoE and the CL will increase. This means the test result becomes more certain.

For example, a criterion might be “98% accuracy with a MoE of $\pm 4\%$ at a 95% CL”. To confirm that the measured accuracy has a maximum MoE of $\pm 4\%$ at the 95% CL, a sample size of 601 test cases is required. This is calculated using the sample size formula for estimating a population proportion. This sample size is based on the conservative assumption that the true accuracy could be anywhere between 0% and 100%, which produces the largest possible uncertainty but still guarantees the $\pm 4\%$ MoE under all conditions. To meet the target accuracy of 98%, at least 589 of the 601 test cases must pass. If, after running all 601 test cases, the observed accuracy is 98%, then the measured MoE at the 95% CL will be narrower than $\pm 4\%$ (approximately $\pm 1.1\%$), because variance is lower at high accuracy levels. The MoE is computed using the margin of error formula for a sample proportion. This means that when testing, it is not always necessary to run all 601 test cases. Sequential testing provides a formal statistical framework for this early stopping. Instead of always running the full fixed sample, these approaches analyze results as they accumulate and stop early when sufficient evidence supports the accuracy target (or rejects it). If the observed accuracy remains consistently high (e.g., no less than 98%), then the statistical uncertainty decreases as more tests are run. In that situation, the required MoE of $\pm 4\%$ at the 95% CL is reached after about 170 test cases, allowing testing to conclude earlier.

NOTE: The formulae and numerical calculations in this example (for sample size, margin of error, and confidence intervals) are provided for illustration and understanding only; candidates will not be required to derive or compute these values using statistical formulae in the exam.

For safety-critical systems, a stricter reliability requirement might be used, such as “99% reliability with 95% confidence,” which would require 299 test cases, all of which must pass to meet the criteria.

To validate these criteria, testing requires a large test dataset that is completely independent of the training and validation datasets. This test dataset must be a representative sample of the operational input domain (see 5.1.4) to verify that the evaluation reflects real-world conditions. The test cases are then executed using the ML model within an ML development framework capable of statistical analysis.

Finally, the aggregated results are interpreted and reported with statistical confidence, not as a simple passed/failed ratio. A final test report would state, for example, “The model achieved an accuracy of 94% $\pm 4\%$ at the 95% CL.” This allows stakeholders to understand the range of expected operational performance and make an informed decision on whether the model’s ML functional performance is acceptable for deployment.

6.1.4 Adversarial Testing of Machine Learning Systems

Adversarial testing involves deliberately providing the model with perturbations to the input data that are often imperceptible to humans. These inputs are designed to cause the model to make incorrect predictions, and, if successful, are called adversarial examples. Thus, the test inputs for adversarial testing, and thus potential adversarial examples, often consist of slightly modified versions of legitimate inputs that cause the model to misclassify them.

Identification of vulnerabilities through adversarial testing allows developers to incorporate safeguards and make the model more robust against adversarial examples. These may be accidental adversarial examples it encounters, or they may be adversarial attacks that involve the malicious use of adversarial examples. Generating effective test inputs for adversarial testing is technically complex, and staying up-to-date with evolving attack techniques is an ongoing challenge.

Adversarial testing can be performed using black-box testing, focusing on the model’s input and output behavior without requiring knowledge of its internal workings. This can be achieved by creating an equivalent model for which the internals are known, from which adversarial test inputs can be created

through knowledge of its internal workings. Then, based on the assumption that equivalent models share classification boundaries (i.e. transferability), the adversarial tests can be applied to the original model. In contrast, a brute force approach using a vast number of tests can be used in the hope that some random tests will coincide with an adversarial example.

Adversarial testing can also be performed using white-box testing. Understanding an ML model's internals (architecture, parameters, training process) typically makes it easier to craft adversarial examples.

Adversarial testing can be done manually by crafting specific adversarial examples or through automated algorithms that generate large numbers of variations to find effective adversarial inputs.

6.1.5 Metamorphic Testing

Metamorphic testing (MT) is a test technique in which new (follow-up) test cases are derived from a previously passed source test case. One or more follow-up test cases are generated by changing (metamorphizing) the source test case using a metamorphic relation (MR). The MR is based on a property of a required function of the test object, and it describes how a change in a test case's inputs is reflected in the same test case's expected results.

MT can be used for most test objects and can be applied to both functional and non-functional testing. Testers verify test objectives such as consistency (outputs align across related inputs), monotonicity (outputs change directionally with input), and invariance (outputs remain stable under perturbations). It is particularly useful where the generation of expected results is problematic because an affordable test oracle is unavailable. This is the case with some MLS that use big data, or with systems where testers are unclear about how the ML model derives its predictions, which is often the case. In the area of AI and ML, metamorphic testing has been used for testing image recognition, search engines, route optimization, and voice recognition.

MT is typically selected over traditional oracle-based testing when:

- no reliable expected outputs exist due to model opacity or data scale;
- the system is a black box; or
- relational properties (not absolute values) suffice for confidence.

MT is often based on a source test case that passed, and it can also be useful when it is not possible to generate an expected result. For instance, where the program implements a function that is too complex for a human tester to replicate and to use as a test oracle, such as with some complex MLS. In this situation, MT can be used to generate test cases which, when run, will create a set of outputs where the relationships between the outputs (rather than their actual values) are checked for validity. With this form of MT, if the relationships between test outputs hold true, it provides improved confidence in the program. For example, a risk-assessment MLS that predicts an age at death, where increasing the number of cigarettes smoked should decrease the prediction (monotonicity).

Testers derive MRs from domain knowledge, requirements, or domain properties (e.g., laws of physics). MRs can be validated by expert review, running them on reference models and checking coverage of edge cases.

Incorrect MRs (e.g., overlooking complex interactions between variables) or incomplete sets of MRs can lead to false confidence. MT detects relational flaws but not all absolute errors, so it should be used in combination with other test techniques.

6.1.6 Hands-on Exercise: Apply Metamorphic Testing

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

- Deriving several MRs for a given MLS. These MRs should include some in which the expected results of the source and follow-up test cases are the same, and some in which they are different.
- Generating source test cases for the MLS. These do not have to be guaranteed to pass, but students should be reminded of the limitations of MT when source test cases that passed are unavailable.
- Using the derived MRs and generated source test cases to derive follow-up test cases.
- Running the follow-up test cases.

6.1.7 Drift Testing

Drift testing can be used to identify two forms of drift in operational MLS:

- Data drift - occurs when the statistical properties of the operational input data change over time. For instance, input data is now significantly different from the data the model is trained on, due to factors like shifts in user behavior or seasonality. For example, a spam filter encounters new types of phishing attacks that did not exist during its training.
- Concept drift - occurs when the relationship between the input data and the correct output changes over time. This means the model's originally learned patterns or rules no longer reflect the current reality. For instance, due to new financial regulations, a transaction type previously considered 'low-risk' might now be classified as 'high-risk'. The meaning of the data has changed, causing the model's learned decision boundaries to become outdated, leading to a decline in its predictive accuracy.

Dynamic drift testing relies on the availability of feedback from the users, which provides the current ground truth. This current ground truth is compared with the model's output, and the difference between the two is determined and compared against a threshold value. User feedback can be direct or indirect. For a film recommendation system, an example of direct feedback is when the user rates a recommendation. An example of indirect feedback for the same system would be extracted from the data on the films the user has watched.

Static drift testing is not dependent on the current ground truth but instead compares the statistical properties of the input and predicted output data distributions using a test such as Kolmogorov-Smirnov [STATS]. A significant difference in either of these distributions is an indicator that drift has occurred.

6.1.8 Testing for Overfitting and Underfitting

Overfitting and underfitting are two of the three possible outcomes encountered in ML models, with the third being a "right-fitting" model. Testing for overfitting and underfitting should occur during training, evaluation, and tuning.

Overfitting occurs when a model learns the training data too well, to the extent that it captures noise in the data rather than the underlying pattern. This results in poor generalization of new, unseen data.

To test for overfitting, the model's ML functional performance is evaluated on a separate test dataset that was not used during training. This test dataset should include some less common examples unlikely to

have been used during training. The model might be overfitting if it performs significantly worse on the test dataset than on the validation dataset.

Underfitting occurs when a model is too simple to capture the underlying structure of the data or when the training data does not contain features that reflect an important relationship between inputs and outputs, resulting in poor ML functional performance on both the training and validation datasets.

During testing, underfitting can be detected by evaluating the model's ML functional performance metrics such as accuracy, precision, recall, or F1 score. If these metrics are consistently low on both the training and validation sets, it suggests that the model is underfitting.

Visual inspection of the model's learning curves can also help detect underfitting. If the training and validation errors remain high and relatively close together, without significant improvement as training progresses, it indicates underfitting.

In summary, detecting overfitting and underfitting during testing involves evaluating the model's ML functional performance on validation data, analyzing ML functional performance metrics, and examining learning curves.

6.1.9 A/B Testing

A/B testing is an approach 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 that typically requires comparing test results from multiple test runs to determine differences between the programs.

A simple example of this approach is where two promotional offers are emailed to a marketing list divided into two sets. Half of the list gets offer A, and half gets offer B; 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 addressing the test oracle problem, typically using the existing system as a partial test 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 incorporated into the tests.

A/B testing can be used to test updates to an AI-based system, provided there are agreed-upon acceptance criteria, such as ML functional performance metrics, as described in 3.3. Whenever the system is updated, A/B testing is used to determine that the updated variant performs as well as, or better than, the previous variant.

Such a test 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 be tested using A/B testing. For instance, by 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 resulting system characteristics are compared with those before the change. If the system is improved, the change is accepted; otherwise, the system reverts to its previous state.

The most popular statistical techniques for A/B testing are the t-test, z-test, chi-squared test, and Mann-Whitney U test [STATS].

6.1.10 Back-to-Back Testing

Back-to-back testing offers a practical solution to the test oracle problem.

Back-to-back testing involves using an alternative version of the system as a reference point (a pseudo-oracle) and comparing its outputs with those of the system being tested when presented with the same inputs. This pseudo-oracle could be an existing system, or one developed specifically for testing, but this comes at a cost.

Ideally, the pseudo-oracle and the system under test should not share common software components. Otherwise, both systems might contain the same defect, causing their outputs to match even when both are wrong. This can be particularly problematic given the widespread use of reusable, open-source AI components in MLS development. For this reason, the pseudo-oracle is often developed by a different and ideally independent team, perhaps using different ML development frameworks, algorithms, or model settings. Sometimes, conventional software can serve as a pseudo-oracle if it solves the same problem.

When performing functional back-to-back testing, the pseudo-oracle only needs to match functional behavior. It does not need to meet the same non-functional requirements as the system being tested, potentially making it less expensive to build.

This test approach requires only generating test inputs, not expected results, since the pseudo-oracle provides the comparison point. These inputs can come from existing test cases, such as regression test suites, or can be automatically generated from training data, allowing for a large number of tests to be run if automated test execution is supported.

Back-to-back testing provides significant value when migrating an MLS to a new environment, such as moving from development to production, and when comparing test results across environments. Additionally, this test approach can reveal subtle defects in model behavior that might not be apparent through other test approaches, especially when comparing responses across a wide range of edge cases or unusual inputs.

One significant difference between A/B testing (see 6.1.9) and back-to-back testing is the use of A/B testing to compare two variants of the same MLS using ML functional performance metrics and statistical techniques, versus the use of back-to-back testing to detect defects.

7 Machine Learning Development Testing – 30 minutes

Keywords

ML development testing, ML functional performance, shadow testing

AI-Specific Keywords

None

Learning Objectives for Chapter 7:

7.1 Machine Learning Development Testing

AI-7.1.1 (K2) Give examples of test approaches used for risk mitigation of ML development

AI-7.1.2 (K2) Explain the various forms of ML system deployment testing

7.1 Machine Learning Development Testing

This chapter focuses specifically on addressing risks introduced by ML development tools, configuration choices, and deployment mechanisms, rather than the ML model itself. It covers test approaches and test types, such as API testing, ML functional performance testing, A/B testing, back-to-back testing, and reviews, to verify AI robustness and efficiency.

7.1.1 Machine Learning Development Risks and Mitigations

The following table lists examples of ML development risks and corresponding testing that could be used for risk mitigation:

Potential Risk	Possible Risk Mitigation
Incorrect or unintended use of library or framework APIs (e.g., TensorFlow, PyTorch)	API testing – see 7.1.2
Sub-optimal framework selection	Framework suitability review
Problems within the algorithm, model, or development framework that introduce systemic unfairness	Testing for bias – see 5.1.2
Defective framework installation or build	Smoke testing
Defective implementation of the evaluation by the framework	Reviews of framework evaluation code Cross-check framework evaluation results (e.g., against manual benchmarks)
Poor performance efficiency (e.g., framework is slow to respond)	Performance testing
Poor usability of the framework	Usability testing
Defect in a library used by the framework (e.g., defect in PyTorch) Defective algorithm implementation	ML functional performance testing – see 6.1.3 Back-to-back testing – see 6.1.10
Security vulnerabilities in the framework	Security testing

Poor user documentation for the framework	Framework documentation review
Sub-optimal algorithm selection	Algorithm suitability review A/B testing – see 6.1.9
Sub-optimal hyperparameter selection (e.g., network structure, learning rate)	ML functional performance testing – see 6.1.3 A/B testing – see 6.1.9
Defective allocation of data to training, validation, and testing datasets	Data allocation review
Poor selection of evaluation approach (e.g., k-fold cross-validation)	ML functional performance testing – see 6.1.3
Incorrect interpretation of test results due to the stochastic nature of the learning process	ML functional performance testing – see 6.1.3
Deployment defect (e.g., from generating a modified version for a target platform)	Smoke testing ML functional performance testing – see 6.1.3 A/B testing – see 6.1.9
Deployed model is incompatible with the operational environment	Smoke testing MLS deployment testing – see 7.1.2
Deployed model is no improvement over the current model	Shadow testing – see 7.1.2

7.1.2 Machine Learning System Deployment Testing

Deploying MLS involves several key test activities focused on verifying the AI-based system functions correctly and reliably in its target environment (e.g., cloud, edge device, mobile). Each of the following test types addresses specific risks during deployment:

- Installability testing - verifies that the MLS can be successfully installed, configured, and subsequently uninstalled across all supported environments. This includes testing system dependencies (e.g., GPU drivers), compatibility with frameworks, and successfully executing installation scripts.

- Rollback testing - verifies the system's capability to successfully revert to a previously stable and operational state following a degraded or failed deployment. This may only cover the model or cover the complete system (e.g., including the data pipeline). Note that rollback testing must be performed before deployment to confirm rollback readiness.
- Canary testing – validates new deployments by releasing an updated model to a small subset of production traffic (e.g., 5% of users). Real-time metrics, such as latency, accuracy, and error rates, are monitored to detect regressions before a full rollout.
- Shadow testing - runs a new model in parallel with the current production model in real-time, routing the same requests to both systems without affecting live responses. It allows comparing new and old models using live data in a controlled, low-risk environment and can uncover defects, such as performance regressions and data drift, before full deployment.
- Model conversion testing - verifies that an ML model retains acceptable predictive accuracy, consistent behavior, and operational efficiency (e.g., inference speed, memory usage) after being converted from its original training format to a deployment format suitable for the target production environment.
- Cross-device testing - verifies that the MLS performs correctly across its intended range of deployment targets, from mobile devices and edge devices to cloud servers.
- API testing - verifies that the MLS exposes well-defined, standards-compliant interfaces. It validates correct handling of inputs and outputs, error messages, and integration workflows with external systems, such as data feeds, clients, and the pipeline.

8 List of Abbreviations

Abbreviation	Description
AI	artificial intelligence
AlaaS	AI as a service
API	application programming interface
CL	confidence level
CNN	convolutional neural network
CPU	central processing unit
DL	deep learning
DNN	deep neural network
EDA	exploratory data analysis
FN	false negative
FP	false positive
GAN	generative adversarial network
GenAI	generative AI
GPU	graphics processing unit
HO	hands-on objective
IAA	inter-annotator agreement
kMNC	k-multisection neuron coverage
LIME	Local interpretable model-agnostic explanations
LLM	large language model(s)
LO	learning objective
ML	machine learning
MLS	machine learning system(s)
MoE	margin of error
NBC	neuron boundary coverage

NLP	natural language processing
RAG	retrieval-augmented generation
RNN	recurrent neural network
RT	red teaming
SVM	support vector machine
TN	true negative
TP	true positive

9 AI-Specific 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.
adaptive AI-based system	An AI-based system that adjusts its behavior in response to changes in its operational environment.
adversarial attack	The deliberate use of adversarial examples to cause an ML model to fail.
AI as a Service	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 model	A computer program that implements AI
AI-based system	A system that incorporates one or more AI components.
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.
artificial intelligence	The capability of an engineered system to acquire, process, create and apply knowledge and skills. (ISO/IEC TR 29119-11)
association	An unsupervised ML learning technique that identifies relationships and dependencies between samples.
augmentation	The activity of creating new data points based on an existing dataset.
Bayesian model	A statistical model that uses probability to represent the uncertainty of both model inputs and outputs.

bias	The systematic difference in treatment of certain objects, people or groups in comparison to others. (After ISO/IEC TR 24027)
big data	Extensive datasets whose characteristics in terms of volume, variety, velocity and/or variability require specialized technologies and techniques to process.
bootstrap technique	A resampling technique that repeatedly draws samples with replacement from a training dataset to estimate the ML functional performance criteria of an ML model.
chatbot	An application used to conduct a conversation via text or text-to-speech.
classification	An 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	An ML function that groups similar data points together.
clustering algorithm	A type of ML algorithm used to group similar objects into clusters.
concept drift	A change in the perceived functional performance of an ML model 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.
convolutional neural network	A type of deep learning model designed to process grid-like data such as images, enabling it to recognize spatial patterns and features through layered operations.
cross-prompt injection attack	An attack in which malicious instructions in one prompt or context segment disrupt an AI model's behavior in a different prompt, turn, or context segment.
data acquisition	The activity of acquiring data relevant to the business problem to be solved by an ML model.
data bias	A systematic error caused by inaccurate, incomplete, or unrepresentative training data that leads to unfair outcomes in ML models.

data drift	A change in the distribution of input data over time, which can negatively impact the functional performance of an operational ML model.
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 preparation	The activities of data acquisition, data preprocessing and feature engineering in the ML workflow.
data preprocessing	The activities of data cleaning, data transformation, data augmentation, and data sampling in the ML workflow.
dataset	A collection of data used for training, evaluation, testing and prediction in ML.
decision tree	A tree-like ML model whose nodes represent decisions, and whose branches represent possible outcomes.
deep learning	ML using deep neural networks to automatically learn complex features and representations from large datasets.
deep neural network	A neural network comprised of several layers of neurons. Synonym: multi-layer perceptron
deepfake	Synthetic media, such as video, audio, or images, created or edited using AI to convincingly mimic or impersonate real people or events.
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	Producing the same set of outputs and final state from a given set of inputs and starting state.
disparate impact analysis	A technique for detecting bias by comparing decisions on original scenarios with their counterfactual versions, where sensitive attributes are swapped.
edge AI	The deployment of AI models on local edge devices, enabling real-time processing near the source of data without relying on cloud-based systems.
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.
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.
explainable AI	The field of study related to understanding the factors that influence AI system outputs.
exploratory data analysis	An interactive, visual, and hypothesis-driven process to summarize, explore, and understand the main characteristics and patterns of data.
F1-Score	An ML functional performance metric used to evaluate a classifier which provides a balance between recall and precision.
false negative	An ML model prediction in which the model mistakenly predicts the negative class.
false positive	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)
feature testing	A test type to determine if an AI model training dataset contains an appropriate set of features.
foundation model	A large-scale ML model trained on large datasets using self-supervised learning, designed as a versatile base that can be fine-tuned or adapted to a wide range of tasks across different domains.
frontier AI	A general-purpose AI-based system that exceeds the capabilities of today's most advanced AI systems.
fuzzy logic	A type of logic based on the concept of partial truth represented by certainty factors between 0 and 1.
general AI	A type of AI that can match human cognitive abilities across most intellectual tasks.
generative AI	A type of AI that creates new content by learning patterns from existing data.

graphics processing unit	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.
intelligent agent	An autonomous program which directs its activity towards achieving goals using observations and actions.
inter-annotator agreement	The degree of consensus or similarity among the annotations made by different annotators on the same data. (ISO/IEC TS 12791)
large language model	A text-to-text generative AI system trained on very large collections of language data.
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.
locked AI-based system	A deterministic AI-based system with a fixed unchanging model which does not change its behavior once it is deployed.
ML algorithm	An algorithm used to create an ML model from a training dataset.
ML development framework	A software platform that provides tools and libraries to build, train, and deploy ML models.
ML function	Functionality implemented by an ML model, such as classification, ML regression or clustering.
ML 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)
MLS	A system that integrates one or more ML models.
ML workflow	The set of activities used to develop, deploy and operate an ML model.
model confidence score analysis	A technique that identifies data points with low confidence scores from training, which are indicators for mislabeled data points.

model loss analysis	A technique that identifies data points with high loss values during training, which are indicators for mislabeled data points.
multimodal model	An ML model designed to handle multiple types of data modalities, such as text, images, audio, and video.
multiple annotation	An approach in which labelled data points from multiple annotators are compared.
narrow AI	AI focused on a single well-defined task to address a specific problem. Synonym: weak AI (ISO/IEC TR 29119-11)
natural language processing	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. Synonym: artificial neural network (ISO/IEC 2382)
neuromorphic processor	An integrated circuit designed to mimic the biological neurons of the human brain.
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-determinism	A property of a system or process in which an outcome is not uniquely determined by its initial conditions.
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)
pretrained model	An ML model that has already been trained on a large, general-purpose dataset and can be reused or fine-tuned for specific tasks.
probabilistic	Behavior described in terms of probabilities, where outcomes are uncertain and described by likelihoods rather than certainty.
random forest	Ensemble ML technology for classification, ML 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.
recall	An ML functional performance metric used to evaluate a classifier, which measures the proportion of actual positives that were predicted correctly. Synonym: sensitivity (After ISO/IEC TR 29119-11)
recurrent neural network	A type of deep learning model designed to process sequential data, enabling it to recognize patterns and dependencies over time.
reinforcement learning	An approach in which an ML model, known as an agent, learns by using a trial-and-reward feedback loop with its environment to achieve specific objectives.
retrieval-augmented generation	An ML technique where a GenAI system dynamically improves its output by retrieving relevant external information to supplement its internal knowledge.
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)
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)
sensitive attributes	Characteristics that define legally or ethically protected groups and individuals that must be controlled to prevent discrimination.
sentiment analysis	The use of natural language processing and machine learning to identify and classify the emotional tone expressed in text.
stratified sampling	A technique to confirm if a sample proportionally represents different sub-populations within the overall population.
super AI	A type of AI that far exceeds human capabilities.
supervised learning	An approach to training an ML model by using a labeled dataset.
support vector machine	A supervised ML algorithm that finds the optimal hyperplane between data points for classification or ML regression tasks.
technological singularity	A point in the future when technological advances are no longer controllable by people. (After ISO/IEC TR 29119-11)
temperature	A setting controlling the randomness of GenAI outputs, with lower values generating more predictable results and higher values generating more creative results.
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.
transformer	A neural network architecture that processes sequential data to capture long-range dependencies, powering NLP tasks, computer vision and multimodal applications.
true negative	A prediction in which the model correctly predicts the negative class.
true positive	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 makes inaccurate predictions. (after ISO/IEC TR 29119-11)
unsupervised learning	An approach to training an ML model by 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.
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.

10 References

10.1 Standards

- **ISO/IEC/IEEE 12207** (2017), Systems and software engineering — Software life cycle processes
- **ISO/IEC 2382** (2015), Information technology — Vocabulary
- **ISO/IEC 22989** (2022), Information technology — Artificial intelligence — Artificial intelligence concepts and terminology
- **ISO/IEC TR 24027** (2021) Information technology — Artificial intelligence (AI) — Bias in AI systems and AI aided decision making
- **ISO/IEC 25010** (2023), Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model
- **ISO/IEC 25059** (2023), Software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality model for AI systems
- **ISO 26262-6** (2018), Road vehicles — Functional safety — Part 6: Product development at the software level
- **ISO/IEC TR 29119-11** (2020), Software and systems engineering — Software testing — Part 11: Guidelines on the testing of AI-based systems
- **ISO/IEC TS 42119-2** (2025) Artificial intelligence — Testing of AI – Part 2: Overview of testing AI systems
- **ISO/IEC 42119 series** (in development) Artificial intelligence — Testing of AI

10.2 ISTQB® Documents

[CTFL] ISTQB® Certified Tester Foundation Level v4.0.1, https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTFL_Syllabus_v4.0.1.pdf (accessed 29.07.2025)

[CT-GenAI] ISTQB® Certified Tester – Testing with Generative AI (CT-GenAI), <https://istqb.org/certifications/gen-ai/> (accessed 19.12.2025)

10.3 Glossary References

Reference for terminology used in this syllabus:

- ISTQB® Glossary <https://glossary.istqb.org/>

10.4 Books, Articles and Web Pages

- [COV_REF] An Overview of Structural Coverage Metrics for Testing Neural Networks, Usman et al, Aug 2022, [arXiv:2208.03407](https://arxiv.org/abs/2208.03407) (accessed 29.07.2025)
- [DATA_DOC] Gebru, T., Morgenstern, J., Vecchione, B., et al. (2021). Datasheets for Datasets. Communications of the ACM, 64(12), 86-92, <https://dl.acm.org/doi/pdf/10.1145/3502158> (accessed 07.10.2025)
- [EU AI Act] EU Artificial Intelligence Act, July 2024, https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L_202401689 (accessed 30.07.2025)
- [MODEL_DOC] Mitchell, M., Wu, S., Varma, R., et al. (2019). Model Cards for Model Reporting. Proceedings of the Conference on Fairness, Accountability, and Transparency, <https://dl.acm.org/doi/10.1145/3287560.3287596> (accessed 07.10.2025)
- [OECD AI] OECD Recommendation of the Council on Artificial Intelligence, May 2024, Organisation for Economic Co-operation and Development (OECD), <https://legalinstruments.oecd.org/en/instruments/oecd-legal-0449> (accessed 07.10.2025)
- [UN Gov AI] Governing AI for Humanity: Final Report, September 2024, United Nations, https://www.un.org/sites/un2.un.org/files/governing_ai_for_humanity_final_report_en.pdf (accessed 29.07.2025)
- [STATS] An Introduction to Statistical Learning: With Applications in R (2nd ed.) by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, Springer.

11 Trademarks

ISTQB® is a registered trademark of International Software Testing Qualifications Board

12 Appendix A – Learning Objectives/Cognitive Level of Knowledge

The specific learning objectives applying to this syllabus are shown at the beginning of each chapter. Each topic in the syllabus will be examined according to the learning objective for it.

The learning objectives begin with an action verb corresponding to its cognitive level of knowledge as listed below.

Level 1: Remember (K1)

The candidate will remember, recognize and recall a term or concept.

Action verbs: Recall, recognize.

Examples
Recall the concepts of the test pyramid.
Recognize the typical objectives of testing.

Level 2: Understand (K2)

The candidate can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify and give examples for the testing concept.

Action verbs: Classify, compare, differentiate, distinguish, explain, give examples, interpret, summarize

Examples	Notes
Classify test tools according to their purpose and the test activities they support.	
Compare the different test levels.	Can be used to look for similarities, differences or both.
Differentiate testing from debugging.	Looks for differences between concepts.
Distinguish between project and product risks.	Allows two (or more) concepts to be separately classified.
Explain the impact of context on the test process.	
Give examples of why testing is necessary.	
Infer the root cause of defects from a given profile of failures.	
Summarize the activities of the work product review process.	

Level 3: Apply (K3)

The candidate can carry out a procedure when confronted with a familiar task, or select the correct procedure and apply it to a given context.

Action verbs: Apply, implement, prepare, use

Examples	Notes
Apply boundary value analysis to derive test cases from given requirements.	Should refer to a procedure / technique / process etc.
Implement metrics collection methods to support technical and management requirements.	
Prepare installability tests for mobile apps.	
Use traceability to monitor test progress for completeness and consistency with the test objectives, test strategy, and test plan.	It could be used in an LO that wants the candidate to be able to use a technique or procedure. Similar to 'apply'.

Level 4: Analyze (K4)

The candidate can separate information related to a procedure or technique into its constituent parts for better understanding, and can distinguish between facts and inferences. Typical application is to analyze a document, software or project situation and propose appropriate actions to solve a problem or task.

Action verbs: Analyze, deconstruct, outline, prioritize, select.

Examples	Notes
Analyze a given project situation to determine which black-box or experience-based test techniques should be applied to achieve specific goals.	Examinable only in combination with a measurable goal of the analysis. Should be of form 'Analyze xxxx to xxxx' (or similar).
Prioritize test cases in a given test suite for execution based on the related product risks.	
Select the appropriate test levels and test types to verify a given set of requirements.	Needed where the selection requires analysis.

Reference

(For the cognitive levels of learning objectives)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

13 Appendix B – Business Outcomes Traceability Matrix with Learning Objectives

This section lists the traceability between the Business Outcomes and the Learning Objectives of the Certified Tester AI Testing.

The first part of the table shows the number of Learning Objectives per Business Outcome, while the second part shows which Learning Objectives are associated with each Business Outcome.

Business Outcomes: AI Testing			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
BO1	Understand the current state of AI, including generative AI.		8							
BO2	Experience the implementation and testing of machine learning models.			10						
BO3	Understand the working and testing of simple neural networks.				2					
BO4	Understand the specific AI quality characteristics defined by ISO/IEC 25059.					3				
BO5	Calculate and interpret ML functional performance metrics for machine learning models.						1			
BO6	Recognize the scope and importance of the two test levels that are specific to the testing of machine learning systems.							3		

Business Outcomes: AI Testing			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
BO7	Contribute to the development of an effective test strategy for a machine learning system.								5	
BO8	Design and execute test cases for machine learning systems.									14
Unique LO	Learning Objective	K-Level								
1	Introduction to Artificial Intelligence									
1.1	Introduction to AI									
AI-1.1.1	Differentiate between AI-based systems and conventional systems	K2	X							
AI-1.1.2	Distinguish between narrow AI, general AI, and super AI	K2	X							
AI-1.1.3	Explain the different types of AI technologies	K2	X							
AI-1.1.4	Explain generative AI	K2	X							
AI-1.1.5	Compare the choices available for hardware to implement machine learning systems	K2	X							

Business Outcomes: AI Testing			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
AI-1.1.6	Compare the options for the development and hosting of AI models	K2	X							
AI-1.1.7	Summarize the functionality provided by ML development frameworks	K2	X							
AI-1.1.8	Explain how regulations and standards affect the development and testing of AI-based systems	K2	X							
2	Quality Characteristics for AI-Based Systems									
2.1	Quality Characteristics for AI-Based Systems									
AI-2.1.1	Classify behaviors of AI-based systems according to the quality characteristics defined in ISO/IEC 25059	K2				X				
AI-2.1.2	Explain the special considerations that arise when AI is used in safety-related systems	K2				X				
2.2	Acceptance Criteria for AI-Based Systems									
AI-2.2.1	Give examples of acceptance criteria for AI-based systems	K2				X				
3	Machine Learning									

Business Outcomes: AI Testing			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
3.1	Introduction to Machine Learning									
AI-3.1.1	Distinguish between the different forms of ML	K2		X						
AI-3.1.2	Summarize the workflow used to create an ML system	K2		X						
AI-3.1.4	Summarize the use of pretrained models, fine-tuning and retrieval-augmented generation	K2		X						
3.2	Data for Machine Learning									
AI-3.2.1	Explain the activities related to data preparation	K2		X						
AI-3.2.3	Contrast the use of training, validation, and test datasets in the development of an ML model	K2		X						
3.3	ML Functional Performance Metrics for Classification									
AI-3.3.1	Calculate common ML functional performance metrics from a given set of confusion matrix data	K3					X			
3.4	Neural Networks									

Business Outcomes: AI Testing			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
AI-3.4.1	Explain the structure and working of a deep neural network	K2			X					
AI-3.4.3	Describe the different coverage measures for neural networks	K2			X					
4	Testing AI-Based Systems									
4.1	Introduction to Testing AI-Based Systems									
AI-4.1.1	Compare the testability of locked and adaptive AI-based systems	K2		X						
AI-4.1.2	Explain why a statistical approach is often needed when testing AI-based systems	K2		X						
AI-4.1.3	Explain the challenges and solutions relating to test oracles for AI-based systems	K2		X						
4.2	Testing Generative AI and LLM									
AI-4.2.1	Explain how generative AI can be tested	K2		X						
AI-4.2.2	Implement red teaming for GenAI systems	K3		X						

Business Outcomes: AI Testing			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
4.3	Test Levels and Machine Learning Systems									
AI-4.3.1	Summarize the test levels used to develop machine learning systems	K2							X	
AI-4.3.2	Explain how risk-based testing is applied to machine learning systems	K2							X	
5	Input Data Testing for Machine Learning Systems									
5.1	Input Data Testing for Machine Learning Systems									
AI-5.1.1	Give examples of test approaches used for the risk mitigation of input data for a machine learning system	K2						X	X	
AI-5.1.2	Explain how to test for bias	K2								X
AI-5.1.3	Summarize the various forms of data pipeline testing	K2								X
AI-5.1.4	Explain how to test for data representativeness	K2								X
AI-5.1.5	Apply dataset constraint testing	K3								X

Business Outcomes: AI Testing			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
AI-5.1.6	Explain label correctness testing	K2								X
6	Model Testing for Machine Learning Systems									
6.1	Model Testing for Machine Learning Systems									
AI-6.1.1	Give examples of test approaches used for risk mitigation of ML models	K2						X	X	
AI-6.1.2	Explain the purpose and focus of reviewing ML model documentation	K2								X
AI-6.1.3	Explain how ML functional performance testing is carried out for probabilistic machine learning systems	K2								X
AI-6.1.4	Summarize adversarial testing of machine learning systems	K2								X
AI-6.1.5	Use metamorphic testing to derive test cases for a given scenario	K3								X
AI-6.1.7	Explain how drift testing is used on operational machine learning systems	K2								X
AI-6.1.8	Explain how overfitting and underfitting are detected by testing	K2								X

Business Outcomes: AI Testing			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
AI-6.1.9	Explain how A/B testing is used in the context of machine learning systems	K2								X
AI-6.1.10	Explain how back-to-back testing is used in the context of machine learning systems	K2								X
7	Machine Learning Development Testing									
7.1	Machine Learning Development Testing									
AI-7.1.1	Give examples of test approaches used for risk mitigation of ML development	K2						X	X	
AI-7.1.2	Explain the various forms of ML system deployment testing	K2								X

14 Appendix C – Release Notes

The ISTQB CT-AI v2.0 is a major update and rewrite of v1.0. Due to rapidly evolving AI technology, a major update was necessary. The focus of v2.0 is clearly on testing of AI-based systems. Due to the release of the ISTQB CT Testing with Generative AI the chapter on testing with AI was completely removed.

This major release has made the following changes:

- Shortened introduction to AI in general
- Inclusion of Generative AI and testing of Generative AI
- Consolidation of AI quality characteristics and their challenges
- Reduced focus on ML performance metrics
- Refined test levels: Input data testing and ML model testing
- Refined test types
- Removed testing with AI
- Exclusion of test environments for testing AI-based systems
- Minimum required training time reduced from 4-days to 3-days

15 Index

All testing terms are defined in the ISTQB® Glossary (<https://glossary.istqb.org/>).

A/B testing, 62
accuracy, 35
adaptive AI-based system, 41
adversarial testing, 59
AI as a Service, 18
AI functional correctness, 23
AI robustness, 23
AI-based system, 15, 40
AI-based systems, 15
API testing, 67
artificial intelligence, 15
association, 28
attack, 43, 61
back-to-back testing, 63
canary testing, 67
classification, 28, 34
clustering, 28
concept drift, 61
confusion matrix, 35
data drift, 61
data pipeline testing, 50
data preparation, 32
data representativeness testing, 51
dataset constraint testing, 52
device compatibility testing, 67
disparate impact analysis, 50
drift testing, 61
dynamic testing, 48, 50
EU AI Act, 20
explainability, 24
exploratory data analysis, 29, 33, 49
exploratory testing, 44
F1-score, 35
fine-tuning, 31
follow-up test case, 60
frontier AI, 15
functional adaptability, 23, 25
functional correctness, 22, 25
general AI, 16
generative AI, 17, 43
input data testing, 45, 48
installability testing, 67
intervenability, 23, 26
k-multisection neuron coverage, 38, 68
label correctness testing, 50, 53
large language model, 32, 42
locked AI-based system, 40
machine learning, 15, 16, 28
metamorphic relation, 60
metamorphic testing, 60, 61
ML algorithm, 29
ML development framework, 19, 29
ML development testing, 65
ML functional performance, 58, 65
ML functional performance criteria, 30
ML functional performance metric, 34
ML functional performance metrics, 29
ML model, 18, 29, 31, 33
ML model documentation, 57
ML model testing, 56
ML regression, 28
ML workflow, 29, 30, 34
model conversion testing, 67
model testing, 45

multiple annotation, 53
narrow AI, 15
neural network, 36, 37
neuron boundary coverage, 38, 69
neuron coverage, 38
non-determinism, 24
non-functional test, 30, 60
overfitting, 62
perceptron, 38
precision, 35
pretrained model, 31
recall, 35
red teaming, 43
reinforcement learning, 16, 28
retrieval augmented generation, 32
review, 49, 57
risk-based testing, 46
robustness, 26
rollback testing, 67
safety, 24, 26
self-learning, 24
shadow testing, 67
societal and ethical risk mitigation, 22, 23, 26
source test case, 60
static analysis, 49
super AI, 16
supervised learning, 16, 28
test dataset, 30, 34
test oracle, 41, 60
testing for bias, 49
training dataset, 33
transparency, 23, 24, 25
underfitting, 62
unsupervised learning, 16, 28
user controllability, 23, 25
validation dataset, 29, 33