Test and Evaluation of Artificial Intelligence Models

What to Consider in a Test & Evaluation Strategy

April 2024

Cleared for Public Release

Table of Contents

T&E Strategies for AIECs

This Section:

+ Specifies the role of the current document within the larger framework.
+ Provides an overview of the framework for the test and evaluation of AI-enabled capabilities produced by CDAO Assess and Assurance.

00

This document is part of a framework for the T&E of AI-enabled capabilities

CDAO Assessment and Assurance is creating a framework to provide guidance on how to test and evaluate (T&E) AI-enabled capabilities (AIECs).

What is the framework?

The T&E of AIEC Framework provides best practices and guidance on how to test and evaluate AIEC.

The framework is organized into four categories of testing and provides different types of resources to AIEC developers and working-level testers.

Why is it needed?

The DoD community for the T&E of AIEC comes from a variety of backgrounds.

The T&E of AIEC Framework promotes a shared understanding between AIEC experts new to T&E and to T&E experts new to AIEC.

What is this document?

This document discusses the test and evaluation of AI models, both standalone and integrated into system-of-systems, in a Defense context.

It is intended to help AIEC developers and working-level testers incorporate operational realism into testing throughout an AIEC's lifecycle.

This document provides:	This document does NOT provide:
✓ **Guidance and best practices**	✗ **Binding policy and requirements**
✓ **A primer on T&E of AI models**	✗ **A comprehensive AI Model T&E guide**
✓ **Strategy-level T&E considerations**	✗ **Detailed T&E implementation**
✓ **T&E at the algorithm level**	✗ **T&E at the system-of-systems level**

CDAO's T&E of AIEC framework is organized into four focus areas

While these T&E focus areas help break critical aspects of T&E into digestible pieces, they are neither mutually exclusive nor cleanly delineated in real testing.

Operational T&E (OT&E)
Evaluating an AIEC performing representative missions within an operationally realistic environment against a realistic adversary.

Human Systems Integration (HSI) T&E
Evaluating an AIEC's ability to help stakeholders observe and orient to their environment, make informed decisions, and carry out their missions.

Systems Integration (SI) T&E
Evaluating an AI component within its larger system to ensure that the AIEC functions as a holistic unit and identify its limitations and risks.

AI Model T&E
Evaluating and documenting AI models and data across performance dimensions informed by system and mission constraints.

This document covers the AI Model T&E focus area

CDAO is developing a series of products that address critical T&E needs

Part 1 is designed to help testers understand core T&E concepts so that working-level testers can write and assess test and evaluation strategies for AI-enabled capabilities

This document focuses on Part 1

1 | Write and assess T&E Strategies

Provides a high-level overview of critical T&E concepts that will be influenced by the inclusion of AI models in the system under test.

Supports testers and developers as they write TESs and assess whether the TES is committed to the right evaluations.

2 | Write and assess Detailed Test Plans

Provides guidance for implementation of T&E concepts introduced in Part 1; highlights promising paths forward for unsolved challenges.

Supports testers and developers as they develop and implement detailed test plans that capture mission objectives.

3 | Engage with other DoD T&E stakeholders

Provides frameworks outlining how T&E is critical to fielding trustworthy AIECs across DoD acquisition pathways and mission applications.

Supports testers and developers as they advocate for policy and investments that address DoD T&E shortcomings.

4 | Execute tests and rigorously analyze results

Provides resources such as templates, validated measurement instruments, and automated analysis tools.

Supports testers and developers by streamlining and automating common T&E activities with tailorable tools.

What is a Test & Evaluation Strategy?

A high-level document in DoD acquisitions that guides test planning and execution.

Captures the mission(s) a capability is intended to perform and all hardware and interfacing systems in the test design.

Identifies and prioritizes assessment areas to inform test team data requirements to support major program decisions.

Specifies the resources required to conduct T&E and shortfalls in resourcing that will require investments.

Describes the test events and activities necessary to evaluate the system and support acquisition, technical, and program decisions.

Learn More

You can read more about DoD TESs at https://www.test-evaluation.osd.mil/T-E-Enterprise-Guidebook/

Thinking about Performance

This Section:

+ Describes the multiple dimensions of performance
+ Outlines how performance varies based on the AI model type
+ Describes common aspects and measures of performance

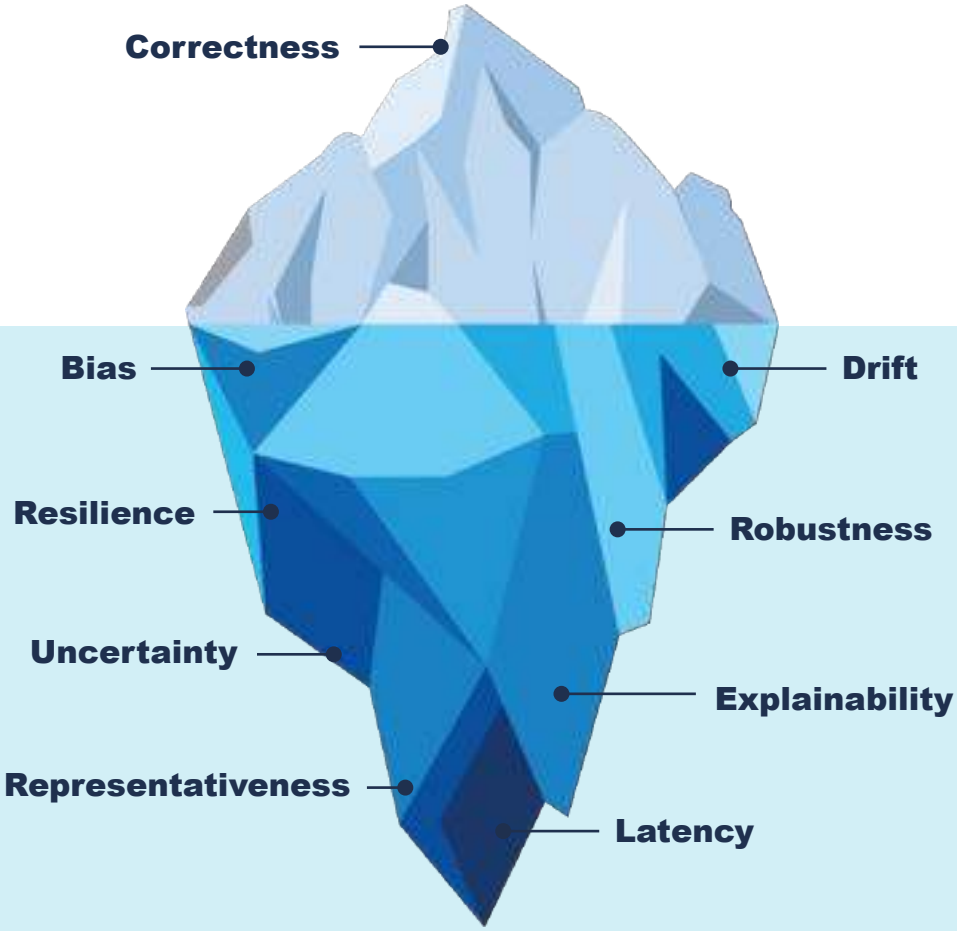
01

The Performance Iceberg

Correctness is just the tip of the iceberg when it comes to rigorously and robustly evaluating the performance of your AI model

Testing an AI model is vital for its quality, reliability, and usefulness. But ensuring testing is sufficiently robust is not simple, as many subtle aspects of performance require evaluation and validation. Correctness, the most visible and intuitive metric, shows how well a model achieves its functional performance goals. But correctness measurement alone is not enough for rigorous and robust performance evaluation.

There are many other aspects that are hidden below the water line, but they are vital for ensuring the quality and reliability of the model. These aspects include how the model handles different sources of error, such as bias and drift, how the model explains its output and reasoning, such as explainability and uncertainty, how the model responds to different situations and inputs, such as latency and robustness, and how the model represents the real-world problem and data, such as representativeness and resilience. These aspects are often interrelated and complex, and they need to be carefully considered and evaluated when testing an AI model. Testing an AI model is not a simple task, but a comprehensive and effective one.



Measuring Correctness

Correctness is the ability of a predictive model to fulfill its functional performance goals. Correctness (accuracy, precision, and/or recall) is frequently a main focus of model developers.

How do you measure it?

Correctness metrics vary by algorithm type. When available, correctness relates prediction results to ground truth. When not available, other comparisons are needed. Common metrics include:

- Classifiers: Accuracy, Precision, Recall, F1 score
- Regression: mean squared error, R^2
- Unsupervised and generative models: often rely on human evaluators
- Reinforcement learning systems: a combination of simulator scores and human evaluation

For classifier models, "Accuracy" is defined:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

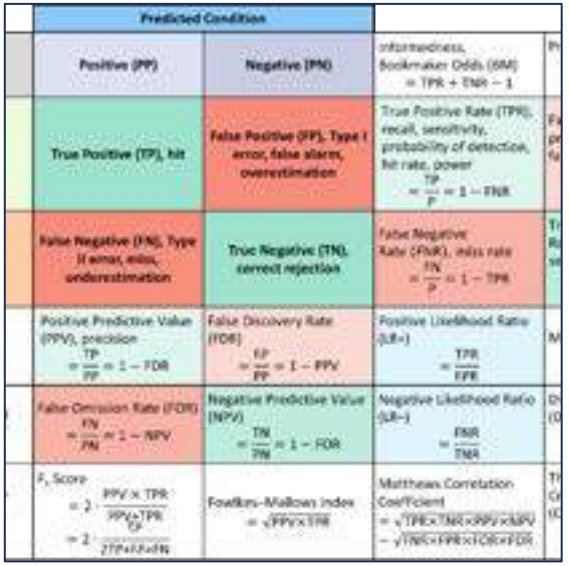
Why should you measure it?

Correctness on the intended task is typically the primary consideration for a model developer. Without a sufficient level of correctness, none of the other measures given here matter.

Correctness is the measure of how well a model predicts the correct or expected output for a given input. It is a basic and intuitive metric for assessing the model's performance on its intended task.

Correctness is important for identifying and improving the model's weaknesses and limitations. It helps to diagnose the sources of errors, such as data quality, algorithm design, or others.

Correctness is useful for comparing different models or model versions.



What should you ask?

? What is the expected output of the model for a given input?

? Does the developers' functional performance metric align with what matters most in operations?

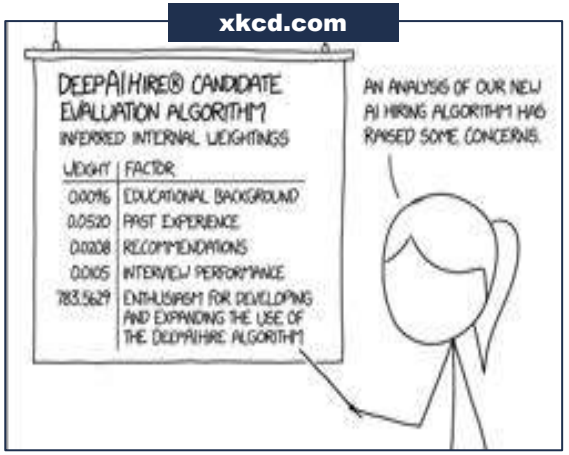
? Can the model achieve its correctness targets?

? How robust is the model under different conditions and scenarios?

Similar Terms functional performance, prediction accuracy, goodness of fit

Measuring Interpretability

Interpretability methods provide insight into how a model produced its output. These methods generally do not provide insight into the underlying data generating process that the model was trained on, but greater interpretability facilitates understanding the model's inner workings.



How do you measure it?

Intrinsic and/or Post-hoc? Intrinsic methods restrict the complexity of the model, while post-hoc methods analyze a trained model.

Local or Global? Does the interpretation method explain an individual prediction or the entire model behavior?

Model-Specific or Model-Agnostic? Model specific methods can be faster, but model-agnostic methods work with more model types.

Textual explanations: This method generates natural language descriptions that explain the model's output for a given input.

Decision trees: This method converts a trained model to show the rules behind the model's predictions.

Why should you measure it?

Detect bias – interpretable models can be checked for protected groups or their correlates in training data.

Auditable – interpretable models can be examined for purposes of debugging or suggesting theories for further testing.

User acceptance – humans rely on explanations to develop trust. Interpretable models can provide useful explanations.

Education – can help users and students learn from the model and know when to use the model, or not.

Adaptation – interpretable models can help users and developers adapt the model to changing needs, preferences, or environments.

What should you ask?

? Who needs to interpret the model?

? How will the interpretability measures be presented?

? How will the interpretation be used?

? What issues might confound the provided explanations?

? How does user feedback get incorporated into the model's interpretability?

Similar Terms explainability, SHAP (SHapley Additive exPlanations), mechanistic interpretability, transparency, decision boundary

Measuring Bias

"Bias" can mean any of several different concepts when applied to AI, including but not limited to: discriminatory or unfair treatment (legal/fairness), underrepresented elements in the data (representation), and differences in average outputs from the true mean (statistical).

How do you measure it?

There are too many concepts of bias to comprehensively list and define here. Some types of bias can be measured directly if ground truth is available, but Others must be inferred. Some cannot be simultaneously minimized.

Fairness bias often relates to correlations in the data; the chosen definition of fairness will inform measurement. Representation bias can occur when there is selection into the training sample, so measurement can be facilitated by distributional measurements. Ground truth is often not available, so statistical bias often must be inferred from prediction error.

Teams should involve stakeholders to identify and create strategies for measurement and mitigation, when appropriate.

Why should you measure it?

Models may exhibit good overall performance but still have unintended bias that may result in harm. Fairness bias can result in harm to individuals and groups – a violation of the DoD Ethical AI Principles. Representation bias can diminish operational effectiveness and suitability, and lead to fairness bias.

Unintended bias can lead to negative consequences, particularly for some subpopulation such as minorities, specific genders, professional groups, or other underrepresented ones. E.g.,

- Model outputs that systematically vary across groups
- Disparate treatment
- Social or economic harm
- Loss of trust

What should you ask?

? What bias is intentional?

? How does the intentional bias tie to mission objectives and performance?

? What undesirable biases might be reflected in your data and amplified by the model?

? What fairness metrics best align with your mission objectives?

Similar Terms Selection bias, representation bias, statistical bias, algorithmic bias, fairness, disparate treatment

Measuring Robustness

Robustness describes how well a model performs outside of conditions on which it was trained. Robustness is typically categorized into two areas: natural (natural variations in input data) and adversarial (perturbations caused by malicious attacks).

How do you measure it?

In evaluating robustness, testers should consider both the smoothness of the performance surface and the performance in specific situations relative to other models.

Natural robustness is evaluated through variation in evaluation data. This can be done by using different datasets or data sources that reflect the diversity and complexity of the real-world data. Metrics should capture how much the model performance degrades under different levels of variation.

Adversarial robustness is evaluated via manual or automated red teaming. Red teaming simulates attacks by an adversary who has some knowledge and access to the model. Metrics should capture how often the model is compromised by adversarial inputs.

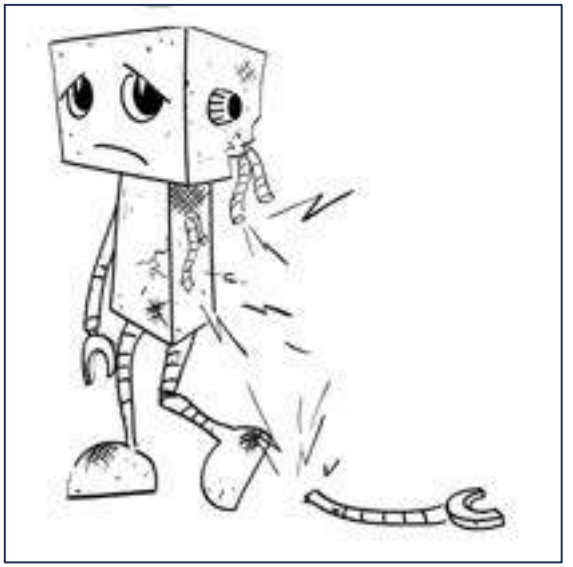
Why should you measure it?

AI models can perform unpredictably when deployed outside of the conditions in which they were trained due to emergent model behaviors and a failure to generalize in ways that conform to human expectations.

A lack of robustness can lead to undesirable or harmful consequences for users, stakeholders, or society at large, such as loss of trust, privacy breaches, discrimination, or physical harm.

By measuring robustness, testers can identify and mitigate potential risks and vulnerabilities of AI models before they are deployed in real-world settings.

Measuring robustness can lead to design decisions to enhance system quality and reliability, as well as better user experience.



What should you ask?

? How does the model handle errors or failures and recover from them?

? What threats or attacks might the model face from adversaries?

? How does the model handle red team attempts to confuse it?

? How does the model adapt to changes over time?

Similar Terms brittleness, adversarial resistance, overfitting, transfer learning

Measuring Resilience

Resilience describes how well a model recovers to desired performance from a performance-degrading event such as a failure. Resilience is reactive, while robustness is proactive.

How do you measure it?

Testing for lingering effects of stress and perturbations may give insights into model resilience

Testers should consider the following:

- The frequency and severity of disruptions that may be encountered
- The time and resources required for the system to recover and resume normal operation.
- The impact of disruptions on system performance such as accuracy, reliability, safety, or user satisfaction.
- The mechanisms employed to cope with disruptions such as error detection, compensation, or learning.

A resilience curve will plot the system performance over time under different scenarios of disruptions or failures.

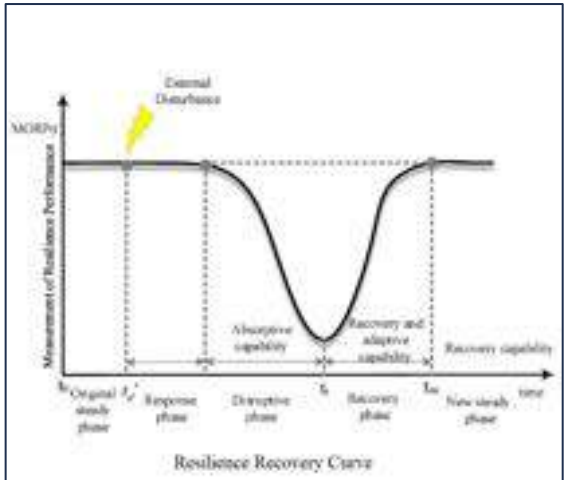
Why should you measure it?

Resilience is a crucial aspect of the system's reliability, usability, and trustworthiness.

Measuring resilience can help identify and improve the system's weaknesses.

The distinction between robustness and resilience is clearer when dealing with dynamic models, such as autoregressive models. With a dynamic model, future output depends on the history of inputs and outputs, so errors propagate forward in time.

A contemporaneous perturbation may result in poor performance right now (robustness issue) but also continued poor performance after the perturbation has passed (resilience issue).



What should you ask?

? Is the model dynamic?

? How will testing measure lagged and lingering effects?

? Can the model identify mistakes and self-correct?

? How does the model handle errors or failures and recover from them?

Similar Terms autoregressive, recovery time

Measuring Uncertainty

Uncertainty refers to the level of confidence users should have in the outputs of a model. There are many sources of uncertainty, including natural random variation, a lack of training data, and others.

How do you measure it?

There are many types and sources of uncertainty, including aleatoric (inherent randomness), epistemic (ignorance regarding what we know), and approximation (how well the chosen model approximates reality).

Many uncertainty quantification methods exist, and the best choice will depend on the task, modeling framework, and your assumptions about the use case. Common methods include subsampling, analytic approaches, Monte Carlo simulation, and ensembling.

One should ensure the validity of the chosen uncertainty quantification method by assessing agreement with the true uncertainty.

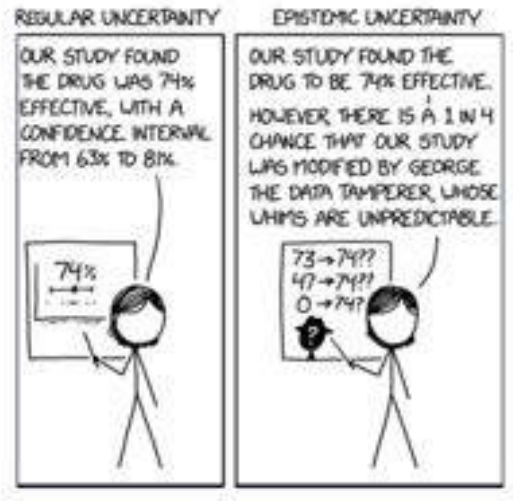
Why should you measure it?

To improve the performance, reliability, and interpretability of the model by providing a measure of confidence or error for predictions. This helps users or stakeholders to trust and use the model predictions more appropriately.

To identify sources and types of uncertainty affecting the model, such as natural variation, lack of training data, model complexity, or adversarial attacks.

To reveal the regions of high or low uncertainty in the input or output space. This helps testers or developers evaluate the model's generalization ability and reliability and prioritize the areas needing more attention or data.

xkcd.com



What should you ask?

- ? How certain do we need to be of a model's predictions?
- ? Will the model be used in a context that was rare in the training data?
- ? How does uncertainty vary across subpopulations?
- ? Is the estimated uncertainty properly calibrated?

Similar Terms Epistemic uncertainty, aleatoric uncertainty, confidence intervals, Bayesian inference, stochastic processes, Monte Carlo methods, outlier detection

Measuring Drift

Drift is the change of model performance over time due to changes in its environment. Drift can apply to the data, the model, or the context/concept underlying the model use case.

How do you measure it?

Model performance should be tracked with updated data, and changes in performance should be investigated.

Assessing data distribution changes over time can reveal sources of drift.

To measure data drift, use feature-based methods to compare the statistical properties of the features in the training and production data.

To measure model drift, one can use output-based methods to compare the predictions of the model on the training and production data.

To measure concept drift, compare the performance of the model on the training and production data. Such comparisons can capture changes in the predictive power, sensitivity, and more.

Why should you measure it?

There are various types of drift that affect AI models: drift may occur when the distribution of inputs or outputs change over time or when the relationship between inputs and outputs changes. Careful modeling can account for some kinds of data drift during development, but the source of drift is often not known or observed. All drift symptomatically manifests as change in performance (typically degradation) over time.

Monitoring for changes in performance and changes in the data can prompt the need to retrain the model, collect more data, update procedures associated with data collection or use of the model, or investigate other sources of degradations more closely.

What should you ask?

? How can the model monitor or report data, concept, and model drift?

? How will updates correct for the effects of drift?

? How can the model be updated, retrained, or revalidated for drift?

? What is the desired data distribution under expected conditions?

Similar Terms Concept drift, context drift, data drift, model drift, model decay, online learning, streaming data, non-stationarity, model monitoring

Measuring Representativeness

Data used for model training must be sufficiently representative of the real operational environment in which the model will be deployed. Model performance depends on the training data.

How do you measure it?

Training data, testing data, & validation data should all be representative of the operational environment, and the model architecture should be sufficient to capture salient operational features.

Test data & validation data should be different from training data.

Typical measures will show the degree of similarity between the planned dataset and the salient features of the operational context. Example measures may include: data similarity, data diversity, data coverage, data currency, and data volume.

Similarity and sufficiency must normally be defined by subject matter experts in the operational context.

Why should you measure it?

To reduce the risks of overfitting, underfitting, or bias in the model.

To evaluate the model performance and characterize how it will generalize.

To identify and improve data quality and quantity issues and the architecture of the model.

To help ensure that the model meets user expectations, needs, and preferences in the real operational environment.

Representative training data that aligns with “real” operational data in all significant aspects is vital to achieving desired model behavior.

Worthwhile testing critically depends on the test data being representative of real operational data.

What should you ask?

? What variables define representativeness for the operational context?

? Are those variables captured in an operationally relevant way?

? What is the fidelity of the data with respect to actual operations?

? Does the operational context and associated data change over time?

Similar Terms Class imbalance, sampling bias, dataset diversity

Measuring Latency

Production AI models must operate at a speed sufficient for their users. Latency can depend on model complexity, input data size, hardware performance, and other factors.

How do you measure it?

Latency is measured by timing responses of the model.

Latency must be measured in an way representative of the operational environment to be meaningful.

Profiling tools can measure the time taken by model components to help identify the bottlenecks.

Benchmarking tools measure the latency of the model on different hardware platforms, such as CPUs, GPUs, TPUs, etc. This can help compare the trade-offs between speed and accuracy, and choose the best platform for deployment.

Load testing tools measure latency under different levels of demand or request frequency or size.

Why should you measure it?

In many applications like real-time analytics or autonomous navigation, latency constrains usability.

Latency also affects the smoothness of the user experience even in less time sensitive applications.

Running a machine learning model in production may incur high costs due to the use of cloud computing, storage, or bandwidth. Measuring latency can help select the most efficient and cost-effective platform and configuration for deployment.

To troubleshoot the model's behavior in real-time. Measuring latency can help detect and diagnose problems due to drift, system failures, or performance degradation and take appropriate corrective actions.

What should you ask?

? Does the model operate at a speed sufficient for its intended operation?

? Is there significant worst-case variance in the latency?

? Are there bottlenecks that could be addressed to boost inference speed?

? How will latency be measured in production?

Similar Terms inference speed, latency, network latency, throughput, edge computing, response time

Thinking about Testing Methods

This Section:

+ Outlines types of tests unique to AI
+ Presents ideas on test design for AI models
+ Discusses balancing test implementations

02

Test Types, Designs & Methods

Test design & test type selection will be informed by both the data and the model.

Section 1, Thinking about Performance, introduced some considerations about the performance of an AI model for DoD applications and ways to measure its performance. With those considerations in hand, this section will introduce considerations for fleshing out the test strategy into an implementable plan.

Test implementations must be balanced between the strengths and limitations of test methods.

Testers must account for priorities and available resources when selecting from the appropriate test types and test design approaches. Though some methods used in test planning require adaptation or invention, many of the techniques in testers' toolboxes will continue to be relevant for AI models and systems.

Test Types for Comparisons

There are various test types for testing model performance. The types shown here differ in how the model is compared and how data are used.

Pairwise Testing

Pairwise testing is a technique where test cases are generated such that all possible combinations of any two variables (or parameters) are covered at least once. The goal is to significantly reduce the number of test cases while still capturing most of the defects that would have been found using exhaustive testing. In machine learning, examples include varying pairs of hyperparameters* to avoid exhaustive grid search or testing the interaction between pairs of features. Pairwise testing does have limits.

In more complex models, analyzing every pair of parameters can be computationally costly, requiring either statistical sampling or testing pairs at a higher level of abstraction that compromises on rigorous definition.

A/B Testing

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

A/B testing is common when comparing a new variant of a model in production. As an example, 50% of a website's users might see content recommended by the current production model, while the other half see recommendations from a new variant. If tests of the new model show improvement, the model is swapped in, otherwise the system keeps its current state.

Back-to-back Testing

Back-to-back testing, also known as differential testing, is related to A/B testing, but rather than comparing two variants of the same model, the model under test is compared with a different model.

The goal of back-to-back testing is usually to identify defects by contrasting performance of the two models. The second model could be in an existing system that is being considered for replacement or it could be a model that is used solely for testing because it lacks properties necessary for production use, e.g. it runs too slowly.

Hyperparameters

Top-level parameters whose values control the learning process. E.g., train-test split, learning rate, activation function, or hidden layers.

Test Types for Adversarial Threats

AI models may or may not be secure against malicious attacks that aim to manipulate their inputs, outputs, or behaviors.

Adversarial Testing

Adversarial testing is a technique that uses adversarial methods to identify and address the vulnerabilities of a model, making it more resilient and trustworthy.

One type of adversarial method is adversarial attack, where an attacker subtly perturbs valid inputs that are passed to the trained model to cause it to provide incorrect predictions. For example, to defeat spam filtering a red team might slightly change an email's wording to avoid classification as spam while remaining readable to the intended audience. Another type of adversarial method is data poisoning, where an attacker manipulates the training data to cause the model to behave incorrectly.

Red Teaming

A red team is a group of testers that uses adversarial methods to probe a model and explore its attack surface, which can be large and unwieldy for some models. Red teaming can reveal unexpected model behavior, coverage gaps in the data, gaps in procedure, security vulnerabilities, other points of stress in the model, and evaluate the effectiveness of current defenses. It is appropriate during all stages of the lifecycle, including deployment.

Red teaming is most often used to emulate potential adversarial attacks and is associated with cyber testing, testing the code, and testing the statistical soundness of the model, but other aspects of the model can be red teamed as well. For example, data curation policies, monitoring and feedback loops, auditing mechanisms, and guardrails for proper use can all be red teamed.

Manning a Red Team

A red team can be internal or external professionals, or crowd-sourced testers, depending on availability, expertise, and budget. Crowd-sourcing can provide a larger and more diverse pool of testers, but it may also pose challenges in quality, security, and ethics.

Red teaming can be costly for complex and large models, but some ways to reduce cost and increase efficiency are red teaming events, such as DEF CON, where experts can congregate and contribute, network, learn, and give feedback.

Policies and procedures are needed to ensure safe and effective red teaming, grade efforts and results, and capture feedback.

Test Types

In cases where the expected outcomes of a model are less well understood, these test types may contribute to the overall characterization of the model.

Experience-Based Testing

Experience-based testing refers to the practice of leveraging the skills, knowledge, intuition, and expertise of testers to identify potential issues in a software system. Error guessing is typically based on testers subject matter knowledge, typical developer errors, and failures in similar systems. An example could be the use of knowledge about how ML systems have failed in the past from a database like the AI Incident Database. Experience-based testing is limited for scalability and is inherently subjective. This method might miss issues that could be captured through more systematic testing approaches.

Metamorphic Testing

Metamorphic testing is a software testing technique that's especially useful when you have a system where it's hard to know the "correct" output for a given input, which is often the case with machine learning models. Unlike traditional testing methods, which compare the output to a known "correct" answer, metamorphic testing focuses on the relationships between inputs and outputs.

The first step is to identify properties or rules that the output should satisfy when the input is changed in specific ways. These are called metamorphic relations (MR). Next, for a given input the model's output is observed. Then, the input is transformed according to the established MRs, and the model is re-run. The outputs are then checked to see if they maintain the metamorphic relations. For example:

- For speech-to-text, slight changes in playback speed should not change the output.
- For clustering, adding a point at the centroid of each identified cluster should not change the number of clusters or any existing point's assignment, and should assign each new point to the cluster that generated it.

Test Design

Test design focuses on selecting specific data points for analysis, enabling you to draw conclusions or make inferences about a larger population or phenomena.

Traditional test design allocates design points in a way that captures maximum variation in the data for testing. This often assumes a smooth response surface, not present in many AI systems.

AI systems may exhibit non-smooth responses, especially near edge cases and low density areas of the training space. Testers should ensure that those areas are not omitted from testing and may wish to prioritize those areas for testing in some situations. Model testing should emphasize regions of low density coverage in the training data.

Factor Selection

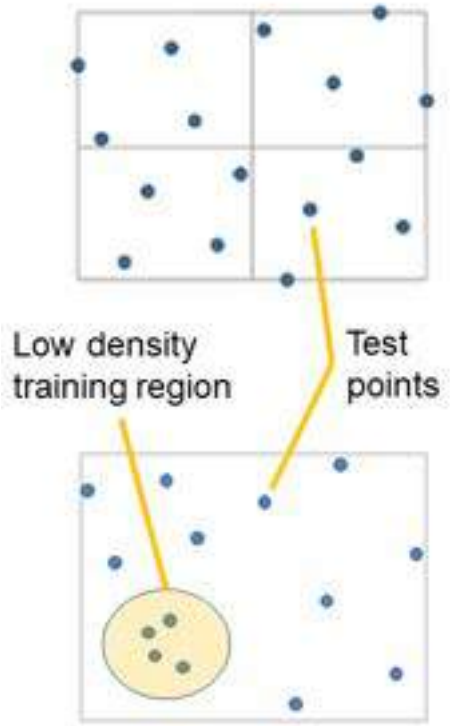
The process of identifying the factors that are most important to the model's performance and ensuring they are included in the test design and test data

Common factor selection methods

- SMEs provide valuable insights from a domain-specific perspective.
- Automated screening applies automated tools to identify potential issues with the model such as overfitting, underfitting, and data leakage and helps ensure comprehensiveness and rigor.

Common test design methods

- **Cross-validation**: evaluate performance across k smaller sets (or folds) of the data, then average the results
- **Holdout validation:** separate the data into training, test, and validation sets
- **Bootstrapping:** generate multiple training datasets by resampling the data with replacement. Train on the multiple training datasets, then evaluate on the original dataset



Testing paradigms: beyond either/or

Synthesize across testing paradigms to strike the right balance.

Many testing paradigms are often framed as extremes or an either/or – often pitting automated testing solutions against processes with ongoing human input.

Reality is typically far more gray, and well balanced solutions will likely leverage a combination of test implementations.

Testing professionals should educate themselves on the strengths and limitations of different testing paradigms and choose the best fit for their context and goals.

Balancing Test Implementations

Continuous Testing	Discrete Test Events
Tests occur at regular intervals or at model updates or other triggering events	Tests occur at specified points in time
Follows pace of development, changes in model over time are observed	Often easier to schedule and resource, can be larger in scope because computational resources are prioritized for testing
Can add complexity to scheduling and resourcing, often less extensive due to computational constraints	Can be out of sync with developer pace

Balancing Test Implementations

Automatic Testing		Manual Testing
Tests specified in code and run at set intervals or every model update		Tests conducted by an expert
Cheap to run once tests are developed, fast, good for catching regressions, thorough, and good for testing predefined logic cases		Handles complex tests difficult to specify in code
Adds to code maintenance burden, difficult to automate complex evaluations		Expensive in long run, slow, can be less thorough

Balancing Test Implementations

Black Box Testing	White Box Testing
Tests that focus on feeding inputs to the model and evaluating its outputs	Tests that require knowledge of the internal structure of a model, such as its parameter weights and training data
Requires less privileged access, not dependent on model architecture	Can be more interpretable, some adversarial techniques require internal access
Can be computationally expensive, difficult to interpret results	Can be difficult to negotiate access to sensitive intellectual property, some techniques limited to particular model architectures

Balancing Test Implementations

Live Test Events		Model & Simulation
Tests occur in environment as close to operational one as possible		Tests occur *in silico* (i.e., experimentation performed by computer) using data from model or simulator.
More operationally realistic		Highly scalable, can be quickly modified
Expensive, infrequent, modification harder		Might have significant differences from deployed environment

Thinking about Data

This Section:

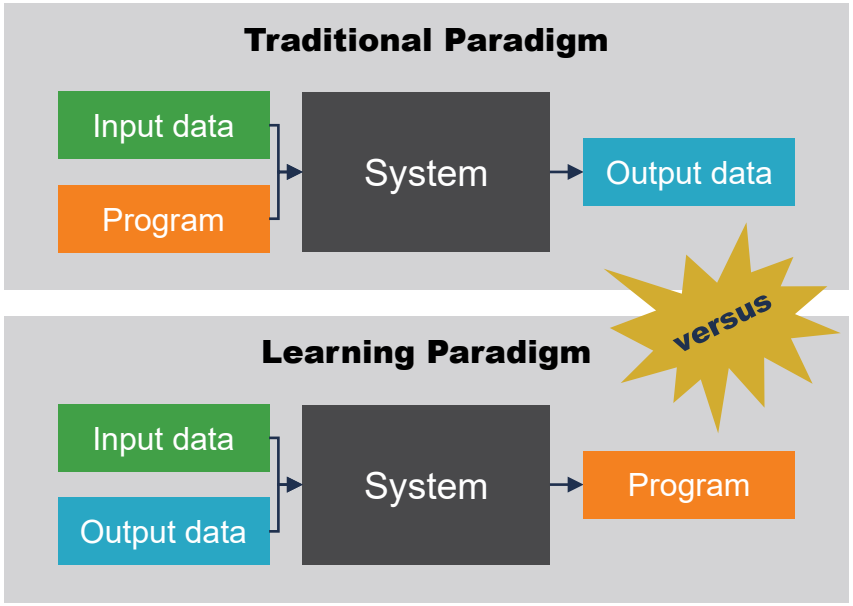
+ Describes data as the foundation of AI models
+ Explains how data affects model performance and testing
+ Provides key considerations tied to the data lifecycle
+ Provides details on common data sources and data types

03

Thinking about Data

Data influences model performance and can introduce testing challenges

Machine learning turns the programming paradigm on its head. Traditional programming specifies some desired process a system should follow or an output it should produce. With AI models, however, the system learns from examples rather than explicit instructions.



Data is the foundation upon which AI models are built, and any shortcomings can result in poor model performance. This paradigm shift introduces new and increased risks compared to traditional software that should be captured via testing.

The rest of the 'Thinking about Data' section covers:

The Data Lifecycle

Select, clean, engineer, split, curate, and verify & validate

Common Data Sources

Commercial, laboratory, manually collected, model output, open source, synthetic

Common Data Types

Audio, image, text, video, tabular

Learn More

NIST's AI Risk Management Framework and CDAO's National AI Infrastructure T&E Capability (NAITIC) Gap study discuss additional differences between software and AI models.

Assuring data quality across its lifecycle

Testers must ensure that data are complete, diverse, and realistic, and that the data lifecycle steps are documented, validated, and reproducible.

1 | Select

Identify, locate, and acquire needed sources of data

2 | Clean

Identify and mitigate errors, missingness, and poor quality

3 | Engineer

Structure data, engineer features, and label as appropriate

4 | Split

Split the data appropriately into subsamples for training, testing, and validation

5 | Curate

Version and store data, code, and other information

Verify & Validate

Ensure the integrity, accuracy, and quality of the data

Important!

This process is far from linear! It's inherently iterative!

1 | Select & acquire your data

Data must be related to the use case, be operationally realistic, and have variation.

How is it relevant to testing?

Testing real world performance and reliability of the model critically depends on the operational realism of testing data.

Bringing operationally realistic data forward in the development and testing process can be costly, but failing testing and retraining can be costly, as well. Synthetic or simulated data can help manage this trade-off, but one must carefully consider the fidelity with which the data is generated.

What are best practices?

You need a lot of data – Data are often noisy. Testing requires sufficient coverage of the operational space; identification of edge cases and exploration of the operational space for unexpected behavior may also require large amounts of data. Response surfaces of AI models may not be smooth, so traditional experimental designs, including DOE, may provide insufficient coverage of all possible values the data could take to capture points of failure.

What should testers ask?

? What rights does the government have with respect to the data?

? What are the data sources? Is sourcing legal and ethical?

? How can you ensure that the data are complete and operationally realistic?

? Think about how the data are generated. Is there relevant information that is not observable? Does the data contain known bias?

2 | Clean your data

Identify and correct or mitigate entry errors; incorrect or corrupted information; incorrectly formatted entries; duplicate, incomplete or missing data; and other quality issues.

How is it relevant to testing?

Data are rarely ready to be used out of the box. Training data must be cleaned, and the cleaning steps must be validated. Test data should be appropriately cleaned and validated as well. These are actions that testers should confirm.

In response to model or data errors discovered in testing or V&V, additional iterations in data cleaning may be required.

Subsequent needs for new data will also drive additional data cleaning.

What are best practices?

Iterate V&V with data cleaning.

Normalize data tables as needed for curation.

Document actions. Decisions made when cleaning data often have advantages and drawbacks. Documentation promotes transparency and facilitates auditing.

Do By Code! Use code to clean the data and maintain a repository of code that reproduces all cleaning steps to aid transparency and reproducibility. Avoid manually cleaning data in spreadsheets!

What should testers ask?

? What does missingness mean to the applicability of the data?

? What are the valid values, have definitions or ontologies changed over time?

? Where is the data incomplete?

? What might affect the operational realism of the data?

? Is there evidence of data poisoning?

? Are the cleaning processes sufficient?

3 | Engineer your data features

Feature Engineering transforms raw or cleaned data elements into a format that is useable by the algorithm.

How is it relevant to testing?

Raw data sets need to be combined into a single data set and transformed into a format that makes sense for use in machine learning. The transformation steps and resulting data should be V&V'd.

Inputs and targets (outputs or reward signals), including labeling, must be specified. Labeling is often cited as being a critical, but costly and labor-intensive part of this process.

What are best practices?

Do by code! Using code to engineer your data and maintaining a repository of code that reproduces all decisions aides in review, auditing, and reproducibility.

Decisions made when engineering features often have advantages and drawbacks. Documenting your decisions promotes transparency and facilitates auditing.

What should testers ask?

? Are the engineering steps appropriate?

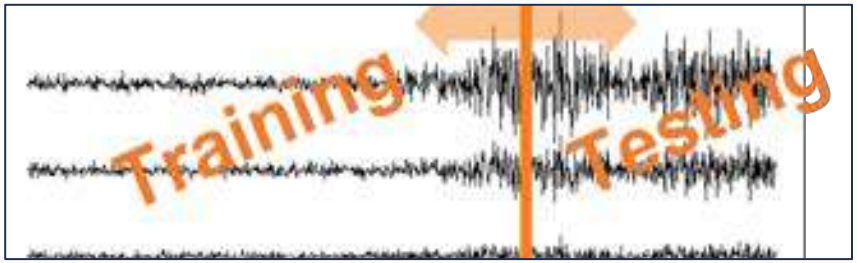
? What are the relevant features for the problem at hand?

? Is the engineered data appropriately operationally representative?

? Were the data feature engineering steps verified and validated? And documented?

4 | Split your data

Splitting data refers to partitioning a data set into various subsamples. Recall that AI learns from the data in its training sample. To properly evaluate an AI model, it must be tested on data that it has not learned from.



How is it relevant to testing?

Similar to how a teacher evaluates student learning, providing the answers to a test before taking it does not produce a trustworthy evaluation. Models should be tested against operationally representative data that they did not see during training.

Ensuring reproducibility: By splitting the data into training, validation, and testing sets, we can ensure that our results are reproducible. Note: terminology differs across fields, a validation split is used for hyperparameter optimization – not validation of the data!

What are best practices?

Do By Code! Maintaining a repository of code that reproduces all preparation decisions aides in review, auditing, and reproducing results.

Re-Validate! Generally, all sample splits should be representative. Re-validate your subsamples for operational realism and representativeness after splitting.

What should testers ask?

- ? Are the data cross-sectional, time-series, or panel?
- ? What variables define representativeness?
- ? How to choose the temporal split points? Does the distribution change over time?
- ? How to choose subsample sizes and the number of subsamples? (Should test set be 10% or 30%, single train/test split, k-fold cross validation, etc.)
- ? Could testing data have leaked into the training set?

5 | Curate

Data Curation refers to deliberate, active management of the data over time alongside metadata, documentation, cleaning and engineering code, and V&V and auditing results.

How is it relevant to testing?

Curation of the data should be adequate to facilitate testing, secure enough to guard against adversarial attack and spills and corruption, and robust enough to facilitate versioning and rollback.

Proper curation facilitates transparency, auditing, and testing.

What are best practices?

Proper version control is not just for code and models! It can be used for data and meta data, too.

Take care in managing data fidelity; balance the cost of curating operationally realistic data with that of synthetic data and ensure model performance remains satisfactory.

Evaluate, assess, and audit all curated information for completeness, correctness, transparency and usefulness on a deliberate schedule.

What should testers ask?

? Who are the stakeholders and collaborators across the dataset's lifecycle

? Are the data and V&V and other auditing results adequately documented in a Data Card?

? Where and how are the data stored and accessed? How is cleaning and engineering code stored?

* | Verify & validate your data

Verification and Validation ensures the integrity, accuracy, and quality of the data. The data should be secure, operationally representative, and of sufficient quality and quantity to enable machine learning.

How is it relevant to testing?

The data should be sufficiently representative of the real operation for which the model will be deployed, and the processes supporting and defining the data lifecycle should be sufficient to support the use case and capture the salient information regarding operational outcomes.

Each of the previous slides in this section implicitly describes specific V&V activities relevant to that part of the lifecycle.

What are best practices?

Validation of the data should be on going rather than just at some fixed point in the lifecycle.

Data properties, contents, and ontologies should be reviewed in light of changing operational environments and user feedback.

Engage and iterate with data providers, subject matter experts, domain experts, developers, and end users.

What should testers ask?

? How are low density regions in the data treated (e.g. edge cases)?

? How will the data will be stored, accessed, and used?

? How are the data generated - is there any nuance that might affect its use or impede its ability to sufficiently capture salient information?

? What might be missing, and what assumptions were made in cleaning and engineering?

Important!

A “validation data set” is not the same as the validation of the data (see p 35.)

Common Data Types

Common data types include audio, image, tabular, text, and video. Data types more unique to DoD uses include LIDAR, RADAR, SONAR, electromagnetic spectrum data.

How is it relevant to testing?

The range of data modalities can shape testing priorities. Some types that are good use cases for AI applications in DoD are uncommon in industry.

The novelty of some DoD data types may drive the need for new evaluation methods, to assure data quality and representativeness.

This diversity of DoD environments, data modalities, and mission purposes require a nuanced understanding of the mission objectives and challenges to inform the handling of datasets.

What are best practices?

Recent progress and innovations in AI model applications have been borne from advances in managing datasets. For example, normalization and regularization have accelerated progress in generative transformers for images and video.

Normalization scales the input features to a similar range, which helps the model converge faster.

Regularization adds a penalty term to the loss function to prevent overfitting.

What should testers ask?

- What preprocessing techniques have been conducted on the varying data types?

Common Data Sources

This includes commercially curated datasets, open source data, lab environment data, manually collected data, real environment data, synthetic data, and other types of data.

How is it relevant to testing?

The nuances of data evaluation and its required rigor will depend on a variety of factors. The origin of the data can influence testing priorities.

Many data sources can introduce opacity regarding data chain of custody and curation processes, posing a potential security risk. These risks give rise to additional testing requirements.

A lack of historical knowledge about the data's provenance could introduce uncertainty about the amount of testing required.

What are best practices?

- Create and maintain an inventory of data sources and documentation.
- Ensure that data-related legal agreements and informed consent procedures document data access and re-use rights.

What should testers ask?

- How is representativeness validated for data sources?
- Has the data from a source undergone rigorous quality control and is it reliable?
- What is the chain of custody from a data source and how it is verified?
- Does the test team have adequate historical knowledge of the data's provenance and processing?

Thinking about AI Models

This Section:

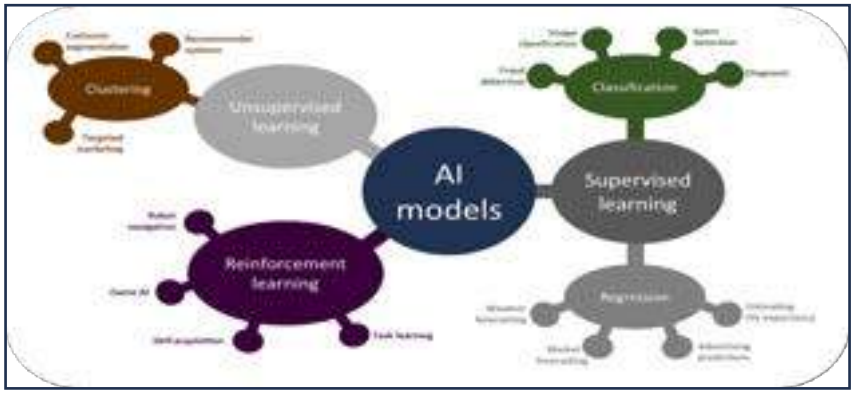
+ Describes the AI model lifecycle and its steps
+ Describes common machine learning paradigms
+ Describes common machine learning algorithms

04

Thinking about AI Models

Testers must ensure that the model is appropriate, accurate, reliable, secure, and compliant with the problem domain and the solution objectives.

Testers should consider the problem type and complexity, the solution objectives and constraints, and the trade-offs and challenges of machine learning when planning a model evaluation. Different problems require different metrics and methods to measure the model performance and accuracy. The evaluation should align with the solution goals and requirements and account for the solution constraints and limitations. The evaluation should also consider the potential issues and difficulties of machine learning, such as bias, overfitting, or interpretability. The most important aspect depends on the problem and context, but the evaluation should be valid, reliable, and meaningful.



The rest of the 'Thinking about AI Models' section covers:

The AI Model Lifecycle

Model type selection, feature selection, train & test, evaluate, deploy, monitor & maintenance

Common Learning Paradigms

Supervised, unsupervised, reinforcement

Common Algorithm Types

Neural nets, linear regression, decision tree, support vector machine, k-nearest neighbor

AI model performance lifecycle

Testers must ensure that the model is appropriate for the given problem and well curated and documented across its lifecycle. The lifecycle is outlined below.

1 | Select

Pick an architecture that aligns to the problem

2 | Define

Specify features, metrics, and interpretation methods

3 | Train & Test

Assign parameters & iteratively train & test

4 | Evaluate

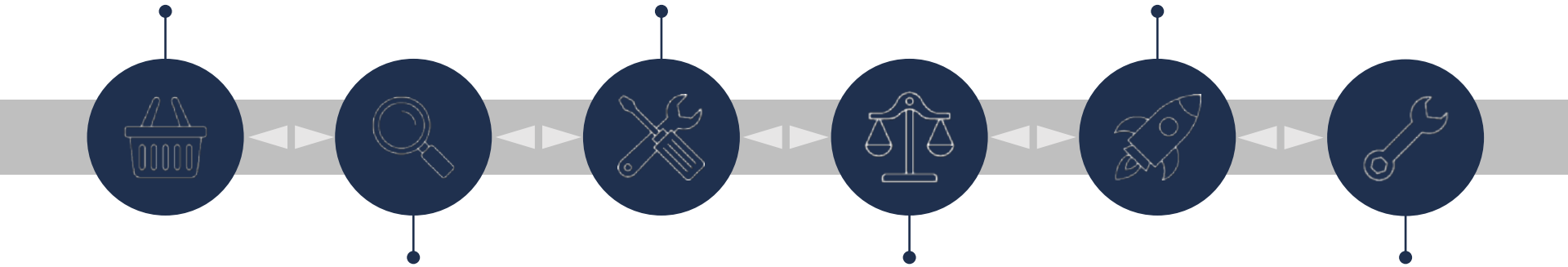
Ensure the performance of the model against solution objectives

5 | Deploy

Deploy model to production with suitable methods and tools.

6 | Maintain

Monitor and maintain model performance



Important!

This process is far from linear! It's inherently iterative!

1 | Select your AI Model Architecture

A model architecture will set limits on model performance and complexity. The optimal model architecture will depend on the task and resource requirements.

How is it relevant to testing?

The choice of model architecture will set constraints on the performance criteria mentioned before. This is typically via architecture genre and complexity.

Different architectures or algorithms have different assumptions, limitations, and performance criteria that need to be tested and evaluated.

Testers should understand how interpretability changes as complexity increases. The right balance between complexity and interpretability hinges on traits of the specific use case.

What are best practices?

The most complicated architecture is not always necessary. Begin iteratively with something basic and then add complexity if accuracy targets are not met.

Simpler architectures may offer higher explainability but may have less predictive capability.

In some cases, AI may not be necessary at all. Try to identify if improved data collection or a deterministic algorithm would be sufficient.

What should testers ask?

? Is the complexity of the model architecture appropriate for the accuracy, interpretability, and latency requirements?

? Does the model architecture follow from successful employment in similar purposes and/or use cases? If not, why wasn't another architecture selected?

? Does the model architecture fit within the operational constraints of the hardware that the model will be run on once deployed?

2 | Define your Model Features

Features are transformed raw data used by the model in training. Sometimes features are engineered, and in other cases they are learned by the model itself. Relevant features must be selected either prior to or during model training.

How is it relevant to testing?

This stage defines the data features that will be used by the model, the performance metrics that will measure the model accuracy and reliability, and the interpretation methods that will provide insight into the model logic and decisions. The data features, performance metrics, and interpretation methods should be suitable for the model type and the problem domain, and they should be tested and validated for their quality and suitability.

What are best practices?

Define features in code! Using code to engineer your features and maintaining a repository of code that reproduces all decisions aides in review, auditing, and reproducibility.

Don't assume features stay relevant or useless forever. Reevaluate their utility periodically.

What should testers ask?

? Are the selected features relevant to predicting the model's output?

? Have the developers checked that features used do not serve as proxies for protected classes?

? Are the features observable in available datasets?

? Are the features meaningful to humans?

? Are the features static or could they be changed by an adversary?

Example

If area and price are in the input dataset, one could create the feature price per square foot which may result in a better performing model

3 | Train and Test your Model

Training involves iteratively updating a model's parameters by minimizing a loss function with training data. Once a model is trained, its performance is estimated by testing on data split out for that purpose.

How is it relevant to testing?

During training, the model's parameters are iteratively updated by minimizing a loss function with training data.

Testing is measuring how well the model performs on data that it has not seen before. The model training and testing processes should include checks for model correctness and latency.

What are best practices?

Document hyperparameters in a manner sufficient to reproduce the training run.

Utilize experiment tracking tools to reduce manual documentation overhead.

What should testers ask?

? What are meaningful metrics of performance?

? Does testing align with the operational space

? How much testing is adequate?

? Are training hyperparameters such as random seeds and learning rates documented?

? How does performance differ among subgroups in the test set?

? How should test results be monitored for drift?

4 | Evaluate your Model

Check and confirm model's alignment with solution objectives and requirements. Iteratively evaluate model on representative data types. Monitor and address model trade-offs, challenges, security, and compliance.

How is it relevant to testing?

This stage involves checking and confirming the model's alignment with the solution objectives and requirements. The model is evaluated on different types of data that was held back from training data—the validation dataset. The model trade-offs and challenges, such as overfitting, underfitting, or interpretability is also assessed and addressed in this stage. The model's security and compliance issues are also ensured in this stage.

What are best practices?

Use cross-validation techniques to estimate how well the model will generalize to new data.

Use appropriate performance metrics that are meaningful in the use case.

Check for bias and ensure that the model is fair and unbiased.

Ensure that the model is interpretable and can be explained to stakeholders.

What should testers ask?

? Is the model architecture and algorithm appropriate for the problem domain?

? Were the hyperparameters optimized for performance?

? Was model trained on a separate subset of the data and evaluated on a different subset?

? How well will the model generalize to new data when cross-validation techniques are used?

5 | Deploy your Model

Once the model is ready, it is exposed to users in production. However, development and testing of a model is never finished. Updates and testing continue for the life of the deployment.

How is it relevant to testing?

In this stage, the model is deployed using appropriate methods and tools, such as cloud services or edge devices. The model performance and functionality are monitored and maintained throughout its life, so deployment should include setting up the methods and tools for monitoring performance.

The model deployment process should be tested and ensured for its scalability, adaptability, availability, and fault tolerance.

What are best practices?

Monitor model performance in the deployed environment. While, on going model testing is not necessarily standard DOD practice, performance and other metrics of a deployed model are likely to change over time.

Use gradually scaling rollouts to identify issues with a subset of users before risking widespread failure.

What should testers ask?

? Does the deployed model's performance match what was expected from earlier testing?

? How can graduated fielding and/or beta users be utilized to reduce risk and test burden?

6 | Maintain your Model

Monitor and update model in production. Retrain or modify model with new or updated data. Test model on updated data or environment. Repeat update cycle as needed.

How is it relevant to testing?

The model needs to be constantly updated and tested throughout the deployment period. Updating the model means retraining it with new or revised data, or changing its structure or parameters. Testing the model means measuring its performance on the new data or situation. The updating and testing process is iterative and never-ending, as the model has to adapt to the changing needs and feedback of the users. The updating and testing process should also be evaluated and enhanced for its speed, stability, and adaptability.

What are best practices?

Monitor model performance in the deployed environment.

Automate tests and monitoring as possible to reduce manual overhead.

Model and/or system instrumentation and data collection automation is vital for model evaluation and maintenance. Often one of the hardest things is recording the data in the field.

The T&E strategy should identify what instrumentation is needed, who provides it, and the organizational roles and responsibilities for verifying, validating, and accrediting it.

What should testers ask?

? How does the model fit into the broader curation approach for the project?

? Is the monitoring approach sufficient to detect drift before it causes mission failures?

? How much and what instrumentation is needed for the test strategy.

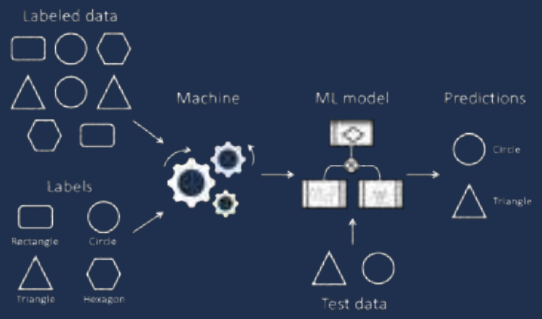
? Will the planned instrumentation yield the needed model metrics?

Common Learning Paradigms

The nuances of AI model evaluation will depend on a variety of factors, including the model's learning paradigm. Some systems use combinations of multiple paradigms.

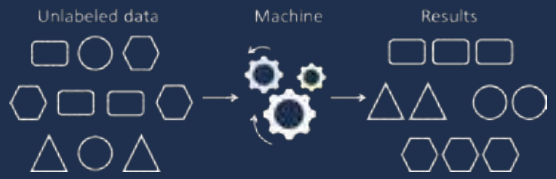
Supervised Learning

These models find a function that maps input data to output data (i.e., labels), by iteratively adjusting a set of parameters and calculating the difference between the model's answers and the correct answers. This is called the training process. Supervised models are commonly evaluated on their predictions using test data that was held back during training. The paradigm is used in Classification and regression.



Unsupervised Learning

These models identify underlying patterns or structures within the data without mapping to known ground truth. Evaluation of unsupervised learning models can be challenging because their assessment often relies on expert judgement of the modeled output. Unsupervised models are used to characterize the unconditional distribution of the data, often by grouping observations based on similarity as with clustering and association problems.



Reinforcement Learning

These models learn how to take courses of action in an environment from trial and error based on rewards for being correct and penalties for being incorrect. To test reinforcement learning models, it may be necessary to use a dynamic test environment that allows testers to assess how the model interacts with and adapts to its surroundings. The approach trains an algorithm through interactions with the environment and a sequence of rewards for optimal behavior.

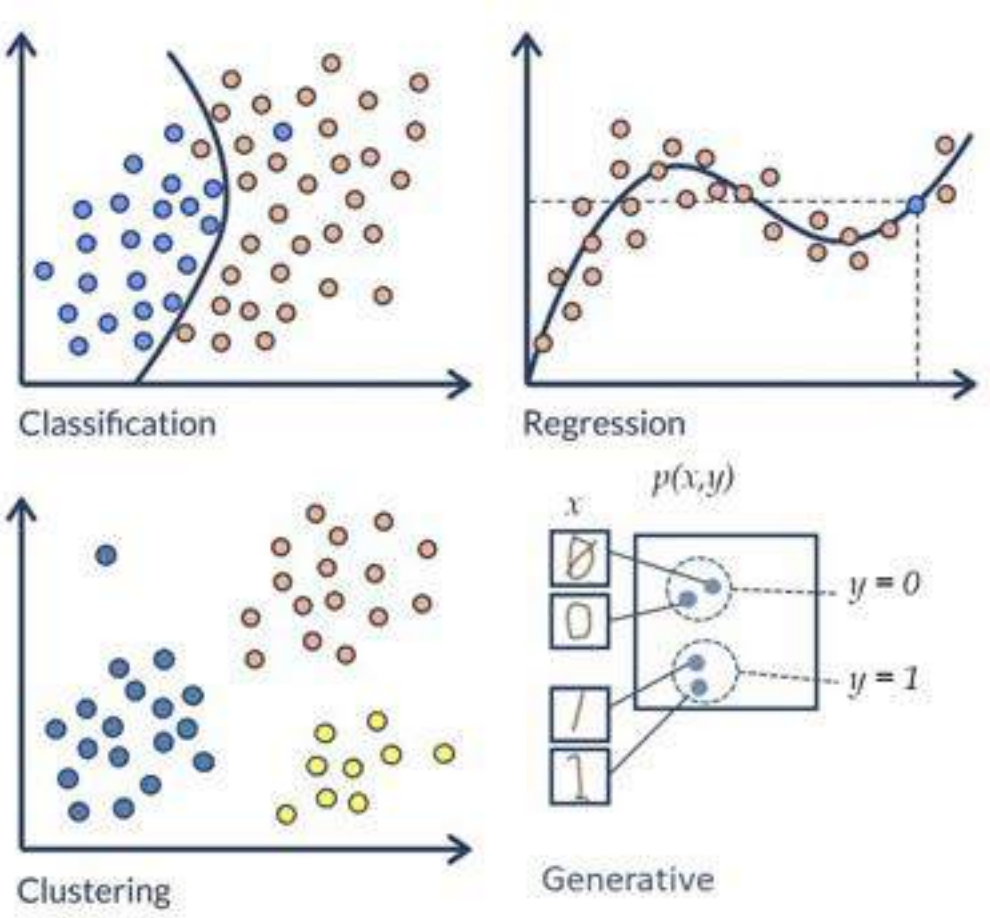
Common Algorithm Types

The next section introduces some of the common types of algorithms used in machine learning development.

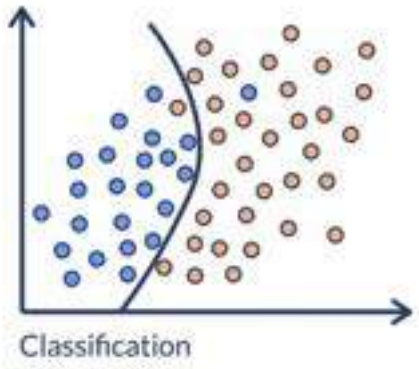
Many machine learning algorithms are useful for more than one type of analytical task. For instance, support vector machines are commonly used for either regression or classification tasks.

While there are many variations of each of these approaches (classification, regression, clustering, and generative), this section will outline a basic introduction to the concepts and provide some key consideration for testers.

Reflecting differing communities, some overlap exists in calling these algorithms versus analytical approaches or statistical methods.



Classification



Classification models are a type of supervised learning model used to predict the category or class of new, unseen data based on the model learning from a labeled dataset.

Examples in DoD systems

- Automatic target recognition systems such as DARPA's Target Recognition and Adaption in Contested Environments (TRACE)
- Predictive maintenance on helicopter engines in the 160th Special Operations Aviation Regiment (SOAR)

You should know

- Evaluation metrics include accuracy, precision, recall, F1 score, and more
- Hyperparameters are set before training begins and can have a significant impact on the performance of the model
- Overfitting is a common problem in classification models. Overfitting can be mitigated with techniques like regularization, early stopping, data augmentation, dropout, and ensemble methods

What should testers ask?

? What is the purpose of the model?

? How should the model be evaluated and what are the evaluation metrics?

? What are the characteristics of the training and test data in terms of its diversity, realism, and coherence.

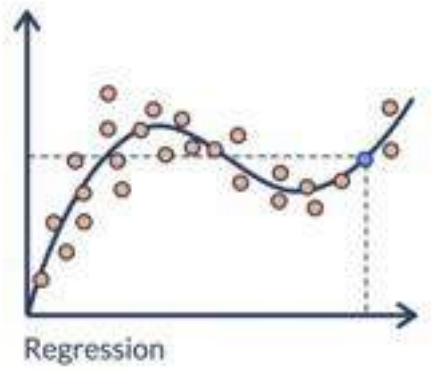
? Are the features of the test data well aligned to that of the training data? Are the training and test data representative of the expected context of the model?

? What are the limitations of the model?

? What is the expected behavior of the model in operating contexts outside of the design context?

Common Algorithms Logistic regression, neural networks, K-nearest neighbors, decision trees, support vector machines

Regression



Regression models involve input and output of continuous values and vary from simple linear regression to many more complex types.

Examples in DoD systems

Regression has been applied many times over many decades. A few examples are:

- Estimating years of service remaining for a service member
- Predicting cost of a project
- Estimating yield for explosives
- Predicting component time to failure

You should know

- Regression as an algorithm or a kind of analysis is ***not*** the same as "regression testing"
- Regression problems can be turned into classification problems by binning
- Normally, regression is used in applications in which a continuous value must be predicted from another continuous value or multiple values. An example could be predicting the numbers of hours before a component fails from the measured viscosity and temperature of the component's lubricant.

What should testers ask?

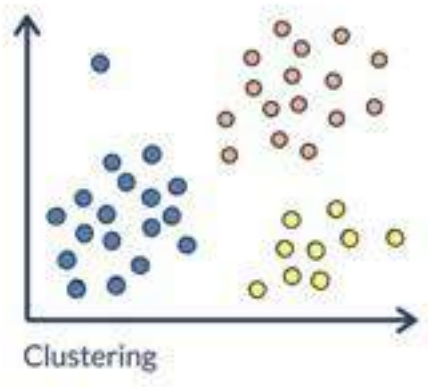
? What is the purpose of the model?

? Are both average and maximum errors acceptable?

? If the regression method assumes certain properties of the data or residuals, are those assumptions valid?

Common Algorithms Linear, multivariate, polynomial, LASSO, ridge, and Poisson regression

Clustering



Clustering models are a type of unsupervised learning model used to group similar points or "clusters" based on their features.

Examples in DoD systems

- Identify operational cells from location tracking data
- Categorizing types of users on a network based on activity logs
- Social network analysis to identify threats to current events

You should know

Clustering algorithm output can be highly influenced by initialization conditions and user-selected parameters.

The definition of similarity is typically developed by subject matter experts that are knowledgeable in the expected use case for the model.

Some evaluation metrics for clustering models include silhouette score, Davies-Bouldin index, Calinski-Harabasz index, homogeneity score, completeness score, and V-measure.

The best partition and optimal cluster number is typically determined in the context of the use case.

What should testers ask?

? What is the purpose of the model?

? Do clusters align with what ground truth is known?

? What is the purpose of the model?

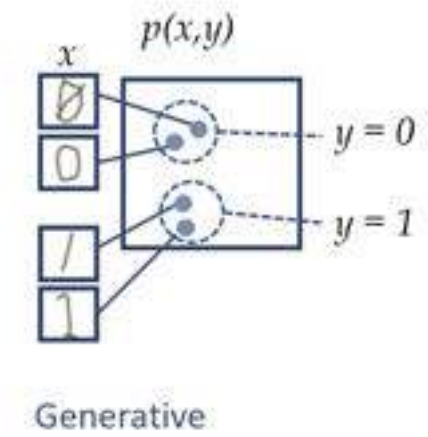
? How should the model be evaluated and what are the evaluation metrics?

? What are the characteristics of the data to be clustered and what are the defining characteristics of the desired clusters?

? What are the limitations of the model and how should it perform when operating on data that are approach the boundaries of the operational use case?

Common Algorithms K-means, hierarchical, DBSCAN, Mean-shift, spectral clustering

Generative



Generative models are a class of statistical models that aim to capture the underlying patterns or distributions of data in order to generate new, similar data.

Examples in DoD systems

- Task Force Lima has been formed to assess, synchronize, and employ generative AI across the DoD
- Large language models are being explored in many areas to better understand appropriate use cases
- Research at DARPA aims to extend generative capabilities using cross-modal (text, image, video) data

You should know

- Generative models are more multipurpose than other algorithm types
- There is often no straightforward ground truth with which to compare generations
- System performance is strongly affected by the interaction of user and system (e.g. prompts used)
- Generative models are typically more resource intensive to train and operate due to their scale
- This area is evolving on the scale of months. Expect rapid change.

What should testers ask?

? What is the purpose of the model?

? How will we develop appropriate evaluation metrics that are relevant to the intended use and ensure that they are applied consistently throughout the evaluation process?

? What is the quality of the generated data including aspects such as diversity, realism, and coherence?

? How do results from the model under test compare to other models that were developed for similar purposes?

? What is the impact of variations in data quality on model performance?

? Is the model generating content that is inappropriate or dangerous?

Common Algorithms Diffusion models, transformers, autoregressive decoders

Thinking about Context

This Section:

+ Discusses use cases of the model, including how it will be used and what problems it will solve.
+ Outlines considerations about the environment in which the model will operate, including data sources, data transport, computing resources, and security.

05

Use Case

The use case is a plain language description or story of how the model will be used and what problem it addresses. The use case provides key context for evaluating its effectiveness and how it is relevant to the problem at hand.

The basics

The use cases for the system should be defined in detail and be consistent with the Concept of Operations (CONOPS) and Concept of Employment (CONEMP).

Ideally, the designers and developers have created use cases. Testers should review use cases and contribute to their improvement if needed.

DoD systems are intended to enable users and units to accomplish their assigned missions.

What should you know?

Leverage the use cases to understand operations and capability supported and how the AI will fulfill operational needs.

The task outcomes require metrics that define success.

The metrics manifest from the interactions between users and units employing their systems and the environments in which they operate.

The AI systems and components roles in these tasks should be clear. The challenge for testers is to identify the problem and use case for the AI model and how the larger metrics of task success are connected to specific characteristics of the AI model performance.

What should you ask?

- Is the model a stand alone application, part of a larger software system, or a component in a physical (hardware) system?
- Does AI change the use case of the system it replaces?
- How does AI affect individual users' actions?
- How does AI affect units' actions (collective action)?

Similar Terms User story, end user, user-centered design

Environment

Consider the available computing resources, network constraints, and security environment that should be incorporated into the testing approach.

The basics

Testers need to consider how the characteristics of the environment drive test requirements and how those are connected with the model.

Key items of the environment are:

- Network connections
- Computing resources
- Security

What should you know?

Testing must be able to realistically portray the data sources the model requires in fielded operations.

Testing must account for the data transport infrastructure. Models operating on ships, aircraft, submarines, etc. will have varying throughput and stability and may not have connections to the internet.

Testing must match the fielded computing resources (memory, processor, and power).

Testing must account for the security constraints in fielded operations (level of classification, sensitive data, restricted access).

Classification can also place requirements on how data is handled and shared.

What should you ask?

? What changes in data sources over time should be expected?

? How will software updates impact data sources?

? How will data sources and transport vary by operating location?

? What data and network connectivity will exist in classified environments?

Similar Terms Containerization, development vs production, scalability, edge computing, TinyML, knowledge distillation, airgapping

Thinking about Documentation

This Section:

+ Discusses data cards and data characteristics
+ Discusses model cards and model characteristics
+ Outlines version control, automated documentation, test metrics, and context characteristics

06

Documentation

Comprehensive documentation, including the data sources, preprocessing, modeling techniques, evaluation metrics, is essential to inform tests of the fielded model.

Surround your documentation with best practices

Version Control

- All documentation should be versioned and curated alongside the model itself and the associated versioned data to facilitate version comparisons, auditing, transparency, and roll back decisions.

Automated documentation

- Where feasible, automation tools should be adopted to reduce the workload of creating and maintaining comprehensive documentation.

Evaluation Metrics and Results

- The documentation should detail the performance of the model in terms of standard metrics such as accuracy, precision, recall, F1 score, AUC-ROC, etc. Additionally, the documentation should describe the test set used for evaluation, its composition, and any biases it may contain.

Operational Characteristics

- The documentation should contain a section that identifies the intended use cases of the model, its limitations, and potential risks associated with its deployment. This could also cover any post-deployment considerations such as monitoring, retraining strategies, and contingency plans for unexpected model behavior.

Documentation helps ensure

The AI model is:

- compliant with relevant regulations and standards,
- comprehensively tested for all key use cases and fielded contexts,
- monitored appropriately for behavior drift and data drift, and
- transparent and understandable

Data Characteristics

Repeatability requires solid data documentation.

What should be documented?	Specifics to include
• Contains thorough documentation of the training and test data. This includes how the data was collected, its overall composition, any cleaning or preprocessing steps, and any data privacy considerations. Documenting potential biases in the training data is also important. • Identify stakeholders and collaborators across the dataset's lifecycle • Collect documentation and other information to populate a Data Card • Collect Data • Collect Code in a repository with documentation of code • Evaluate, assess, and audit the information for completeness, correctness, transparency and usefulness • Version control: All documentation should be versioned alongside the model itself and the associated versioned data to facilitate version comparisons, auditing, transparency, and roll back decisions. • Collaboratively review and update content and code	• The data, • Meta-data, • Data dictionaries, • Location of data and code, • Source information, • Points of contact, • A change log, • A chain of custody record, • Known limitations and caveats, • Code that reproduces cleaning, merging, engineering, preparation, etc. and • other relevant information

Data Card

The Data Cards Playbook, shown below, is licensed under a Creative Commons Attribution-Share Alike 4.0 International License

What should you know?

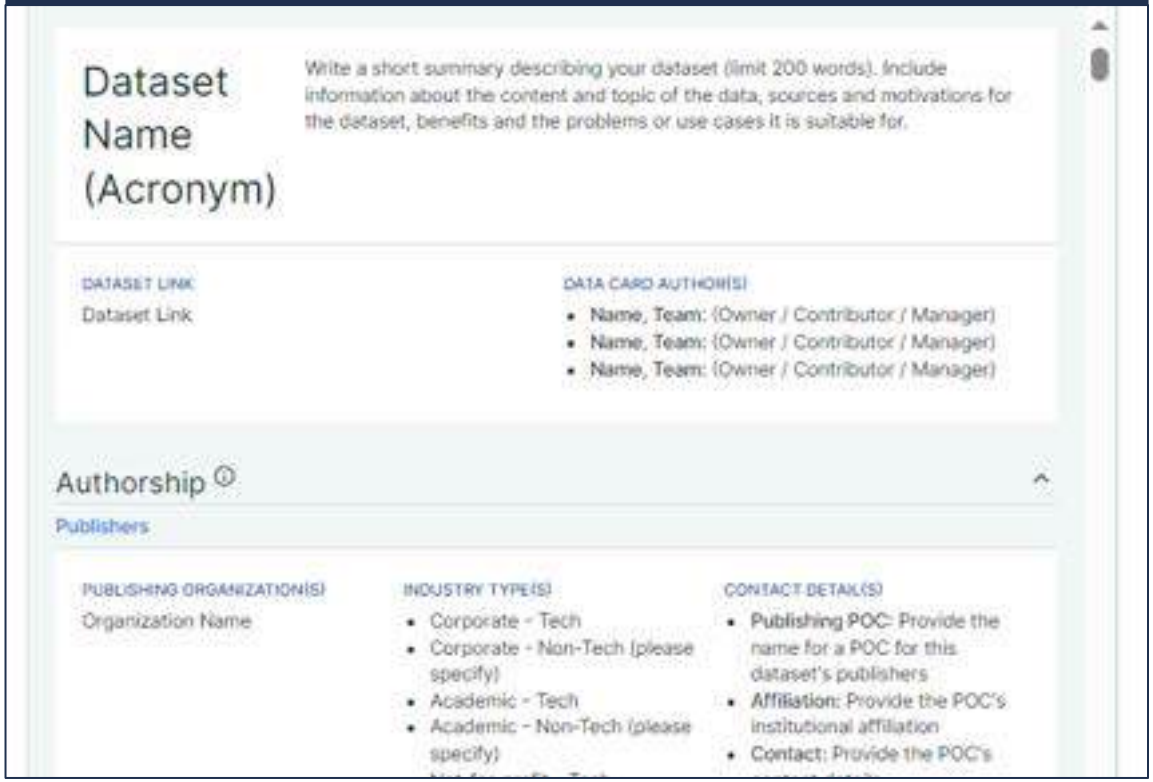
- The image at right shows the first 2 sections of an example of a data card template from https://sites.research.google/datacardsplaybook/.
- Best practices in documenting data is an active area of development and you should expect to see evolution there.
- More recommendations for the content and structure of a Data Card can be found at

  https://doi.org/10.1145/3531146.3533231

  https://arxiv.org/abs/2204.01075

  https://ai.googleblog.com/2022/11/the-data-cards-playbook-toolkit-for.html

Example Data Card



Model Characteristics

Repeatability requires solid model documentation.

What should be documented?	Specifics to include
Model documentation is an integral part of AI model test and evaluation. Thorough documentation is pivotal to promote transparency, enhance reproducibility, facilitate model understanding, and enable responsible usage. A model's documentation should encompass the following: Model Description • Provides details of the model's architecture and function. It includes the problem it is intended to solve, type (e.g., regression, classification, neural network), its input and output types, and the techniques used in its training. It should also note unique characteristics that distinguish it. Training Process • Documentation should describe the training methodology, including any preprocessing steps, the type of learning algorithm(s) employed, model parameter settings, performance metrics used, and any techniques applied to avoid overfitting (like regularization, dropout, or data augmentation). Information about the computing resources used can also be mentioned if applicable.	• Model details • date, version, type, architecture, training algorithms, parameters, fairness constraints, contact information, citation details, license info • Intended use • Performance metrics • Training data • Quantitative analysis • Ethical considerations and recommendations

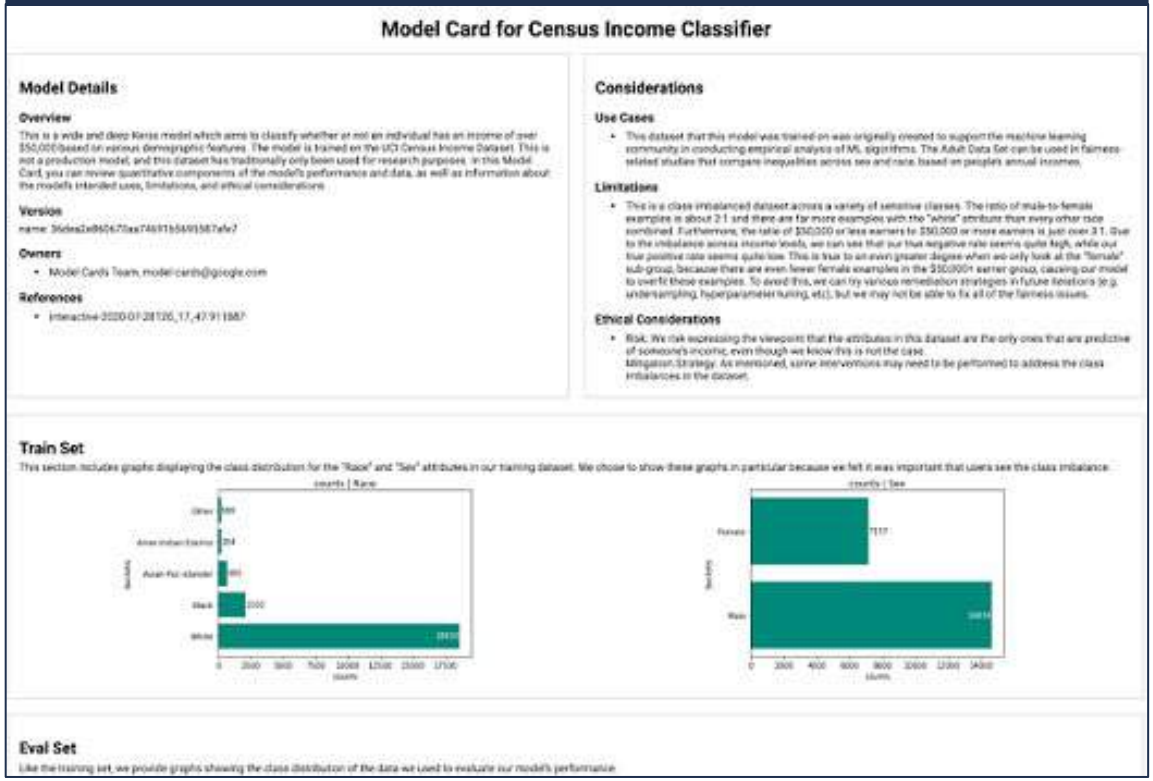
Model Card

Below is an example model card from:
https://www.tensorflow.org/responsible_ai/model_card_toolkit/guide

What should you know?

Introduced by researchers at Google, a model card serves as a concise yet comprehensive "report card" for a trained ML model. It is designed to provide essential information that allows evaluators, users, and stakeholders to understand the model's behavior and make informed decisions about its deployment. A model card typically includes the categories of documentation mentioned on the prior page. To aid their creation, some organizations have built model card tools. Two popular examples are Huggingface's Model Card Writing Tool and Google's Model Card Toolkit.

Model Card Example



Model Card for Census Income Classifier

Model Details

Overview

Version

Owners

References

Considerations

Use Cases

Limitations

Ethical Considerations

Train Set

Eval Set

CONCLUSION

07

+ *Testers should consider the operational use case, the environment, the data, the learning paradigm, and the algorithms in planning and conducting a rigorous evaluation of the AI model.*
+ *Solid documentation and deliberate curation of the data, the model, and surrounding context is required for repeatability.*

AI Model T&E Framework

Rigorous and comprehensive T&E for AI models is vital and demands innovation and adaptation as the field rapidly evolves.

You should know

Solid T&E strategy development remains vital

AI models are in a period of very rapid experimentation and innovation. Self-education is necessary to maintain awareness of new test requirements and approaches.

What should testers ask?

? Is our test strategy comprehensive?
? Are the test methods appropriate?
? Was the data split reasonable?
? How does the fielded context impact the T&E plan?
? Did the documentation sufficiently support the test planning & conduct

The bottom line

- Data is the foundation of the performance of an AI model regardless of the algorithm, learning paradigm, and parameter settings.
- Insight into the lifecycle of AI model development enables rigorous T&E
- Documentation should cover everything needed to inform testing for the fielded model.