A TECHNOLOGICAL AND INTELLECTUAL HISTORY

History of ML and AI Diagnostics, Observability, and Debugging

From statistical residuals and reasoning traces to training diagnostics, model monitoring, foundation-model evaluation, and agentic observability

Historical development of diagnostic artifacts, analysis methods, and mechanisms

Concept and direction by Dmitry Vostokov, DumpAnalysis.org

Developed with AI assistance (GPT-6 Astra Max and Fable 5.1 Extra) and checked against the sources listed in this book.

Version 7 · September 2026

Historical synthesis with 143 references, chronology, glossary, and evidence taxonomy

Abstract

Here we consider the historical development of diagnostics, observability, and debugging for machine learning and artificial intelligence systems. Such systems may execute without any software exception and still produce incorrect results. For example, a model may use information that is unavailable during deployment, depend on an accidental feature of its training data, or produce confident predictions for a different population. An agent may also return the expected answer after performing an incorrect sequence of actions. To diagnose these problems, we need suitable artifacts and methods for their analysis.

We follow several related threads of thinking: statistical residuals and process control; feedback and adaptive systems; symbolic reasoning and justification; validation, benchmarks, and uncertainty; training dynamics and numerical computation; interpretation, testing, and causal analysis; and production monitoring. We also consider reinforcement learning, generative models, and agents. Their diagnostic artifacts include datasets, checkpoints, gradients, predictions, explanations, software traces, and records of external actions. The same artifact may support different diagnostic questions, depending on the model of expected structure and behavior.

By ML and AI observability, we mean the designed ability to obtain evidence about otherwise hidden system state and behavior. Diagnostics includes recognizing a problem, distinguishing possible causes, and interpreting the available evidence. Debugging uses such findings to investigate and verify changes. We distinguish these activities from using AI to diagnose other systems, such as a patient or a computer. Throughout this history, we examine what became observable, which analysis methods became possible, and what remained outside their scope. An improvement in a reported score is only one part of this development.

How to read this history

This book follows related diagnostic ideas across statistics, artificial intelligence, software engineering, operations, and other disciplines. We do not assume that these ideas have a single origin. For example, a statistical residual, an expert-system explanation, and an agent trace are all artifacts for analysis. However, their interpretation requires different knowledge, and their presence does not establish the same kind of explanation.

We arrange the chapters around ideas and their practical realization. A learning curve requires a training and evaluation procedure. A feature attribution requires a reference state and a method for varying features. An alert requires a baseline and a rule for deciding when a difference matters. We can analyze an individual example, a model, a pipeline, or the surrounding organization. When moving between these levels, we need to preserve their connections and the provenance of the evidence.

References in square brackets point to the linked bibliography. We use original papers, archival publications, official records, and research syntheses where appropriate. A publication date does not

necessarily identify the first appearance of an idea; some methods have earlier mathematical formulations or independent implementations. We also distinguish documented historical developments from our diagnostic interpretation of them. In the case studies, we consider the interactions among data, models, software, people, and their working environment before discussing possible mechanisms.

Neighboring practices and their primary questions

Table 1. Related practices and the diagnostic questions they address.

Practice	Primary question	Characteristic evidence
Statistical diagnostics	Do residuals, influence, uncertainty, or assumption checks reveal mismatch between a model and its data-generating process?	Residuals, likelihoods, leverage, goodness-of-fit tests, posterior checks, control charts
ML evaluation	How well does a learned predictor generalize under a stated sampling design, task, metric, and decision threshold?	Holdout sets, cross-validation, benchmark scores, calibration, subgroup and stress tests
AI verification and validation	Does the system satisfy intended requirements, constraints, and safety claims in its operating context?	Specifications, scenarios, test oracles, simulations, assurance cases, acceptance evidence
Interpretability and explainability	What input evidence, representation, rule, mechanism, or counterfactual supports a behavior?	Coefficients, trees, attributions, concepts, circuits, examples, causal interventions
Robustness and security	How does behavior change under perturbation, attack, distribution shift, or component failure?	Adversarial tests, fuzzing, invariance checks, out-of-distribution and fault-injection evidence
Fairness and accountability	Who experiences errors or benefits, under which definitions, and can decisions be audited, challenged, and repaired?	Disaggregated outcomes, documentation, impact assessments, audit trails, governance records
MLOps and reliability engineering	Can the data, code, model, configuration, and service be reproduced, deployed, monitored, and rolled back safely?	Lineage, registries, validation gates, telemetry, service objectives, incidents and runbooks
AI incident response	What happened, why did controls fail, what was the impact, and what prevents recurrence?	Timelines, traces, model and data versions, user reports, containment and corrective actions

SEVEN RECURRING TRANSITIONS

- From manually inspected residuals to automated evaluation of populations and their subsets
- From explicit rules and proof traces to learned representations, features, and circuits

- From an aggregate score to separate evidence about error, uncertainty, robustness, and consequences
- From a fixed evaluation dataset to continuing analysis of change and feedback
- From an isolated model to the connected data, software, service, and human system
- From describing an observation to testing possible mechanisms through intervention
- From predictions and generated outputs to agents whose activities and side effects require trace analysis

Contents

Seven parts · 35 chapters · three reference appendices

Table 2. Book contents and opening pages.

PART I

Foundations: making learning and reasoning inspectable

1. Vocabulary, scope, and historical method

We begin by distinguishing incorrect computation from an incorrect result in the problem domain. An ML program may complete every operation successfully while its predictions are unsuitable for use. For example, low training loss does not establish that the training data represent the deployment population. Similarly, a high benchmark score does not establish that an agent performed only the permitted actions. We therefore need to ask separately about computation, learning, task behavior, and the consequences of deployment.

For our purposes, we consider three nested diagnostic objects. The model includes its parameters, representations, optimization state, and predictive behavior. The production system also includes data collection, transformations, orchestration, serving, and feedback. The surrounding socio-technical system includes users, operators, institutions, and affected people. Work on technical debt and AI risk makes these additional dependencies explicit. [1, 3] Model-based diagnosis provides a related formal distinction among a system description, observations, and candidate faulty components. [2]

Observability does not imply that we can reconstruct every internal state. We use the term in relation to a diagnostic question. A log without model and data versions may be unsuitable for comparing two incidents. A dashboard may hide a failing subgroup by averaging its results with a larger population. We need evidence that helps us distinguish possible explanations. Breiman's discussion of statistical modeling cultures is relevant here because different modeling approaches expose different aspects of a problem. [4]

When reviewing earlier work, we should avoid assigning contemporary meanings to historical terms. Expert-system explanations were not early versions of every present-day explanation method. Nevertheless, we can compare their diagnostic organization: an observation, an expectation, a discrepancy, possible causes, and a method for checking them. This is also compatible with our pattern-oriented view of software diagnostics, where we identify recurrent structures in artifacts and then investigate their possible mechanisms. [142] The comparison provides a method of analysis; it does not establish a direct historical influence between all the methods discussed.

2. Before machine learning: residuals, goodness of fit, and statistical quality control

Statistical diagnostics already treated the difference between observations and a fitted model as material for further analysis. Such differences, called residuals, could reveal curvature, changing variance, dependence, or unusual observations. Influence analysis asked whether a small part of the data had a large effect on the result. Therefore, obtaining fitted coefficients did not complete the investigation. We also needed to examine the structure of the remaining disagreement and relate it to the assumptions used in fitting.

The iris example illustrates the connection between measurement and classification. Edgar Anderson collected the measurements that R. A. Fisher analyzed in his 1936 discriminant study. [5] The resulting dataset later became a familiar example in machine learning. When reusing it, we should also consider how the measurements were obtained and what the classes represented. A table of numbers does not preserve every condition of its collection, and an accurate separation of its classes does not automatically transfer to another sampling situation.

Shewhart's control charts connected measured variation with rules for investigating a process. [6] They distinguished routine process variation from observations suggesting a change that required attention. A chart signal did not identify the cause of that change. It supplied a reason to collect or examine further evidence. We can recognize a similar organization in drift monitoring: select a baseline, measure departures, and investigate significant differences. Dependence, seasonality, repeated testing, and unsuitable limits can affect both the historical and contemporary forms of this process.

A diagnostic check has a limited scope. If it finds no discrepancy, we have not proved that the model is correct. We have established that this particular check, applied to the available data, did not reveal the discrepancy it was designed to find. This distinction becomes useful when we consider validation sets, robustness tests, and AI red teaming. Each provides evidence about certain possible failures and leaves other failures unexamined.

Bayesian model checking extends this line of inquiry. We can simulate observations from a fitted model and compare their characteristics with the observations we actually collected. Gelman, Meng, and Stern developed posterior predictive assessment using discrepancies chosen for the question under investigation. [128] For example, a model may reproduce an average while failing to reproduce the variation or the frequency of extreme values. Such checking examines the model's adequacy for the data; it is distinct from checking whether the numerical procedure used to fit the model has worked.

3. Cybernetics, feedback, error, and adaptive systems

Cybernetics placed diagnostic error in a feedback loop. We compare an observed state with a desired state, act on the difference, and observe the result. Wiener's synthesis connected these ideas across communication and control in machines and organisms. [7] The diagnostic object is therefore larger than a sensor or a controller. A problem may involve the measured process, the sensor, the reference value, the controller, or a delay between them. We need their relationships to interpret an individual observation.

McCulloch and Pitts described idealized neural elements in terms of logical computation. [8] Turing later considered machine intelligence through observable performance and the possibility of learning machines. [9] We can see two complementary viewpoints here. One examines the internal organization that produces behavior; the other examines behavior at the system boundary. Boundary tests remain useful when the internal structure is unavailable. However, a successful test does not tell us which internal mechanism produced the result or whether it will work in a different context.

An adaptive system changes during use. Its earlier observations may therefore describe a different system state. An online learner changes parameters, a recommender changes what users encounter, and an operator changes behavior after an alert. These changes may reduce a problem or introduce further problems. For diagnostic analysis, we need to relate observations to policy versions, time, and previous interventions. A final model file contains only part of this history.

Feedback also helps explain why a collection of locally correct components may behave incorrectly together. A delay may turn a corrective response into an excessive response. Repeated adaptation may invalidate the baseline used by a monitor. In production ML, similar questions arise around retraining, ranking policies, delayed outcomes, and human overrides. We therefore consider the closed loop as a diagnostic object in addition to its separate components.

4. Symbolic AI: reasoning traces, explanation facilities, and truth maintenance

Symbolic AI produced artifacts that exposed parts of its internal reasoning. The Logic Theory Machine represented goals and heuristic search operations in a form that could be examined after execution. [10] We could ask which transformation was used, where a search branched, or which premise was missing. Such a trace described the implemented computation. It did not necessarily describe human reasoning, but it provided a sequence that connected an initial problem with a computed result.

Explanation became part of the interface of expert systems. MYCIN could answer questions about why information was requested and how a conclusion had been reached by displaying relevant rules and intermediate conclusions. [11] We should distinguish this explanation from validation of the medical knowledge or the supplied facts. A faithful record of rule execution cannot establish that the knowledge base is complete or that its assumptions apply to a new patient population. The distinction remains useful when interpreting explanations produced by learned systems.

Truth-maintenance systems preserved the justifications for beliefs and revised dependent beliefs when assumptions changed. Doyle made these dependencies explicit computational objects. [12] Reiter's theory of diagnosis related a system description and its observations to possible faulty components. [2] Both approaches show why recording a conclusion alone is insufficient. If we also preserve its dependencies, we can examine what must change when an observation contradicts it. This provides a useful comparison with later provenance graphs and fault-localization methods.

Readable rules did not remove the need for diagnostics. A large rule base could contain inconsistent assumptions, omit relevant cases, or use predicates whose meaning changed outside the original environment. Learned systems later replaced many explicit rules with parameters obtained from data. Their explanation methods attempted to recover different kinds of structure. We can compare these systems by asking which dependencies are recorded, which can be reconstructed, and which remain outside the available artifacts.

5. Perceptrons, ADALINE, and early learning diagnostics

The perceptron made adjustable parameters central to learning. Rosenblatt's model connected examples, adjustable connection strengths, and learned discrimination. [13] Widrow and Hoff's adaptive switching circuits used an error signal to adjust weights. [14] The resulting diagnostic artifacts included error counts, weight values, and convergence behavior. We could now investigate both the current decision and the sequence of updates that produced the decision function.

Samuel's checkers program provides another early example of learning examined through recorded performance. Self-play and evaluation functions allowed changes in the program to be compared over successive games. [15] If performance stopped improving, several explanations were possible. The representation might omit useful information, the experience might be insufficient, or the evaluation procedure might reward an unsuitable strategy. These alternatives show why a learning result depends on the complete experimental arrangement.

Minsky and Papert examined representational limitations of particular perceptron architectures. [16] Such analysis concerns a class of models, whereas a failed training run concerns one realization of a model.

We should keep these levels separate. Repeating training cannot remove a representational impossibility, but a limit proved for one architecture should not be extended to all neural networks. The historical discussion of perceptrons often blurred this distinction when explaining changes in research interest.

Early learning systems therefore provided two related forms of diagnostic evidence. We could inspect decisions on examples, and we could examine the structure and changes of the learning mechanism. Contemporary ML analysis continues to move between these viewpoints. A failed example identifies behavior to explain; weights, gradients, activations, and training records may help explain how it arose. Neither viewpoint, taken alone, covers every possible problem.

PART II

From statistical diagnosis to machine-learning evaluation

6. Holdout testing, cross-validation, resampling, and generalization

Holdout testing separates the examples used to construct a model from those used to evaluate it. This separation is useful only when it also holds in the preparation and selection process. For example, related records may occur on both sides of the split, or a transformation may be fitted before the split is made. Future information may also enter a feature. In these cases, a test file exists, but the intended independence of evaluation has already been lost.

Cross-validation makes repeated use of limited data by changing which part is withheld. Stone studied cross-validatory model choice, and Efron compared methods for estimating prediction error through resampling. [17, 18] Kohavi subsequently examined practical choices for classification. [19] The folds still require a suitable unit of independence. If several records belong to one patient or device, separating individual records may answer a different question from separating patients or devices. Time-ordered prediction requires similar care with the direction of the split.

We can use the difference between training and validation performance as a diagnostic sign. A large difference may suggest overfitting. Poor performance on both sets leaves other possibilities, including unsuitable features, failed optimization, inconsistent labels, or limits of the available information. Learning curves help us compare these possibilities as training time or sample size changes. Their interpretation still depends on whether the evaluation data represent the intended use of the model.

Repeated inspection gradually changes the role of a test set. A team may revise features, parameters, or prompts after studying its errors. Even when the examples are never added to training, they have influenced model selection. Therefore, we should describe how a dataset was withheld, who examined its results, and which decisions followed. The diagnostic artifact includes this history as well as the final score and the files used to compute it.

7. Bias–variance, regularization, early stopping, and learning curves

Bias–variance analysis distinguishes systematic approximation error from sensitivity to the sampled training data. Geman, Bienenstock, and Doursat examined this distinction for neural networks. [20] It provides useful diagnostic questions: does the model repeatedly make a similar error, or does the result vary substantially when the training sample changes? The mathematical decomposition has assumptions,

so we should not use its vocabulary as a complete explanation of every learning curve or modern training regime.

Regularization constrains the fitted solution. Ridge regression uses shrinkage to stabilize estimation, while the lasso combines shrinkage with variable selection. [21, 22] Neural-network constraints may also arise from architecture, augmentation, noise, or the optimization procedure. We can compare validation behavior under different strengths of regularization. Such a comparison may reveal sensitivity, but selecting its best result is itself another use of the validation data.

Early stopping uses the duration of training as a control on the fitted model. Training loss may continue to decrease after validation performance begins to deteriorate. Prechelt examined the practical choice of stopping time and its relation to generalization and computation. [23] We need to retain checkpoints and the validation sequence to investigate this decision. A noisy change in one measurement is different from a sustained deterioration, and a stopping rule must distinguish them under its stated conditions.

A learning curve is an artifact with several possible interpretations. A flat region may indicate exhausted capacity, an optimization problem, inconsistent labels, or a metric that hides small changes. A widening training–validation difference may arise from overfitting or from different data populations. We therefore compare controlled variations in data size, model capacity, regularization, and initialization. Preserving what changed between runs is part of the analysis, because the shape of a curve alone does not identify its mechanism.

8. Shared datasets and benchmarks: from Iris and UCI to MNIST, ImageNet, and GLUE

Shared datasets allowed researchers to compare algorithms on common material. Anderson's iris measurements, analyzed by Fisher in 1936, became a small and widely reused example. [5] The UCI Machine Learning Repository, established in 1987, provided many further datasets for comparison and teaching. [24] This reduced the work needed to reproduce an experiment. However, a dataset may gradually acquire a role larger than its original scope, so its collection conditions should remain part of the interpretation.

MNIST joined a defined recognition task to a stable training and test arrangement. [25] ImageNet increased the scale and range of visual categories. The 2012 convolutional-network result combined this data with accelerators, architecture, regularization, and training decisions. [26, 27] A leaderboard records the resulting comparison compactly. To diagnose why one result differs from another, we need to recover the relevant differences in those components, including augmentation, ensembles, computation, and the distribution of errors.

GLUE combined several language tasks into one evaluation suite. [28] Such a suite gives a broader view than a single task, but it also introduces choices about task weighting and aggregation. When leading models approach the highest measurable scores, the benchmark becomes less useful for distinguishing them. We should interpret this as a limitation of that measurement arrangement. It does not establish that all the underlying language problems have been solved.

We can analyze a benchmark as a diagnostic instrument. Its labels, exclusions, duplicates, population composition, and intended use determine which claims it can support. Public reuse also creates opportunities for contamination. Consequently, benchmark maintenance includes examining and replacing the measurement instrument itself. Comparing models without examining their common test material may reproduce the same error across an entire set of experiments.

9. Metrics, ROC analysis, precision–recall, calibration, and decision costs

A metric summarizes selected aspects of error. Accuracy counts correct decisions, whereas sensitivity, specificity, precision, and recall condition on different parts of a confusion matrix. The distinction matters when the classes have different frequencies or when different errors have different consequences. Before interpreting a score, we should identify the underlying cases and the decision it is intended to support. Otherwise, the same number may be used to answer several incompatible questions.

ROC analysis compares discrimination across thresholds. Precision–recall analysis emphasizes retrieval of the positive class and can be more informative under substantial class imbalance. [29, 30] Neither curve supplies the operating threshold by itself. That choice also depends on costs, prevalence, available capacity, and acceptable risk. For example, a useful ranking of cases may still produce too many false alerts at the threshold selected for deployment.

Probabilistic forecasts require a different comparison. The Brier score measures squared disagreement between a forecast probability and the observed outcome. [31] For calibration, events assigned a probability near 0.8 should occur at approximately that frequency among comparable cases. We can examine this relationship in reliability diagrams. However, an aggregate diagram may hide a different relationship within a class or subgroup, and a calibrated forecast may still offer little useful discrimination.

Metric analysis also asks how a score can improve while the intended result fails to improve. Predicting only the majority class may produce high accuracy in an imbalanced dataset. A text metric may reward overlap without establishing factual correctness. An average may hide a small number of severe failures. We therefore retain examples, subgroup results, and uncertainty alongside the summary. These additional artifacts help explain what the metric has compressed.

Unsupervised learning requires another distinction because reference labels may be unavailable. Rousseeuw's silhouettes, introduced in 1987, compare how an observation relates to its assigned cluster and to other clusters. [143] They help us inspect separation under the chosen dissimilarity measure. A high silhouette value does not establish that the groups have the intended meaning in the application. We should also examine changes under resampling, scaling, and alternative representations. Here, structural regularity in the data and usefulness of the resulting classification are separate questions.

10. Probabilistic prediction, uncertainty, ensembles, and conformal evidence

A confidence value is not automatically a reliable probability. Guo and colleagues examined miscalibration in modern neural networks and the use of temperature scaling on held-out data. [32] The calibration procedure belongs to a particular model and evaluation population. It may need to be repeated after a model or population change. We should also examine rare and consequential subsets, because good overall calibration does not establish calibration for every subgroup.

Ensembles provide another source of evidence through disagreement among trained models. Deep ensembles demonstrated the usefulness of this approach for predictive uncertainty. [33] Their interpretation depends on how the members differ. Models trained on similar data with similar procedures may confidently repeat the same error. Evaluation under dataset shift showed that uncertainty methods can behave differently when their original conditions change. [34] Therefore, agreement and disagreement both require context.

Conformal prediction produces prediction sets or intervals with coverage guarantees under specified conditions, commonly exchangeability of calibration and future examples. [35] The basic guarantee concerns coverage across the relevant sequence or population; it does not promise the same coverage for every individual input or subgroup. This is a useful example of a diagnostic claim whose conditions should accompany its result. Dependence, distribution change, and adaptive reuse require additional analysis rather than an unchanged interpretation of the original guarantee.

Uncertainty becomes operationally useful when it affects a decision. A system may request more evidence, defer a case, or route it to a human reviewer. If the workflow always takes the same action, adding an uncertainty display may not change the problem being diagnosed. We therefore need to inspect both the estimate and the response to it, including the authority and capacity available to handle uncertain cases.

Probabilistic computation has its own diagnostic artifacts. Gelman and Rubin's 1992 work compared multiple simulation sequences to assess convergence. [127] In Bayesian analysis, apparently stable

predictions may conceal poor exploration of the posterior distribution. We therefore distinguish convergence of the computational procedure from adequacy of the statistical model. Trace plots and comparisons among chains address the former; posterior predictive checks address the latter. Passing either kind of check does not replace the other.

11. Data diagnostics: missingness, outliers, leakage, duplicates, and label error

Data problems do not always interrupt execution. A pipeline may fill missing values, convert strings, or group unfamiliar categories and then continue training. The absence of an exception therefore tells us little about the meaning of the processed data. We need to examine schemas, ranges, relationships, timestamps, uniqueness, and provenance. For example, a successful join can still associate records with the wrong entity or time period.

Leakage is difficult to notice because it may improve the reported result. A model may receive an outcome proxy, future information, or a related record from the test population. Kaufman and colleagues examined the formulation and detection of such leakage. [36] We can investigate unusually strong performance by changing the split unit or removing suspect features. A collapse after splitting by patient, device, customer, or time is evidence to examine; it does not by itself identify every source of leakage.

Labels also require diagnostic analysis. Confident learning estimates possible label issues from predicted probabilities, and benchmark audits found test-label errors that could affect comparisons among models. [37, 38] A disagreement between a model and a label is a candidate for review. We should not automatically replace the label with the prediction. The disagreement may reflect an ambiguous task, multiple acceptable answers, changing circumstances, or different annotator interpretations.

Datasheets for datasets record collection purpose, composition, processing, intended uses, and maintenance. [39] These records help us connect a later error with earlier data decisions. Studies of data cascades showed how neglected data work in high-stakes applications could produce delayed effects. [40] Diagnostic improvement therefore also requires ownership and escalation procedures. A numerical quality check cannot substitute for knowing who can investigate the meaning and origin of a field.

PART III

Debugging data, training, and model behavior

12. Backpropagation, automatic differentiation, and gradient checking

Derivatives serve two related purposes in gradient-based learning. They guide parameter updates, and they expose how changes propagate through a computation. Wengert's automatic derivative evaluation and Linnainmaa's reverse accumulation supplied important computational foundations. [41, 42] Their place in this history should not be confused with a claim that the 1986 neural-network work invented every form of reverse differentiation. To analyze a gradient, we also need the graph, loss, and data path whose dependencies it represents.

Rumelhart, Hinton, and Williams demonstrated learning internal representations through back-propagation of error. [43] We can examine such learning through gradients, activations, loss components, and parameter updates. For example, an inactive branch may receive no gradient, or one loss component may dominate the combined objective. Non-finite values may identify a numerical failure. These are different diagnostic signs, although they can all appear in a run that fails to learn as expected.

Gradient checking compares a computed derivative with the change observed after a small numerical perturbation. It is particularly useful for custom operations and loss functions. The comparison requires suitable perturbation sizes, adequate numerical precision, and care at nondifferentiable points. We normally begin with a small, controlled computation. A successful comparison for selected parameters supports that implementation under the tested conditions; it does not certify every branch of a large training program.

Automatic differentiation can correctly differentiate a computation that expresses the wrong task. A misplaced mask, incorrect target alignment, or unintended broadcast may produce valid derivatives. We therefore combine gradient checks with checks of shapes, units, and graph structure. Small examples with known expected behavior are especially useful. They allow us to compare the intended computation with the one the framework actually performs before adding the complexity of a complete training run.

13. Loss landscapes, vanishing and exploding gradients, initialization, normalization, and optimization

Some learning problems become visible only through time. Loss may initially decrease and then remain unchanged, oscillate, or diverge. Hochreiter's 1991 thesis examined vanishing gradients before Bengio,

Simard, and Frasconi's 1994 analysis of long-term dependencies. Subsequent work investigated both vanishing and exploding gradients and the use of clipping. [44–46] Gradient norms, activation distributions, update magnitudes, and comparisons across sequence lengths provide artifacts through which we can investigate these mechanisms in a particular run.

Initialization determines the initial scale of signals and gradients. Glorot and Bengio related this scale and the choice of activation to training difficulty. [47] Batch normalization and Adam introduced further state and changed the dynamics of training. [48, 49] We therefore need more than a loss value when examining a checkpoint or a resumed run. Running statistics, optimizer moments, decay settings, and their restoration can explain differences that are not visible in the parameter values alone.

Loss-landscape visualization projects selected directions of a high-dimensional parameter space into an inspectable picture. [50] It can help us compare solutions and training paths, but the choice of directions and normalization affects the result. We should treat the picture as a view of the computation under those choices. It is not a complete map of the optimization problem, and apparent sharpness in one view is insufficient to explain all differences in generalization.

Controlled reduction connects these observations with debugging. We can first attempt to fit a small batch, use a synthetic example with known structure, or compare a single update with an independently calculated result. We can then restore augmentation, lower precision, and distributed execution one at a time. This procedure preserves a sequence of diagnostic comparisons. If a problem appears after a particular change, we obtain a smaller set of possible mechanisms to investigate.

Mixed-precision training added an explicit numerical dimension to this process. Micikevicius and colleagues combined lower-precision computation with measures such as higher-precision weight storage and loss scaling. [138] For diagnosis, we should distinguish small gradients lost through underflow from genuinely small derivatives. We can record scaling changes and non-finite values and compare the suspect step with a higher-precision computation. A reduced loss or a faster run does not establish numerical equivalence by itself.

Generative adversarial networks introduced coupled training of a generator and a discriminator. Goodfellow and colleagues formulated this adversarial learning arrangement in 2014. [132] The state of one model changes the learning problem faced by the other. A collection of plausible generated examples may therefore coexist with missing kinds of examples, commonly described as mode collapse. To investigate the run, we need to examine both participants, their update schedules, and the diversity of outputs. A single decreasing loss is insufficient for diagnosing this coupled process.

14. Interpretable structures: coefficients, rules, trees, and feature importance

Some model structures permit relatively direct inspection. A linear coefficient has an interpretation under its parameterization, a rule describes a condition, and a tree records a sequence of partitions. Classification and Regression Trees brought recursive partitioning, pruning, and validation into a systematic treatment. [51] We still need to examine the data and representation. Correlated variables or different encodings may alter coefficients and splits without substantially changing predictive behavior.

Ensembles distribute a decision across several models. Random forests combine resampling with feature randomness, while gradient boosting constructs successive corrections. [52, 53] A feature-importance ranking summarizes part of this arrangement. However, importance based on impurity reduction, permutation, or gain does not have one interchangeable meaning. Correlated inputs, category counts, and the evaluation population can affect these measures. We should state the method before interpreting the ranking.

We also distinguish a description of one decision from a description of typical model behavior. A tree path identifies the conditions followed by one record. A partial dependence curve summarizes a modeled response after averaging over other features. Permutation importance examines predictive performance after disrupting feature information. These operations answer different questions. None, without additional assumptions, shows that changing the corresponding property in the world will cause the predicted outcome to change.

Directly inspectable models allow domain knowledge to enter diagnostic analysis. We can examine thresholds, missing-value routes, and interactions that appear inconsistent with the task. Such inspection may reveal a problem before deployment. As models become more complex, a simpler surrogate may help us understand their behavior. We should preserve the distinction between the surrogate and the original computation and check where the approximation is adequate.

15. Neural visualization: activations, feature visualization, saliency, and concepts

Neural visualization makes selected internal responses available as images. Saliency methods relate outputs to changes in input pixels, and deconvolutional approaches help examine patterns associated with activations. [54, 55] A visually coherent result is useful for forming a hypothesis, but it is not sufficient evidence for the hypothesis. We need to identify which computation produced the image and what changes in the model or input would change it.

Feature visualization searches for inputs that strongly activate selected parts of a network. The Distill treatment showed how regularization and other choices affect the resulting images. [56] Such an image is an artifact of both the model and the visualization procedure. We should not interpret it as a direct picture of a concept stored at one location. A feature may be distributed across units, and a unit may respond to several different structures.

Concept activation vectors use sets of examples to define a direction associated with a human-specified concept. TCAV tests model sensitivity in relation to that direction. [57] This connects analysis with domain vocabulary, such as color or texture. However, the examples, comparison set, network layer, and statistical test become part of the diagnostic method. An apparently meaningful concept label does not remove the need to inspect how that concept was constructed.

Visualization therefore creates an additional object for testing: the diagnostic instrument itself. We can ask whether a picture changes when the learned model changes and whether an intervention on the suggested feature changes the relevant behavior. The second question is necessary when we make a causal claim. These checks help separate an attractive presentation from evidence about the computation it is supposed to explain.

16. Reproducibility, seeds, experiment tracking, versioning, and provenance

A reported model result depends on more than its source code. Data order, random seeds, preprocessing, driver versions, accelerator kernels, and checkpoint selection may affect the computation. Reproducibility programs made such details part of the research record. [58] For diagnostic work, these details allow us to compare two runs whose outcomes differ. If they are omitted, we may attribute a difference to an algorithm when it originated elsewhere in the experimental environment.

Deep reinforcement learning made variation among runs especially visible. Henderson and colleagues examined the effects of seeds, environments, and implementation choices on reported improvements. [59] Raff studied independent reproduction as an outcome to be measured. [60] These investigations remind us that a successful repetition is evidence about a procedure, whereas a single favorable run is evidence about one realization. We need to preserve this distinction when making comparative claims.

Agarwal and colleagues subsequently showed how conclusions about deep reinforcement-learning benchmarks can change when uncertainty across a small number of runs is included. [131] They advocated interval estimates, performance profiles, and more robust aggregate summaries. Here, the diagnostic object is the comparison itself. We should examine how tasks, seeds, and aggregation contribute to the reported difference before treating that difference as an algorithmic improvement.

Experiment tracking connects parameters and metrics with code identifiers and saved artifacts. MLflow is one example of this development. [61] Instead of relying on a notebook or recollection, we can inspect a record of each run and compare its dependencies. The record remains incomplete if an external dataset, environment variable, or manual change is not captured. A stored model identifier helps only when we can recover the material that gave the identifier its meaning.

Reproducibility may refer to exact repetition of a computation, repetition of a statistical conclusion, or independent implementation of a described method. These requirements are different. We should specify the intended level and collect artifacts accordingly. For example, identical predictions on one machine do not establish the stability of a conclusion across samples, while modest numerical differences do not necessarily invalidate a statistical result.

17. Pipeline validation, training–serving skew, and slice-based error analysis

Production ML connects ingestion, transformation, feature calculation, training, validation, packaging, and serving. A failure may occur between these stages even when the model itself is unchanged. An online feature may use different units, an old category vocabulary, or the wrong timestamp. TFX provided an example of organizing such work as a production pipeline with reusable components. [62] This made the interfaces and intermediate artifacts explicit diagnostic objects.

Training–serving skew is a difference between the conditions used to train a model and those used to apply it. Data-validation systems compare schemas and distributions and look for anomalies along the pipeline. [63] These checks are useful when malformed data would otherwise pass through successfully. However, a field may retain its name and type while its collection policy changes. Structural compatibility therefore needs to be considered together with semantic compatibility.

Slice analysis examines errors within selected subsets of the data. We may group cases by language, device, class, source, or population and then inspect their results separately. The subset should contain enough evidence for the intended inference. It should also correspond to a meaningful diagnostic question. Automatically discovered slices can suggest where to investigate, but their apparent differences may require further validation on independent data.

Pipeline checks and behavioral checks complement each other. The former examine whether the expected fields, versions, and transformations are present. The latter examine whether results meet the task requirements. We can combine them to localize a discrepancy to the data path, the trained model, the serving path, or the population. This is a form of differential analysis: we compare corresponding artifacts and preserve where their structures or behaviors begin to differ.

Underspecification explains another source of variation. D'Amour and colleagues showed that a training pipeline can produce models with similarly strong test scores but materially different deployment behavior. [134] The existing tests do not constrain all the behaviors required in use. We therefore need to compare such models on task-relevant stress tests and slices. A tie on the original benchmark does not establish diagnostic equivalence between the resulting artifacts.

18. Dataset shift, out-of-distribution detection, drift, and continuous model monitoring

Deployed systems encounter changing conditions. Inputs change, class frequencies vary, labeling practices are revised, and users adapt to the system. Concept-drift research examined different forms of change and methods for adaptation. [64] We should distinguish a change in inputs from a change in the relation between inputs and outcomes. Both may produce an alert, but they imply different investigations and may require different responses.

Label shift is a particular case: class proportions change while the input distribution within each class remains stable. Lipton and colleagues investigated detection and correction under explicit assumptions. [65] Out-of-distribution detection addresses other departures from familiar data. The success of a detector on a selected anomaly dataset does not establish success on every future departure. We need to identify the kinds of change represented in its evaluation.

Continuous monitoring combines evidence from inputs, predictions, uncertainty estimates, service behavior, and later outcomes. Model assertions express relationships that should hold and can flag suspicious cases without a complete test oracle. [66] The monitoring instruments themselves may fail under shift. Work on predictive uncertainty illustrates why we need to test them when their original data conditions no longer hold. [34]

A drift signal identifies a difference to investigate. It does not establish that the model requires retraining. The changed input may be irrelevant to the decision, and an unchanged aggregate distribution may hide a conditional failure. We should connect each alert with its baseline, responsible owner, and possible follow-up observations. A repair should then be checked against the relevant outcome, including any feedback created by the repair itself.

PART IV

Interpretability, robustness, fairness, and causal testing

19. Global and local explanations: partial dependence, LIME, and SHAP

Model-agnostic explanation examines behavior without requiring one particular internal model structure. Partial dependence averages modeled responses after varying selected features. Permutation methods compare performance before and after disrupting feature information. We can use these operations to investigate which inputs affect predictions under the chosen procedure. They are operations on a model and its data representation. Interpreting them as interventions in the real world requires further assumptions.

LIME approximates a prediction locally with a simpler model fitted to perturbed examples. [67] Its diagnostic use depends on the neighborhood, representation, weighting, and quality of the approximation. If two explanations differ, we need to investigate both the predictor and the explanation procedure. The difference may reflect genuine variation in model behavior, different sampled perturbations, or an unstable local approximation. Saving only the final explanation would lose these distinctions.

SHAP relates additive feature attributions to Shapley-value properties. [68] It provides a common way to decompose a prediction relative to a reference. We still need to specify what it means for a feature to be absent and how the background data are used. Correlated features make this choice particularly important. Numerical precision in an attribution does not eliminate ambiguity about which question the decomposition answers.

Local and global explanations have different scopes. A local approximation may describe one region well and fail elsewhere. A global average may hide several different behaviors within the population. We therefore record the output, population, reference state, and approximation method associated with an explanation. This also helps distinguish predictive questions from causal questions and from questions about whether a decision is justified.

20. Attribution quality: integrated gradients, Grad-CAM, and sanity checks

Integrated gradients accumulates gradients along a path between a reference input and the observed input. Its formulation uses properties such as sensitivity and implementation invariance. [69] The reference remains part of the analysis. A zero value, a blurred image, and another meaningful example describe different comparisons. When interpreting the resulting attribution, we should preserve both the path and the reason for selecting its starting point.

Grad-CAM uses gradients associated with convolutional feature maps to produce a class-related localization map. [70] The highlighted region can help us select material for further investigation. It is not a direct recording of an attention mechanism, nor does it prove that the model recognized the same object or used the same reasoning as a human observer. Resolution, processing, and the selected output affect what the map shows.

Adebayo and colleagues tested whether saliency methods depended on learned parameters and labels by randomizing them. [71] Some visually convincing results changed less than expected. Later debugging tests examined whether explanation methods revealed deliberately introduced data or model problems. [72] These experiments shifted the question from whether an explanation looked reasonable to whether it responded to changes it was expected to detect.

We can therefore apply diagnostic reasoning to explanations themselves. An explanation may faithfully describe a computation but be difficult to use. It may also be easy to understand while describing the wrong mechanism. Stability is useful only in relation to the changes that should preserve or alter the result. We need to specify the intended use of an explanation before deciding which tests it should pass.

21. Mechanistic interpretability: circuits, sparse features, and causal interventions

Mechanistic interpretability attempts to identify the computation responsible for a behavior. The circuits approach examines features and their interactions as parts of a computational graph. [73] We can compare this activity with reverse engineering, where an observed result is connected to internal operations. The analogy has limits: learned features may be distributed, and a single unit may participate in several computations. Selecting the unit of analysis is therefore part of the investigation.

Work on transformer circuits decomposed attention-only models into operations that could be examined separately and in combination. [74] Such analysis made heads and paths into candidates for explaining

particular behaviors. It also showed why a neuron need not correspond to one human concept. We may need to examine directions or combinations in activation space to find a more useful representation of the computation.

Dictionary learning and sparse autoencoders attempt to recover more interpretable features from those activations. Studies progressed from smaller transformers to large collections of features in a deployed language model. [75, 76] A feature name summarizes an interpretation of activating examples. We should treat that interpretation as a hypothesis and examine whether changing the feature changes the relevant behavior. A plausible label is not itself a demonstrated mechanism.

Circuit tracing in 2025 used an interpretable replacement model to construct attribution graphs for particular prompts. [77] This provided another view of the computation, with limitations introduced by approximation and graph selection. We can see a historical movement from inspecting weights to examining units, distributed features, and candidate causal paths. Each additional view also requires checks of the instrument used to construct it.

22. Influence functions, training-data attribution, counterfactuals, and causal diagnostics

We can investigate a prediction through its current input, through the model's internal computation, or through the training examples that influenced it. Influence functions approximate how changing the weight of a training example would affect a fitted model and a test result. [78] They can identify candidates for label review or further analysis. Their accuracy depends on the approximation and the optimization conditions, so the returned ranking is evidence to test rather than an exact reconstruction of all training effects.

TracIn estimates influence using gradient similarity at training checkpoints. [79] This makes retained checkpoints useful for investigating the relation between training examples and later predictions. Attribution remains relative to the training process. A contribution assigned to one example is not a statement about the responsibility of its author. Duplicated, correlated, and distributed evidence can also make an individual assignment difficult to interpret.

Counterfactual explanations identify changes that would alter a model's decision. [80] They may help a user understand which conditions matter to the predictor. However, a mathematically small change may be impossible or inappropriate in the application. It may change an immutable property or violate a relation among features. We therefore distinguish a counterfactual input to the model from an action that a person can actually take and from a causal prediction about its consequences.

Causal diagnostics requires an intervention or a sufficiently supported causal model. Ablation, activation patching, data removal, and retraining can help test a candidate mechanism when relevant alternatives are controlled. We can compare this reasoning with truth maintenance and first-principles diagnosis: change a premise or component and examine whether the discrepancy remains. [2, 12] Similar diagnostic organization does not make the methods interchangeable, but it helps us state which counterfactual claim is being tested.

23. Adversarial examples, robustness evaluation, and attack-aware testing

Adversarial examples showed how carefully selected perturbations could change neural-network predictions despite appearing small to a human observer. [81] The diagnostic example was produced by a search rather than sampled directly from ordinary use. This expanded evaluation from performance on a fixed dataset to behavior in selected neighborhoods of its examples. We need to retain the search conditions to understand what the resulting failure demonstrates.

Adversarial training and robust optimization evaluate behavior against defined classes of perturbations. [82] The norm, perturbation size, attacker knowledge, and search budget determine the scope of a robustness claim. A failed attack may indicate robustness, but it may also indicate an ineffective search or obscured gradients. Therefore, the attack is another instrument whose adequacy must be examined, especially when assessing a proposed defense.

Work on non-robust features suggested that some adversarial behavior arises from predictive statistical structure that differs from human perceptual priorities. [83] In this interpretation, the model may use information that is useful on the original benchmark but unsuitable for the intended application. Diagnostic changes may therefore concern data, representation, and objectives. Removing visibly malformed inputs alone does not address every such mechanism.

We should distinguish several meanings of robustness. Ordinary corruption, distribution change, malicious input, compromised training, and a failed component do not follow the same causal path. A test for one may provide little evidence about another. We therefore state the threat or failure model and preserve the conditions under which a counterexample was found. This also gives a more precise regression test after a proposed repair.

Privacy testing introduced failures that need not reduce prediction accuracy. Shokri and colleagues investigated membership inference: whether model outputs reveal that a record was used in training. [136] Backdoor research addressed a different problem. BadNets demonstrated models that retained strong ordinary performance while responding incorrectly to attacker-selected triggers. [137] These

results made training provenance and special input conditions part of the diagnostic scope. A clean validation score does not establish either privacy or the absence of a hidden trigger.

24. Testing learned systems: metamorphic tests, neuron coverage, fuzzing, and behavioral checklists

A complete expected answer is often unavailable for a learned system. Metamorphic testing uses expected relations between inputs and outputs instead. For example, a harmless image transformation may be expected to preserve a class, or a paraphrase may be expected to preserve sentiment. We should first establish that the relation is valid for the task. Otherwise, a generated test pair can introduce an incorrect specification and make a correct change in behavior look like a fault.

DeepXplore combined differential testing across models with neuron coverage to generate difficult cases. [84] TensorFuzz used coverage-guided fuzzing to find numerical and behavioral anomalies. [85] These methods connect learned-system testing with earlier software-testing techniques. However, activation coverage does not directly measure coverage of meaningful behaviors. A disagreement among models also identifies a case to investigate without necessarily identifying which model is wrong.

CheckList organized language-model testing around capabilities and tests of minimum functionality, invariance, and expected directions of change. [86] This gives the analyst a way to state behaviors that a random test sample may miss. Templates make the tests reusable, but their meaning remains dependent on the task and language context. Test construction is therefore part of diagnostic knowledge, rather than a neutral step that can be omitted from the record.

A failing test can become a starting point for debugging. We inspect related cases, formulate possible causes, change the relevant component, and repeat the comparison. A regression test preserves the discovered failure for later releases. The accumulated suite is useful only within its coverage. New populations, capabilities, or operating conditions may require additional tests even when every earlier test continues to pass.

Formal verification supplies a complementary kind of evidence. Reluplex, published in 2017, checked stated properties of ReLU networks and could return counterexamples, including for a prototype neural-network implementation of ACAS Xu, an airborne collision-avoidance system for unmanned aircraft. [135] Here, we ask whether a property holds throughout a specified input region, rather than only on sampled cases. The result still depends on the encoded property, network model, and input constraints. A proof about these objects is not a proof of unrestricted safety of the deployed system; an inconclusive or timed-out check is not a successful verification.

25. Fairness, subgroup evaluation, documentation, audits, and accountability

Fairness diagnostics examines how errors and benefits are distributed. Equality-of-opportunity formulations made one family of criteria explicit. [87] Other criteria answer different questions and may conflict with it. We therefore need to state the groups, outcomes, and decision process being examined. Selecting one numerical criterion cannot settle all disagreements about what the system should do.

Gender Shades exposed large intersectional differences in the performance of commercial gender-classification systems. [88] Its diagnostic significance includes the choice to examine combinations of attributes that aggregate accuracy could hide. Dataset composition, external testing, and subgroup results were considered together. We can apply the same analytical principle elsewhere without assuming that the study's categories or measures are suitable for every application.

Model cards proposed structured descriptions of intended use, evaluation, and ethical considerations. [89] Internal auditing frameworks connected such evidence with organizational stages and responsibilities. [90] The resulting documents are artifacts for analysis. They may become stale or omit an important condition. Their practical value increases when release decisions and incident investigations can be traced back to the claims and tests recorded in them.

The NIST AI Risk Management Framework, ISO/IEC 42001, and the EU AI Act represent different institutional approaches to managing AI-related risk. [3, 91, 92] We should distinguish a voluntary framework, a management-system standard, and legislation. Their publication does not make them interchangeable diagnostic methods. For this history, their relevance lies in connecting technical evidence with responsibilities, documentation, and change control. They do not prescribe one universal performance metric or make every deployment decision follow automatically from a score.

PART V

Production ML observability and MLOps

26. ML systems and hidden technical debt

Production ML made dependencies outside the model increasingly visible. Sculley and colleagues described entanglement, data dependencies, undeclared consumers, feedback loops, and configuration debt. [1] These become diagnostic problems when a failure cannot be localized within the component first reported as faulty. We may need to follow a feature back to its source or a prediction forward to another service before understanding the observed behavior.

Entanglement limits the usefulness of changing one component in isolation. Retraining may redistribute how a model uses its features, and a changed prediction may affect an undocumented consumer. An improved offline score may also accompany worse latency or other operational behavior. We therefore need a reproducible release configuration that connects the model with data, feature code, runtime, and interfaces. Rolling back only the model file may not restore the earlier system.

The ML Test Score expressed production readiness through checks of data, features, models, infrastructure, and monitoring. [93] We can read these checks as assertions about the surrounding system. Their value comes from making expectations explicit and testing them repeatedly. Conventional software tests remain necessary because a learned prediction is still produced by software components with ordinary implementation and integration defects.

Missing observability is one form of technical debt. If we do not know the lineage of an artifact, its consumers, or the feedback it receives, we cannot explain many changes in behavior. Stable identifiers, dependency records, ownership, and incident histories reduce this uncertainty. Replacing the learning framework without restoring these connections may leave the diagnostic problem unchanged.

27. MLOps: continuous integration, delivery, training, registries, lineage, and validation gates

MLOps extends integration and delivery practices to systems that change through new data and repeated training. The same source code may produce a different model on a later day. We therefore have several kinds of change to examine: code changes, data changes, training changes, and deployment changes. A release procedure may validate each new dataset, compare candidate models, and expose a selected candidate to a limited part of the production workload before wider deployment.

TFX organized analysis, transformation, training, validation, and serving into reusable production components. [62] Data validation and the ML Test Score supplied related checks for pipeline and release conditions. [63, 93] These arrangements make intermediate decisions more visible. However, an automated gate still requires a policy: what counts as failure, who can accept an exception, and which evidence must be retained when an exception is made.

Model registries identify saved artifacts and their deployment states. Lineage records connect them with data, code, parameters, environments, and approvals. Experiment tracking, including MLflow, helps preserve the path from a research run to a released package. [61] For diagnostic use, these dependencies should be recoverable. A registry record pointing to a mutable dataset may identify a name while failing to identify the data actually used.

A validation gate can examine several independent requirements. These may include schema compatibility, data freshness, subgroup regressions, calibration, robustness, memory use, and latency. Passing the gate means that the recorded requirements were satisfied under the recorded tests. We should preserve both the evidence and the policy version. Otherwise, a later incident may be impossible to compare with the conditions under which the release was accepted.

28. Serving diagnostics: latency, throughput, resource telemetry, and prediction traces

Inference also involves a running service. Requests may wait in queues, share batches, retrieve features, compile kernels, or repeat failed network operations. These activities can dominate latency and resource use even when the model computation is correct. Site reliability engineering supplied methods for service objectives, alerting, and incident analysis. [94] For an ML service, we need to connect these operational observations with prediction quality rather than treating them as unrelated monitoring systems.

A prediction trace can connect a request with its feature version, model, routing decision, post-processing, and resulting action. The required content depends on the diagnostic question. Durations may help locate a slow stage but provide little evidence about a changed answer. Conversely, recording complete inputs without stable identifiers may leave different services difficult to compare. Sampling, access control, redaction, and retention should be designed together with these diagnostic requirements.

OpenTelemetry developed from the merger of OpenTracing and OpenCensus. [95] It provided common infrastructure for traces, metrics, and logs, which ML systems could use alongside model-specific records. A prediction usually belongs to a larger request path. The reported model error may originate in feature retrieval, a routing policy, or post-processing. Correlating those stages allows us to investigate the request as one activity while retaining the different component boundaries.

Resource and runtime events can also help explain changes in numerical behavior or reproducibility. Out-of-memory recovery, kernel selection, reduced precision, or a different accelerator may coincide with a changed result. We need to align these events with batches, checkpoints, and deployments. This produces candidate relationships for testing. Separate dashboards showing events at roughly the same time are insufficient to establish a causal connection.

Quantized deployment adds a comparison between representations of the model. Jacob and colleagues described integer-arithmetic inference and training that accounts for quantization. [139] For diagnosis, we can compare the reference and converted models on the same inputs and inspect where intermediate values diverge. Calibration ranges, rounding, unsupported operations, and preprocessing are possible sources of difference. The deployed artifact therefore needs its own validation; its relationship to the original checkpoint should be preserved in the release record.

29. Feedback loops, delayed labels, human oversight, and incident response

Production outcomes may arrive late or only for selected cases. A recommender observes responses to displayed items, and a human reviewer may examine only cases routed for review. The missing outcomes are part of the diagnostic situation. We should not assume that observed labels form an unbiased sample of every possible action. Their relationship with the deployed policy must be examined before they are reused for monitoring or training.

Feedback arises when a prediction changes exposure or behavior and thereby changes future data. Hidden feedback was identified as a source of technical debt, and concept-drift research examined changing streams. [1, 64] We need the history of interventions to interpret such observations. Which policy was active? Which alternatives were available? Which outcomes could be observed? Without these details, a monitor may repeatedly confirm the consequences of the policy it is supposed to assess.

Off-policy evaluation makes part of this problem explicit. Dudík, Langford, and Li studied evaluation of a new contextual-bandit policy from records produced by an earlier policy, combining reward and action-selection models in a doubly robust estimator. [130] The records need sufficient support for the actions being evaluated. A method cannot recover reliable evidence about an action that the logging policy never took in the relevant context merely by reusing its observed rewards. Recording selection probabilities and checking uncertainty therefore serve a diagnostic purpose.

Reinforcement learning also requires us to inspect the reward itself. The 2016 paper Concrete Problems in AI Safety distinguished reward hacking, side effects, supervision difficulties, unsafe exploration, and distribution change. [129] An agent may increase measured reward while failing the intended task. We

can investigate this by comparing reward components, complete episodes, termination conditions, and external state changes. The same distinction applies when preference signals are used to train or select generative models.

Human oversight depends on its implementation. A reviewer needs time, relevant evidence, and authority to change the outcome. If these conditions are absent, a recorded approval may reveal little about the quality of the decision. Overrides, escalations, and disagreements can become useful diagnostic evidence about both the model and the workflow. Their collection should preserve privacy and should not discourage legitimate disagreement.

Incident response connects a signal with containment, investigation, repair, and subsequent checking. Model assertions, risk frameworks, and SRE practices supply different parts of this process. [3, 66, 94] The incident record should identify affected versions and populations and preserve the observations used to select a response. We then test whether the change addresses the recurring failure mechanism. Correcting the one visible example may leave other instances of the same problem unchanged.

30. Automated fault diagnosis for training and inference infrastructure

Large distributed training jobs produce artifacts across accelerators, networks, storage, compilers, and orchestration systems. A component may fail visibly, slow the job, or corrupt state without an immediate crash. Logs, collective-operation timings, hardware counters, checkpoints, and loss curves describe different aspects of the same execution. We need to relate them to ranks, hosts, batches, and topology before combining their evidence.

Aegis, developed at Alibaba Cloud, is a production system for diagnosing faults in AI training services. [96] It illustrates the movement from manually searching logs after an incident to continuous analysis informed by distributed training behavior. An automated result still needs supporting observations and an account of uncertainty. The operator must be able to distinguish a directly observed component failure from a cause inferred through a diagnostic model.

Using AI to analyze AI infrastructure introduces another diagnostic layer. A generated log summary may help select a hypothesis, but the hypothesis should remain connected to the original observations and topology. We also need to distinguish a retrieved fact from an inferred explanation. Otherwise, a coherent account may acquire the status of a root cause without a test that separates it from competing accounts.

Our pattern-oriented approach connects software execution artifacts across these layers. Python Debugging for AI, Machine Learning, and Cloud Computing applies this approach to Python and its

surrounding runtime. [97] A stalled training job, for example, may require thread, stack, memory, and communication analysis before any change to the learning algorithm is justified. We can isolate a rank, replay a batch, compare a checkpoint, or inject a known fault and then verify the resulting behavior.

We can also relate this process to the distinction between analysis patterns and defect mechanisms in Theoretical Software Diagnostics. [142] Repeated retries, missing messages, or a long interval identify structures in the collected artifacts. They do not uniquely identify why those structures occurred. A checkpoint is a selected snapshot of training state, not necessarily a complete process memory dump. By combining checkpoints with runtime artifacts and traces, we can construct and test a more complete problem narrative while retaining what each artifact omits.

PART VI

Foundation models, generative AI, and agents

31. Foundation-model evaluation: scaling, benchmark suites, contamination, and open-ended outputs

Transformers and large-scale language modeling changed the kinds of behavior available for analysis. Attention-based architectures supported parallel training, and large autoregressive models demonstrated few-shot behavior through prompting. [98, 99] The input to an application now includes instructions, examples, conversation history, and sometimes retrieved material. We therefore need to identify the complete input configuration before comparing two outputs. The model name alone does not specify the experiment.

Scaling studies related language-model loss to parameters, data, and computation. Compute-optimal training work examined their allocation. [100, 101] These relationships help compare and plan training runs. They do not establish the behavior of every downstream application. A lower aggregate loss may coexist with a factual error, a failing language subset, or an unsuitable response to a particular instruction. Diagnostic analysis therefore needs measures at the level of the intended use.

The foundation-model formulation emphasized the use of common upstream models in many downstream systems. [102] This creates shared capabilities and shared sources of failure. HELM evaluated multiple scenarios and measures, while BIG-bench assembled a broad collection of tasks. [103, 104] Such suites expand the observable behavior, but comparisons remain dependent on prompts, scoring, access conditions, and the exact model version. These details belong with the reported results.

Benchmark contamination occurs when evaluation material, answers, or related versions enter training or influence repeated optimization. It weakens the distinction between construction and evaluation discussed earlier. Open-ended outputs add another difficulty because one reference answer may not represent every acceptable response. We therefore combine refreshed tasks, explicit rubrics, human review, automated judges, and adversarial cases according to the question being examined. The evaluation procedure needs a version and provenance just as the model does.

Generative evaluation also has a history before foundation language models. Heusel and colleagues introduced Fréchet Inception Distance in their 2017 work on GAN training. [133] It compares summaries of generated and reference images in a learned feature space. This provides evidence about distributional similarity, but it does not establish the correctness of every image or its compliance with a prompt. The

reference set, feature extractor, and sample size are part of the measurement. This earlier example helps explain why one aggregate generation score cannot replace inspection of diversity, individual failures, and task requirements.

32. Hallucination, factuality, alignment, red teaming, and model-as-judge evaluation

Generated text may be linguistically plausible without being supported by evidence. TruthfulQA examined whether models reproduced common falsehoods, and FActScore evaluated factual precision through smaller claims within generated text. [105, 106] We should preserve the question, reference source, and time of evaluation. An unsupported claim may be false, unverifiable from the chosen source, or outside that source's scope. These cases require different interpretations even when an automated evaluator groups them together.

Retrieval-augmented generation adds an external evidence store to generation. [107] We can examine query construction, retrieval, ranking, context assembly, generation, and citation separately. If an answer is wrong, the evidence may have been absent, missed, truncated, or ignored. A final accuracy score does not distinguish these possibilities. Intermediate artifacts allow us to compare what was available with what the generator actually received and used.

Learning from human preferences and instruction-following training added further models and datasets to the process. [108, 109] The resulting behavior depends on the selected comparisons, reward signals, and optimization procedure. Red teaming with language models increased the scale of searching for undesirable outputs. [110] Such testing still depends on the diversity of attempted cases and the sensitivity of the evaluator. The search record helps us state which failure conditions were actually examined.

Model-based judges reduce some of the work involved in rating open-ended outputs. Studies using MT-Bench and Chatbot Arena also identified position, verbosity, and self-enhancement biases. [111] We therefore treat the judge as a diagnostic instrument with its own version, prompt, rubric, and validation examples. Human disagreements should remain available for analysis. NIST's generative-AI profile places these questions alongside confabulation, provenance, information integrity, and human–AI configuration in a broader lifecycle view. [112]

A generated explanation is a further output to evaluate. Turpin and colleagues showed that chain-of-thought explanations can omit factors that influence the answer, including deliberately introduced prompt biases. [141] We should therefore distinguish an explanation presented by a model from a verified account of its internal computation. It may be useful evidence of behavior, but its causal faithfulness

requires additional tests. Tool execution records and observed state changes supply a different kind of evidence and should not be replaced by the model's account of what it did.

33. Multimodal and agentic observability: prompts, retrieval, memory, tools, trajectories, and side effects

An agent connects generated output with actions and persistent state. ReAct combined reasoning-like text, actions, and observations; Reflexion used verbal feedback across attempts. [113, 114] We can analyze the resulting trajectory: the instruction, selected tools, arguments, observations, retries, memory changes, and termination. A correct final answer does not establish that the preceding actions were appropriate. The trajectory and its external effects are therefore separate diagnostic objects.

SWE-bench evaluated systems on real repository issues rather than isolated code-generation examples. [115] Such evaluation requires an environment, tool configuration, tests, and a way to check the resulting changes. Multimodal systems introduce additional transformations before a model call. Image cropping, audio segmentation, temporal alignment, and modality selection may remove relevant evidence. To localize a failure, we need to compare the original material with the representation delivered to the model.

AgentOps proposed observability for agent components, traces, costs, and failures. [116] MASEval treated the complete multi-agent implementation as the unit of evaluation, including its harness and context engineering. [117] This distinction matters when the same model behaves differently under different coordination, memory, or recovery procedures. A comparison of model names may conceal the component that actually produced the difference.

OpenTelemetry's generative-AI work extends distributed tracing to model and agent activities. [118] The GenAI conventions were introduced as experimental in May 2024 with semantic-conventions v1.26.0. They moved to a dedicated repository in June 2026 with v1.42.0 and remain in Development as of September 2026. [119] Common event meanings help compare calls across components, but we need to record the relevant conventions and instrumentation versions. We can collect external events, identifiers, costs, and outcomes while applying suitable controls to sensitive content. Internal reasoning may be unavailable and should not be assumed to have been captured merely because a trace contains explanatory text.

Indirect prompt injection illustrates why the source of an instruction matters. Greshake and colleagues examined attacks placed in material retrieved by an LLM application. [140] The application may incorrectly treat external data as authority to change its behavior. For diagnosis, we need to preserve the distinction among the user's request, retrieved content, model output, tool request, and executed operation. We

can then investigate where the trust boundary failed. A fluent final response may omit the unauthorized activity entirely.

Agent tracing also connects with our software narratological viewpoint. [142] Several tool calls may belong to one logical activity even when they execute on different threads or services. Conversely, one execution thread may serve several activities. Correlation identifiers let us reconstruct selected threads of activity, while event times and causal links help distinguish ordering from concurrency. We should record both a requested operation and its observed completion, since an intent message does not establish that a side effect occurred.

PART VII

Case studies, recurring principles, and future directions

34. Diagnostic lessons from consequential AI and ML failures

Google Flu Trends: proxy drift and platform dynamics

Google Flu Trends illustrates the diagnostic difficulty of using an evolving proxy. Lazer and colleagues examined errors in relation to changing search behavior, algorithm dynamics, and comparison with established surveillance. [120] We should retain the relationship among proxy, platform, population, and target when interpreting a result. A large quantity of search data does not remove the need to investigate how that relationship changes over time.

Tay: adversarial social deployment

Microsoft's Tay chatbot was taken offline after coordinated users elicited offensive behavior. Microsoft's account described earlier testing and filtering and acknowledged an attack that had not been anticipated. [121] This case expands the evaluation context beyond a set of ordinary conversations. The deployed system also encounters adaptive users and coordinated abuse. Monitoring, containment, and subsequent investigation become parts of the system whose behavior we need to examine.

The Tempe automated-driving crash: system boundaries and safety culture

The NTSB investigation of the 2018 fatal Tempe crash involving an Uber ATG developmental automated-driving system examined perception, response, operator supervision, risk assessment, and safety culture. [122] The report supports analysis of the connected testing system rather than attribution to one classification result. Component records become meaningful in relation to control decisions and human responsibilities. We need these connections to distinguish an immediate event from the conditions that allowed it to become consequential.

COVID-19 prediction models: urgency without adequate validation

The initial 2020 systematic review of COVID-19 diagnostic and prognostic prediction models rated all included models at high risk of bias and found poor reporting. [123] The problems included limited samples, overfitting, and inadequate validation. For this history, the case concerns the evidence supporting an ML model, rather than the clinical diagnosis performed with it. Elaborate computation cannot repair an evaluation design that does not support the claimed use.

Mata v. Avianca: generated authority without verification

In Mata v. Avianca, attorneys submitted nonexistent cases and false quotations generated with AI and failed to verify them before the court. The sanctions decision records the resulting failure. [124] We can distinguish several stages: generation, acceptance, incorporation into a document, and verification. Text formatted as a citation was treated as if it already established a source. A diagnostic correction needs a traceable connection from each claim to an authentic authority and a completed review.

The GPT-4o sycophancy rollback: preference signals and behavioral regression

OpenAI's account of its 2025 GPT-4o rollback described an update that had become excessively agreeable and a release process that had not adequately weighted the relevant evidence. [125, 126] The case shows how preference signals and product feedback can accompany a broad behavioral regression despite passing other evaluations. We need to compare qualitative reports, targeted tests, rollout observations, and the changes introduced by the release. A rollback contains the incident, while further analysis is required to explain its mechanism.

Cross-case synthesis

These cases involve different systems, but we can identify recurring diagnostic relationships. A proxy changes, an adversary supplies unexpected input, a review stage fails, or an aggregate measure omits important behavior. We use these similarities to guide analysis without claiming that the incidents share one cause. In each case, expanding the collected evidence beyond the initially visible output helps reconstruct the surrounding process.

35. Recurring principles and future directions

Make every claim conditional and scoped

We should retain the conditions of a diagnostic claim. Accuracy refers to a population and a decision rule. Robustness refers to a threat model. An attribution refers to a baseline and a method. A trace refers to selected recorded events. If these conditions are removed, an observation may acquire a broader meaning than the evidence supports. Keeping them with the result allows later analysts to check whether the claim applies to their problem.

Align identifiers and time across system levels

Faults and their effects may cross several levels of a system. A label affects training, training affects a representation, and a deployed decision changes the data collected next. We need identifiers and times

that connect these stages. The purpose is to reconstruct relevant relationships, not to collect every available personal detail. Diagnostic design should identify which observations are necessary and how their provenance can be preserved.

Test the diagnostic instruments

Diagnostic instruments require their own validation. A saliency map, drift detector, sparse feature, automated judge, or fault-diagnosis model can fail in a characteristic way. Sanity checks and controlled interventions help us examine those failures. [71, 77, 96, 111] We should retain known-fault examples and compare the instrument's response with the claim it is intended to support. Adding a second model to an analysis does not automatically provide independent evidence.

Connect observability to control

Observation needs a path to action. An uncertain result may require abstention or further review; a failing subgroup may require a deployment change; an agent's tool request may require a permission check. Frameworks and management systems connect these activities with responsibilities. [3, 91, 92, 112] We need to inspect whether those responsibilities can actually be exercised and whether the resulting intervention changed the observed problem.

Treat trajectories and side effects as evidence

Agentic and multimodal systems require evidence about complete activities and their effects. Evaluation benefits from environments that can be reset and from explicit records of permissions, tool state, memory, and termination. System-level evaluation and telemetry conventions support this direction. [117, 119] We should distinguish the performance of the model from the behavior of its harness and external components. Long activities also require checking what happened between the initial request and the final output.

Preserve disagreement and plural evidence

We should preserve competing explanations and the observations used to reject or revise them. Raw artifacts, failed tests, human overrides, and uncertainty help a later analyst reconstruct the reasoning. In our pattern-oriented terminology, the analysis itself can be examined as a narrative. [142] Its sequence may show where evidence was missing or a hypothesis was accepted too early. This provides another way to improve diagnostic practice without treating one final explanation as an irreversible conclusion.

Conclusion

We have followed several historical threads of ML and AI diagnostics. Residual analysis and process control made discrepancies available for investigation. Symbolic systems recorded rules and justifications. Learning systems added weights, gradients, validation results, and checkpoints. Production systems required data lineage, service traces, and records of feedback. Generative models and agents added further questions about output support, evaluation procedures, actions, and their consequences.

Each development extended the available evidence without removing the need to interpret it. More data may introduce new collection problems. A more accurate model may be harder to inspect directly. Additional telemetry may still omit the event needed to distinguish two hypotheses. We therefore continue to use controlled comparisons: independent evaluation data, separate population slices, numerical reference computations, interventions, and verified external outcomes. Their usefulness depends on the question and on how the artifacts were obtained.

The diagnostic object has also expanded. A classifier or rule base remains a useful object of analysis, but a deployed AI system includes transformations, prompts, retrieval, tools, policies, interfaces, and people. Many failures occur in the relationships among these components. We cannot expect a parameter file to explain a missing retrieval result, an incorrect permission decision, or an unexamined human approval. We need artifacts that preserve those relationships.

From our pattern-oriented viewpoint, this history describes an expanding collection of observable structures and methods for their interpretation. We first identify what the artifacts show, then consider mechanisms consistent with those observations, and test possible changes. A repeated structure can guide the investigation, but it does not replace that investigation. Observability supplies the material, diagnostics organizes and interprets it, and debugging checks proposed interventions. Preserving the evidence and its conditions allows this process to be reviewed and continued when new information becomes available.

Appendix A. Selected chronology

Table 3. Selected developments; dates mark documented milestones rather than single origins.

DATE	DEVELOPMENT	DIAGNOSTIC CONSEQUENCE
1931	Shewhart's statistical quality control	Control charts join measured variation to detection rules and operational investigation.
1936	Fisher analyzes Anderson's iris measurements	A small measured dataset becomes an enduring demonstration of classification and statistical separation.
1943	McCulloch–Pitts neural model	Idealized neurons connect formal logic, networks, and inspectable computational structure.
1948	Wiener's Cybernetics	Feedback, error, communication, and control provide a systems vocabulary for adaptive behavior.
1950	Turing on machine intelligence	Observable behavior and learning machines frame evaluation without requiring privileged access to inner essence.
1950	Brier probabilistic score	Probability forecasts receive a proper score sensitive to both confidence and outcome.
1956	Logic Theory Machine	Heuristic search and proof construction produce replayable symbolic reasoning traces.
1958	Rosenblatt's perceptron	Classification and adjustable connection strengths become central objects for investigating learned behavior.
1959	Samuel's checkers learning program	Self-play and evaluation functions make performance improvement an experimental object.
1960	ADALINE and the Widrow–Hoff rule	Continuous error correction supports adaptive circuits and convergence diagnostics.
1964–1970	Automatic differentiation foundations	Program structure is traversed to calculate derivatives and later support training telemetry.
1969	Minsky and Papert's Perceptrons	Formal analysis exposes representational limitations of a defined model class.
1970	Ridge regression	Deliberate shrinkage stabilizes estimation and makes the bias–variance tradeoff operational.
1974	Cross-validatory model assessment	Withheld partitions become a practical instrument for estimating predictive generalization.
1979	Truth-maintenance systems	Beliefs retain justifications and can be revised when supporting assumptions change.
1983	Bootstrap error estimation	Resampling supports empirical comparison of prediction-error estimators.
1984	CART	Tree paths, pruning, and validation join prediction to directly inspectable decision structure.
1984	MYCIN experiments published	Explanation facilities expose rules and intermediate conclusions in an expert system.

DATE	DEVELOPMENT	DIAGNOSTIC CONSEQUENCE
1986	Back-propagation of error	Efficient credit assignment makes gradients, activations, and learning curves central diagnostic signals.
1987	UCI Machine Learning Repository	Shared datasets lower the cost of repeatable algorithm comparison.
1987	Diagnosis from first principles	System description, observations, and candidate faulty components are joined in formal diagnosis.
1987	Silhouette analysis for clustering [143]	Cluster separation becomes an inspectable property under a selected dissimilarity measure.
1991–1994	Vanishing-gradient analysis	Long-range credit assignment failure is connected to recurrent training dynamics.
1992	Neural bias–variance analysis	Prediction error is decomposed into approximation and sample sensitivity for learning systems.
1992	Multiple-sequence convergence diagnostics [127]	Comparing simulation chains helps assess computational convergence separately from model adequacy.
1995	Empirical cross-validation study	Practical fold and resampling choices receive comparative evaluation.
1996	Lasso	Sparsity and shrinkage make feature selection part of regularized model fitting.
1996	Posterior predictive model assessment [128]	Observed discrepancies are compared with replicated data under a fitted Bayesian model.
1998	MNIST and gradient-based document recognition	A stable benchmark, architecture, and training pipeline support reproducible comparison.
2001	Two cultures, random forests, and boosting	Predictive modeling, ensembles, and variable-importance questions reshape evaluation and explanation.
2006	ROC analysis synthesis	Threshold tradeoffs become a routine visualization for classifier discrimination.
2009	ImageNet	Large-scale hierarchical labels establish a new benchmark regime for visual recognition.
2010	Initialization analysis	Signal and gradient scale are tied to activation choice and trainability.
2011	Doubly robust policy evaluation [130]	Logged actions and rewards support evaluation of a different policy under explicit assumptions.
2012	AlexNet	Benchmark progress visibly depends on data, accelerators, architecture, augmentation, and optimization together.
2012	Journal treatment of data leakage	Unavailable information is analyzed as a source of deceptively strong evaluation; the conference paper appeared in 2011.
2013–2014	Adversarial examples	Search-generated perturbations reveal failure beyond naturally sampled test sets.
2014	Google Flu critique	Proxy instability, platform dynamics, and comparison with established surveillance become central diagnostic lessons.
2014	Concept-drift survey	Temporal change in streaming relationships receives a systematic adaptation vocabulary.

DATE	DEVELOPMENT	DIAGNOSTIC CONSEQUENCE
2014	Generative adversarial networks [132]	Coupled generator and discriminator training requires joint analysis of dynamics and output diversity.
2015	Hidden technical debt in ML systems	The diagnostic boundary expands from model code to data, feedback, consumers, and operations.
2015	Batch normalization and Adam	Training-state instrumentation adapts to new normalization and optimizer dynamics.
2015	Precision–recall analysis for imbalance	Evaluation focuses more directly on positive-class retrieval under skewed prevalence.
2016	LIME and the Tay incident	Local surrogate explanation rises as adversarial social deployment exposes conversational-system risk.
2016	Concrete problems in AI safety [129]	Reward hacking, side effects, supervision, exploration, and shift become distinct diagnostic problems.
2016–2017	Membership inference and BadNets [136, 137]	Privacy leakage and trigger-dependent failures show why ordinary validation accuracy is insufficient.
2017	ML Test Score and TFX	Production readiness becomes a portfolio of automated checks embedded in a continuous pipeline.
2017	Transformers	Prompt-conditioned sequence models create new internal and behavioral diagnostic surfaces.
2017	SHAP, integrated gradients, and Grad-CAM	Feature attribution receives unified, axiomatic, and localization-oriented methods.
2017	DeepXplore	Differential testing and neuron coverage adapt software-testing ideas to neural networks.
2017	Reluplex verification [135]	Properties of ReLU networks can be proved or refuted within stated constraints.
2017	Fréchet Inception Distance [133]	Generated images are compared with reference images through learned feature statistics.
2017–2018	Mixed precision and quantized inference [138, 139]	Numerical formats and model conversion require explicit diagnostic comparisons.
2018	Gender Shades	Intersectional subgroup evaluation demonstrates how aggregate scores conceal disparate error.
2018	TCAV and saliency sanity checks	Concept-level analysis and tests of explanation dependence raise validation standards.
2018	Fatal Tempe automated-driving crash	Investigation exposes system-wide interactions among perception, controls, oversight, and safety culture.
2018–2019	Experiment tracking and data validation	Run provenance, schemas, anomalies, and training–serving skew become platform concerns.
2019	Model cards and predictive uncertainty under shift	Documentation and stress-tested uncertainty broaden evidence beyond accuracy.

DATE	DEVELOPMENT	DIAGNOSTIC CONSEQUENCE
2019	OpenTelemetry project begins	Vendor-neutral traces, metrics, and logs provide a substrate later extended to AI workloads.
2020	GPT-3, RAG, CheckList, and TracIn	Few-shot generation, retrieval pipelines, behavioral tests, and training-data influence expand the diagnostic toolkit.
2021	Foundation-model report and dataset accountability	Shared upstream models, datasheets, label-error audits, and data cascades shift attention to inherited risk.
2021	Statistical uncertainty in deep RL evaluation [131]	Intervals and performance profiles expose limitations of comparisons based on a few runs.
2022	TruthfulQA, InstructGPT, and Chinchilla	Truthfulness tests, preference training, and compute–data balance reshape evaluation.
2022	Underspecification study published in JMLR [134]	Models with similar held-out scores can exhibit different deployment behavior; preprint appeared in 2020.
2023	HELM, BIG-bench, ReAct, FActScore, and LLM judges	Multimetric suites, tool trajectories, atomic factuality, and automated grading address open-ended systems; HELM and BIG-bench preprints appeared in 2022.
2023	Mata v. Avianca sanctions	Fabricated citations show why generated claims need source-level verification and accountable review.
2023–2024	Sparse-feature interpretability	Dictionary learning and large-scale feature maps pursue more legible units inside language models.
2023	Indirect prompt injection and explanation faithfulness [140, 141]	Retrieved instructions and generated explanations become separate targets for security and causal checks.
2024	NIST generative-AI profile, EU AI Act, and AgentOps	Lifecycle risk, legal duties, and agent tracing become institutional and technical observability requirements.
2024	Experimental GenAI semantic conventions [119]	Version 1.26.0 introduces common telemetry definitions for GenAI clients in May; the conventions remain under development.
2025	Aegis and circuit tracing	Production training diagnosis and prompt-specific attribution graphs advance automated and mechanism-level analysis.
2025	GPT-4o sycophancy rollback	A broad behavioral regression exposes limits of aggregate release evaluation and preference signals.
2026	GenAI conventions move to a dedicated repository [119]	Version 1.42.0 records the June move; the GenAI repository retains Development status as of September 2026.
2026	MASEval [117]	System-level multi-agent evaluation treats the harness, context, tools, and trajectories as diagnostic objects.

Appendix B. Glossary

Abstention. A policy under which a model declines to predict or act when evidence is insufficient or risk is too high.

Activation. The value produced by a unit, feature, or layer for a particular input during a forward computation.

Agent trajectory. The ordered record of an agent's instructions, states, actions, tool calls, observations, retries, and termination.

Algorithmic audit. A structured examination of an AI system's design, evidence, outcomes, controls, and accountability.

Analysis pattern. A reusable way to recognize or interpret a recurring structure in diagnostic artifacts within a stated context.

Attribution. An estimate of how input features, training examples, components, or internal representations relate to an output.

Automatic differentiation. Programmatic computation of derivatives by applying the chain rule to an executed computational graph.

Backdoor. An intentionally introduced behavior activated by selected trigger conditions while ordinary behavior may remain apparently correct.

Backpropagation. Reverse propagation of output error or gradients through a network to compute parameter updates.

Baseline. A reference input, distribution, period, model, or policy against which a diagnostic comparison is made.

Benchmark. A standardized dataset, task suite, protocol, and metric used to compare systems.

Benchmark contamination. Direct or indirect exposure of evaluation material to training, tuning, prompting, or repeated development.

Bias–variance tradeoff. The relation between systematic approximation error and sensitivity to variation in the training sample.

Brier score. The mean squared error between predicted probabilities and binary outcomes.

Calibration. Agreement between predicted probabilities and observed frequencies for comparable predictions.

Canary deployment. Limited release to a small traffic share or population so behavior can be checked before wider rollout.

Causal intervention. A deliberate change used to test whether a component or variable influences an outcome under stated assumptions.

Chain-of-thought explanation. Generated intermediate explanatory text whose relationship to the computation producing an answer requires separate validation.

Champion–challenger. A comparison in which a candidate model is evaluated against the currently deployed or approved model.

Checkpoint. A saved selection of training state, which may include weights, optimizer state, and progress; it need not contain complete process memory.

Concept activation vector. A direction in representation space derived from examples of a human-defined concept and used to test sensitivity.

Concept drift. Change over time in the relationship between inputs and the target or decision-relevant outcome.

Conformal prediction. A method for producing prediction sets or intervals with coverage guarantees under specified assumptions; ordinary marginal coverage is not a guarantee for every subgroup or individual input.

Confusion matrix. A table of predicted and actual classes that separates true and false positive and negative outcomes.

Convergence diagnostic. A check of whether an iterative computational procedure has sufficiently stabilized or explored its target for the intended inference.

Counterfactual explanation. A proposed change to an input or state that would change a model decision.

Covariate shift. Change in the input distribution while the conditional relationship between target and input is assumed stable.

Cross-validation. Repeated fitting and evaluation across partitions of a dataset to estimate generalization or guide model choice.

Data cascade. A compounding sequence in which early data problems create delayed downstream model and organizational effects.

Data drift. A monitored change in data values, distributions, relationships, or collection process over time.

Data leakage. Use of information during training or evaluation that would not be legitimately available at decision time.

Dataset shift. A difference between training, evaluation, and deployment data-generating conditions.

Decision threshold. The score or probability boundary used to map a model output to an action or class.

Deep ensemble. A collection of independently trained neural networks whose combined predictions can improve accuracy and uncertainty estimates.

Defect mechanism. A process or interaction that explains how a defect produces observed symptoms under particular conditions.

Delayed label. An outcome that becomes observable only after a material interval following prediction or action.

Diagnostic hypothesis. A testable proposed explanation for an observed discrepancy or failure.

Disaggregated evaluation. Reporting performance separately for meaningful classes, contexts, or population groups.

Distribution shift. A broad term for change between the distributions encountered in training, evaluation, or use.

Drift detector. A statistical or learned mechanism that flags change relative to a reference period or process.

Early stopping. Ending training according to validation behavior or another criterion before the optimization budget is exhausted.

Embedding. A learned vector representation of an item, token, example, or state.

Error analysis. Structured examination of failed cases to identify patterns, slices, mechanisms, or repair opportunities.

Evaluation harness. The code, environment, prompts, tools, data, and scoring logic that execute an evaluation.

Explainability. The production and communication of reasons, evidence, or representations intended to make system behavior understandable.

Faithfulness. The degree to which an explanation or trace accurately reflects the behavior or mechanism it claims to describe.

Feature. A measured, engineered, or learned variable made available to a model.

Feature attribution. Assignment of output credit or sensitivity to input features under a defined method and reference.

Feature store. A managed system for defining, computing, serving, and versioning features across training and inference.

Feedback loop. A dynamic in which system outputs change future inputs, behavior, labels, or training data.

Formal verification. Checking whether an encoded system satisfies a stated property under explicit assumptions and constraints.

Foundation model. A broadly trained model adapted to many downstream tasks and systems.

Fréchet Inception Distance. A comparison of generated and reference image distributions using Gaussian summaries in a learned feature space.

Fuzzing. Automated generation or mutation of inputs to search for crashes, anomalies, or behavioral failures.

Generalization. Performance on relevant observations or conditions not directly used to fit the model.

Generative adversarial network. A generative model trained through interaction between a generator and a discriminator.

Gradient. The derivative of an objective with respect to parameters, inputs, or intermediate values.

Gradient check. A comparison between computed gradients and numerical finite-difference estimates on a controlled problem.

Hallucination or confabulation. Generated content presented without adequate support from the model's evidence or external sources.

Holdout set. Data reserved from fitting and tuning for an independent evaluation under a defined process.

Human-in-the-loop. A workflow in which people review, correct, approve, or otherwise influence model operation.

Incident. An event in which an AI or ML system causes, risks, or reveals unacceptable behavior or operational failure.

Indirect prompt injection. An attack that places instructions in externally supplied content that an application later processes as data.

Influence function. An approximation of how changing the weight of a training example affects fitted parameters or predictions.

Integrated gradients. An attribution method that integrates input gradients along a path from a baseline to the observed input.

Interpretability. The extent to which a person can understand a model's structure, representation, or decision process.

Label error. A target annotation that is wrong, inconsistent, outdated, or incompatible with the intended task definition.

Label shift. Change in class proportions under an assumption that class-conditional input distributions remain stable.

Learning curve. A plot of training or evaluation performance against data quantity, iterations, time, or another learning axis.

Lineage. Traceable relationships among data, code, configuration, runs, models, deployments, and outcomes.

LLM-as-a-judge. Use of a language model to score, compare, classify, or critique outputs from another system.

Local explanation. An explanation scoped to one prediction or a small neighborhood of inputs.

Loss. An objective quantity that measures model error or penalty and is usually minimized during training.

Mechanistic interpretability. Analysis aimed at identifying internal computational structures and causal mechanisms that produce behavior.

Membership inference. Attempting to determine whether a particular record was included in a model's training data.

Metamorphic test. A test of a relation expected to hold across transformed inputs when a complete output oracle is unavailable.

Metric. A defined numerical summary used to compare behavior, performance, cost, risk, or quality.

Mixed precision. Use of more than one numerical precision within training or inference, with associated choices about accumulation and scaling.

Mode collapse. A generative-model failure in which output diversity is substantially reduced and parts of the target distribution are omitted.

Model assertion. An executable expectation about model inputs, predictions, relationships, or outcomes used for monitoring or testing.

Model card. Structured documentation of a model's intended uses, limitations, evaluation, and relevant ethical considerations.

Model registry. A managed catalogue of model versions, metadata, approval stages, and deployment status.

Monitoring. Repeated or continuous observation intended to detect change, degradation, risk, or control failure.

Observability. The designed ability to obtain evidence about otherwise hidden system state and behavior.

Off-policy evaluation. Estimation of a target policy's performance from data collected under another policy, subject to support and modeling assumptions.

Out-of-distribution detection. Estimation of whether an input differs materially from the distribution represented by training or reference data.

Partial dependence. An average modeled response to selected features after marginalizing over other observed features.

Post-training. Training after pretraining, including supervised instruction tuning, preference optimization, or safety adaptation.

Posterior predictive check. Comparison of observed data with data replicated from a fitted Bayesian model using selected discrepancies.

Precision–recall curve. A plot of precision against recall across thresholds, especially useful for imbalanced positive classes.

Prediction trace. A record linking a request to features, model and policy versions, outputs, timing, and downstream handling.

Prompt injection. Input designed to redirect a model or agent away from intended instructions, permissions, or data boundaries.

Provenance. Evidence about the origin, transformation, ownership, and history of data or artifacts.

Quantization. Representation of values with a restricted set of numerical levels, requiring checks of the resulting computational and behavioral differences.

Red teaming. Adversarial exploration intended to discover harmful, insecure, or policy-violating behavior before or during deployment.

Regularization. A constraint, penalty, data transformation, or training effect that discourages overly complex or unstable solutions.

Residual. The difference between an observed value and a model prediction or fitted expectation.

Retrieval-augmented generation. Generation conditioned on evidence retrieved from an external corpus or system at inference time.

Reward hacking. Behavior that improves a specified reward while departing from the intended task or its constraints.

Rollback. Restoration of a prior model, data, configuration, or service state after a problematic release.

Saliency map. A visual attribution of output sensitivity or relevance to parts of an input, commonly an image.

Service-level objective. A target for a service indicator such as availability, latency, or error rate over a defined interval.

Shadow deployment. Running a candidate system on live traffic without allowing its outputs to control user-visible action.

Silhouette. A measure comparing an observation’s relation to its assigned cluster with its relation to other clusters under a chosen dissimilarity.

Slice. A defined subset of examples used to inspect heterogeneous behavior.

Sycophancy. A tendency to agree with, flatter, or reinforce a user at the expense of truthfulness, balance, or appropriate challenge.

Test oracle. A source or rule that determines the expected result of a test.

Threat model. An explicit description of attacker goals, knowledge, access, constraints, and protected assets.

Training–serving skew. A mismatch between data or computation used during training and that used during live inference.

Uncertainty quantification. Methods for representing limits, variability, or confidence in predictions and model behavior.

Underspecification. Insufficient constraints in a training and evaluation procedure to distinguish models that behave differently in relevant deployment conditions.

Validation gate. A policy checkpoint that a candidate artifact or release must pass before progressing.

Appendix C. Evidence taxonomy

Table 4. Evidence classes and their scope. Combining independent observations can strengthen an explanation when their provenance and assumptions remain visible.

EVIDENCE CLASS	TYPICAL EXAMPLES	CHARACTERISTIC STRENGTH	CHARACTERISTIC LIMITATION
Statistical fit	Residuals, likelihood, goodness of fit, influence, posterior or predictive checks	Model–data mismatch under stated assumptions	Passing a check is not proof that the model is correct
Generalization	Holdout, cross-validation, temporal or grouped evaluation	Expected performance on a defined target population	Leakage, repeated tuning, and distribution mismatch can invalidate estimates
Metric and decision	Confusion matrices, curves, calibration, utility and cost analysis	Discrimination and operational tradeoffs	A metric encodes prevalence, weighting, threshold, and consequence choices
Data quality	Schemas, missingness, ranges, duplicates, labels, provenance, collection records	Whether training and serving evidence is usable and comparable	Semantic change may pass structural checks
Training dynamics	Losses, gradients, activations, optimizer state, checkpoints, resource events	Whether learning proceeds numerically and behaviorally as intended	Telemetry can faithfully describe the wrong objective or graph
Behavioral testing	Capability tests, metamorphic relations, fuzzing, adversarial and scenario suites	Whether intended invariants and constraints hold	Coverage is bounded by the generated cases and oracle
Interpretation	Coefficients, rules, attributions, concepts, examples, counterfactuals	What evidence or representation relates to a prediction	Plausibility, stability, faithfulness, and causality are different properties
Mechanism	Ablation, activation patching, sparse features, circuits, causal tracing	Internal computations supporting a behavior	Methods approximate and select from distributed computation
Robustness and security	Perturbations, attacks, membership tests, triggers, shift, fault injection, permissions tests	Behavior beyond nominal samples and under adversarial pressure	Results apply only to the stated threat and failure models
Fairness and impact	Disaggregated errors, outcomes, user research, appeals, impact assessments	Who benefits, fails, or bears risk	Group definitions and fairness criteria are contextual and normative
Provenance and reproducibility	Data snapshots, code, environments, run records, lineage, approvals	Whether a result or release can be reconstructed and compared	Uncaptured dependencies create false reproducibility
Production telemetry	Logs, metrics, traces, service objectives, prediction records, incidents	Operational state and request-level localization	Sampling, redaction, and schema gaps can remove decisive evidence

EVIDENCE CLASS	TYPICAL EXAMPLES	CHARACTERISTIC STRENGTH	CHARACTERISTIC LIMITATION
Human and organizational	Overrides, review records, staffing, incentives, policies, post-incident analyses	How people and institutions control or amplify system behavior	Nominal oversight may lack time, authority, or independence
Agent trajectory	Prompts, plans, tool calls, observations, memory, costs, side effects, termination	How a stateful system reached an outcome	Sensitive content and hidden state constrain what can safely be recorded
External outcome	Delayed labels, real-world effects, complaints, safety and business indicators	Whether deployed behavior achieves acceptable consequences	Outcomes may be selectively observed and affected by the policy itself
Computational convergence	Simulation chains, convergence statistics, trace plots, numerical reference runs	Whether an inference or training procedure supplies adequate computational evidence	Convergence does not establish that the statistical model is appropriate
Formal properties	Encoded constraints, solver results, proofs, counterexamples	Whether a stated property holds for a specified model and input region	Unmodeled conditions, incorrect specifications, and inconclusive checks limit the claim
Policy and reward	Episodes, action probabilities, rewards, external state changes, off-policy estimates	How behavior relates to the objective and to the policy that collected observations	Missing support and an unsuitable reward can invalidate an apparently strong result
Generative distribution	Diversity checks, reference samples, feature statistics, individual output review	Which aspects of a target distribution are reproduced by a generator	Aggregate similarity does not guarantee correctness, coverage, or prompt compliance

References

[1] D. Sculley et al. “Hidden Technical Debt in Machine Learning Systems.” Advances in Neural Information Processing Systems 28 (2015): 2503–2511. Source

[2] Raymond Reiter. “A Theory of Diagnosis from First Principles.” Artificial Intelligence 32, no. 1 (1987): 57–95. doi:10.1016/0004-3702(87)90062-2. Source

[3] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, January 2023. doi:10.6028/NIST.AI.100-1. Source

[4] Leo Breiman. “Statistical Modeling: The Two Cultures.” Statistical Science 16, no. 3 (2001): 199–231. doi:10.1214/ss/1009213726. Source

[5] R. A. Fisher. “The Use of Multiple Measurements in Taxonomic Problems.” Annals of Eugenics 7, no. 2 (1936): 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x. Source

[6] Walter A. Shewhart. Economic Control of Quality of Manufactured Product. D. Van Nostrand, 1931. Source

[7] Norbert Wiener. Cybernetics: Or Control and Communication in the Animal and the Machine. The Technology Press and John Wiley & Sons; Hermann & Cie, 1948 (first edition); 2nd ed., MIT Press, 1961. Source

[8] Warren S. McCulloch and Walter Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity.” Bulletin of Mathematical Biophysics 5 (1943): 115–133. doi:10.1007/BF02478259. Source

[9] A. M. Turing. “Computing Machinery and Intelligence.” Mind 59, no. 236 (1950): 433–460. doi:10.1093/mind/LIX.236.433. Source

[10] Allen Newell and Herbert A. Simon. The Logic Theory Machine: A Complex Information Processing System. RAND P-868, 1956. doi:10.7249/P868. Source

[11] Bruce G. Buchanan and Edward H. Shortliffe, eds. Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley, 1984. Source

[12] Jon Doyle. “A Truth Maintenance System.” Artificial Intelligence 12, no. 3 (1979): 231–272. doi:10.1016/0004-3702(79)90008-0. Source

[13] Frank Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” Psychological Review 65, no. 6 (1958): 386–408. doi:10.1037/h0042519. Source

[14] Bernard Widrow and Marcian E. Hoff Jr. “Adaptive Switching Circuits.” IRE WESCON Convention Record, Part 4 (1960): 96–104. Source

[15] Arthur L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers.” IBM Journal of Research and Development 3, no. 3 (1959): 210–229. doi:10.1147/rd.33.0210. Source

[16] Marvin Minsky and Seymour Papert. Perceptrons: An Introduction to Computational Geometry. MIT Press, 1969; expanded edition, 1987. Source

[17] Mervyn Stone. “Cross-Validatory Choice and Assessment of Statistical Predictions.” Journal of the Royal Statistical Society: Series B (Methodological) 36, no. 2 (1974): 111–133 (discussion, 133–147). doi:10.1111/j.2517-6161.1974.tb00994.x. Source

[18] Bradley Efron. “Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation.” Journal of the American Statistical Association 78, no. 382 (1983): 316–331. doi:10.1080/01621459.1983.10477973. Source

[19] Ron Kohavi. “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.” Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), vol. 2, 1137–1143. Morgan Kaufmann, 1995. Source

[20] Stuart Geman, Elie Bienenstock, and René Doursat. “Neural Networks and the Bias/Variance Dilemma.” Neural Computation 4, no. 1 (1992): 1–58. doi:10.1162/neco.1992.4.1.1. Source

[21] Arthur E. Hoerl and Robert W. Kennard. “Ridge Regression: Biased Estimation for Nonorthogonal Problems.” Technometrics 12, no. 1 (1970): 55–67. doi:10.1080/00401706.1970.10488634. Source

[22] Robert Tibshirani. “Regression Shrinkage and Selection via the Lasso.” Journal of the Royal Statistical Society: Series B 58, no. 1 (1996): 267–288. doi:10.1111/j.2517-6161.1996.tb02080.x. Source

[23] Lutz Prechelt. “Early Stopping—but When?” In Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science 1524, 55–69. Springer, 1998. doi:10.1007/3-540-49430-8_3. Source

[24] UCI Machine Learning Repository. “About.” University of California, Irvine. Archive created in 1987 by David Aha. Accessed 1 August 2026. Source

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-Based Learning Applied to Document Recognition.” Proceedings of the IEEE 86, no. 11 (1998): 2278–2324. doi:10.1109/5.726791. Source

[26] Jia Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database.” IEEE Conference on Computer Vision and Pattern Recognition, 2009, 248–255. doi:10.1109/CVPR.2009.5206848. Source

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” Advances in Neural Information Processing Systems 25 (2012). Source

[28] Alex Wang et al. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding.” Proceedings of the 2018 EMNLP Workshop BlackboxNLP, 353–355. Source

[29] Tom Fawcett. “An Introduction to ROC Analysis.” Pattern Recognition Letters 27, no. 8 (2006): 861–874. doi:10.1016/j.patrec.2005.10.010. Source

[30] Takaya Saito and Marc Rehmsmeier. “The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets.” PLOS ONE 10, no. 3 (2015): e0118432. doi:10.1371/journal.pone.0118432. Source

[31] Glenn W. Brier. “Verification of Forecasts Expressed in Terms of Probability.” Monthly Weather Review 78, no. 1 (1950): 1–3. Source

[32] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. “On Calibration of Modern Neural Networks.” Proceedings of ICML, PMLR 70 (2017): 1321–1330. Source

[33] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. “Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles.” Advances in Neural Information Processing Systems 30 (2017). Source

[34] Yaniv Ovadia et al. “Can You Trust Your Model’s Uncertainty? Evaluating Predictive Uncertainty under Dataset Shift.” Advances in Neural Information Processing Systems 32 (2019). Source

[35] Glenn Shafer and Vladimir Vovk. “A Tutorial on Conformal Prediction.” Journal of Machine Learning Research 9 (2008): 371–421. Source

[36] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. “Leakage in Data Mining: Formulation, Detection, and Avoidance.” ACM Transactions on Knowledge Discovery from Data 6, no. 4 (2012): article 15. doi:10.1145/2382577.2382579. Source

[37] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. “Confident Learning: Estimating Uncertainty in Dataset Labels.” Journal of Artificial Intelligence Research 70 (2021): 1373–1411. doi:10.1613/jair.1.12125. Source

[38] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. “Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks.” NeurIPS 2021 Datasets and Benchmarks Track. Source

[39] Timnit Gebru et al. “Datasheets for Datasets.” Communications of the ACM 64, no. 12 (2021): 86–92. doi:10.1145/3458723. Source

[40] Nithya Sambasivan et al. “‘Everyone Wants to Do the Model Work, Not the Data Work’: Data Cascades in High-Stakes AI.” Proceedings of CHI 2021. doi:10.1145/3411764.3445518. Source

[41] R. E. Wengert. “A Simple Automatic Derivative Evaluation Program.” Communications of the ACM 7, no. 8 (1964): 463–464. doi:10.1145/355586.364791. Source

[42] Seppo Linnainmaa. “Algoritmin kumulatiivinen pyöristysvirhe yksittäisten pyöristysvirheiden Taylor-kehitelmänä” [The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors]. Master’s thesis (pro gradu, in Finnish), Department of Mathematics, University of Helsinki, 1970. 63 pp. Catalogue record; scanned copy

[43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning Representations by Back-Propagating Errors.” Nature 323 (1986): 533–536. doi:10.1038/323533a0. Source

[44] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Technische Universität München, 1991. Source

[45] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning Long-Term Dependencies with Gradient Descent Is Difficult.” IEEE Transactions on Neural Networks 5, no. 2 (1994): 157–166. doi:10.1109/72.279181. Source

[46] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the Difficulty of Training Recurrent Neural Networks.” Proceedings of ICML, PMLR 28 (2013): 1310–1318. Source

[47] Xavier Glorot and Yoshua Bengio. “Understanding the Difficulty of Training Deep Feedforward Neural Networks.” Proceedings of AISTATS, PMLR 9 (2010): 249–256. Source

[48] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” Proceedings of ICML, PMLR 37 (2015): 448–456. Source

[49] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” ICLR 2015. Source

[50] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. “Visualizing the Loss Landscape of Neural Nets.” Advances in Neural Information Processing Systems 31 (2018). Source

[51] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. Classification and Regression Trees. Wadsworth, 1984; Routledge reissue, 2017. doi:10.1201/9781315139470. Source

[52] Leo Breiman. “Random Forests.” Machine Learning 45 (2001): 5–32. doi:10.1023/A:1010933404324. Source

[53] Jerome H. Friedman. “Greedy Function Approximation: A Gradient Boosting Machine.” Annals of Statistics 29, no. 5 (2001): 1189–1232. doi:10.1214/aos/1013203451. Source

[54] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.” ICLR Workshop, 2014. Source

[55] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks.” ECCV 2014, 818–833. doi:10.1007/978-3-319-10590-1_53. Source

[56] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. “Feature Visualization.” Distill, 2017. doi:10.23915/distill.00007. Source

[57] Been Kim et al. “Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV).” Proceedings of ICML, PMLR 80 (2018): 2668–2677. Source

[58] Joelle Pineau et al. “Improving Reproducibility in Machine Learning Research: A Report from the NeurIPS 2019 Reproducibility Program.” Journal of Machine Learning Research 22, no. 164 (2021): 1–20. Source

[59] Peter Henderson et al. “Deep Reinforcement Learning That Matters.” Proceedings of AAAI 32, no. 1 (2018). doi:10.1609/aaai.v32i1.11694. Source

[60] Edward Raff. “A Step Toward Quantifying Independently Reproducible Machine Learning Research.” Advances in Neural Information Processing Systems 32 (2019). Source

[61] Matei Zaharia et al. “Accelerating the Machine Learning Lifecycle with MLflow.” IEEE Data Engineering Bulletin 41, no. 4 (2018): 39–45. Source

[62] Denis Baylor et al. “TFX: A TensorFlow-Based Production-Scale Machine Learning Platform.” Proceedings of KDD 2017, 1387–1395. doi:10.1145/3097983.3098021. Source

[63] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. “Data Validation for Machine Learning.” Proceedings of Machine Learning and Systems 1 (2019): 334–347. Source

[64] João Gama et al. “A Survey on Concept Drift Adaptation.” ACM Computing Surveys 46, no. 4 (2014): article 44. doi:10.1145/2523813. Source

[65] Zachary Lipton, Yu-Xiang Wang, and Alexander Smola. “Detecting and Correcting for Label Shift with Black Box Predictors.” Proceedings of ICML, PMLR 80 (2018): 3122–3130. Source

[66] Daniel Kang et al. “Model Assertions for Monitoring and Improving ML Models.” Proceedings of Machine Learning and Systems 2 (2020): 481–496. Source

[67] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “‘Why Should I Trust You?’: Explaining the Predictions of Any Classifier.” Proceedings of KDD 2016, 1135–1144. doi:10.1145/2939672.2939778. Source

[68] Scott M. Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions.” Advances in Neural Information Processing Systems 30 (2017). Source

[69] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic Attribution for Deep Networks.” Proceedings of ICML, PMLR 70 (2017): 3319–3328. Source

[70] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.” IEEE International Conference on Computer Vision, 2017, 618–626. Source

[71] Julius Adebayo et al. “Sanity Checks for Saliency Maps.” Advances in Neural Information Processing Systems 31 (2018). Source

[72] Julius Adebayo et al. “Debugging Tests for Model Explanations.” Advances in Neural Information Processing Systems 33 (2020). Source

[73] Chris Olah et al. “Zoom In: An Introduction to Circuits.” Distill 5, no. 3 (2020). doi:10.23915/distill.00024.001. Source

[74] Nelson Elhage et al. “A Mathematical Framework for Transformer Circuits.” Transformer Circuits Thread, 2021. Source

[75] Trenton Bricken et al. “Towards Monosemanticity: Decomposing Language Models with Dictionary Learning.” Transformer Circuits Thread, 4 October 2023. Source

[76] Adly Templeton et al. “Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet.” Transformer Circuits Thread, 21 May 2024. Source

[77] Emmanuel Ameisen et al. “Circuit Tracing: Revealing Computational Graphs in Language Models.” Transformer Circuits Thread, 27 March 2025. Source

[78] Pang Wei Koh and Percy Liang. “Understanding Black-Box Predictions via Influence Functions.” Proceedings of ICML, PMLR 70 (2017): 1885–1894. Source

[79] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. “Estimating Training Data Influence by Tracing Gradient Descent.” Advances in Neural Information Processing Systems 33 (2020): 19920–19930. Source

[80] Sandra Wachter, Brent Mittelstadt, and Chris Russell. “Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR.” Harvard Journal of Law & Technology 31, no. 2 (2018): 841–887. Source

[81] Christian Szegedy et al. “Intriguing Properties of Neural Networks.” ICLR 2014. Source

[82] Aleksander Madry et al. “Towards Deep Learning Models Resistant to Adversarial Attacks.” ICLR 2018. Source

[83] Andrew Ilyas et al. “Adversarial Examples Are Not Bugs, They Are Features.” Advances in Neural Information Processing Systems 32 (2019). Source

[84] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. “DeepXplore: Automated Whitebox Testing of Deep Learning Systems.” Proceedings of SOSP 2017, 1–18. doi:10.1145/3132747.3132785. Source

[85] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. “TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing.” Proceedings of ICML, PMLR 97 (2019): 4901–4911. Source

[86] Marco Tulio Ribeiro et al. “Beyond Accuracy: Behavioral Testing of NLP Models with CheckList.” Proceedings of ACL 2020, 4902–4912. Source

[87] Moritz Hardt, Eric Price, and Nati Srebro. “Equality of Opportunity in Supervised Learning.” Advances in Neural Information Processing Systems 29 (2016). Source

[88] Joy Buolamwini and Timnit Gebru. “Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification.” Proceedings of FAT* 2018, PMLR 81:77–91. Source

[89] Margaret Mitchell et al. “Model Cards for Model Reporting.” Proceedings of FAT* 2019, 220–229. doi:10.1145/3287560.3287596. Source

[90] Inioluwa Deborah Raji et al. “Closing the AI Accountability Gap: Defining an End-to-End Framework for Internal Algorithmic Auditing.” Proceedings of FAT* 2020, 33–44. doi:10.1145/3351095.3372873. Source

[91] International Organization for Standardization. ISO/IEC 42001:2023, Information technology—Artificial intelligence—Management system. Source

[92] European Union. Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, 12 July 2024. Source

[93] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. “The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction.” IEEE Big Data 2017, 1123–1132. doi:10.1109/BigData.2017.8258038. Source

[94] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy, eds. Site Reliability Engineering: How Google Runs Production Systems. O’Reilly Media, 2016. Source

[95] Ben Sigelman and Morgan McLean. “OpenTelemetry: The Merger of OpenCensus and OpenTracing.” Google Open Source Blog, 21 May 2019. Source

[96] Jianbo Dong et al. “Evolution of Aegis: Fault Diagnosis for AI Model Training Service in Production.” 22nd USENIX Symposium on Networked Systems Design and Implementation, 2025, 865–881. Source

[97] Dmitry Vostokov. Python Debugging for AI, Machine Learning, and Cloud Computing: A Pattern-Oriented Approach. Apress, 2024. doi:10.1007/978-1-4842-9745-2. Source

[98] Ashish Vaswani et al. “Attention Is All You Need.” Advances in Neural Information Processing Systems 30 (2017). Source

[99] Tom B. Brown et al. “Language Models Are Few-Shot Learners.” Advances in Neural Information Processing Systems 33 (2020): 1877–1901. Source

[100] Jared Kaplan et al. “Scaling Laws for Neural Language Models.” 2020. Source

[101] Jordan Hoffmann et al. “Training Compute-Optimal Large Language Models.” arXiv:2203.15556, 2022. Published as “An Empirical Analysis of Compute-Optimal Large Language Model Training,” Advances in Neural Information Processing Systems 35 (2022): 30016–30030. Preprint; Proceedings

[102] Rishi Bommasani et al. On the Opportunities and Risks of Foundation Models. Stanford Center for Research on Foundation Models, 2021. Source

[103] Percy Liang et al. “Holistic Evaluation of Language Models.” Transactions on Machine Learning Research, 2023. Source

[104] Aarohi Srivastava et al. “Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models.” Transactions on Machine Learning Research, 2023. Source

[105] Stephanie Lin, Jacob Hilton, and Owain Evans. “TruthfulQA: Measuring How Models Mimic Human Falsehoods.” Proceedings of ACL 2022, 3214–3252. Source

[106] Sewon Min et al. “FActScore: Fine-Grained Atomic Evaluation of Factual Precision in Long Form Text Generation.” Proceedings of EMNLP 2023, 12076–12100. Source

[107] Patrick Lewis et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.” Advances in Neural Information Processing Systems 33 (2020): 9459–9474. Source

[108] Paul F. Christiano et al. “Deep Reinforcement Learning from Human Preferences.” Advances in Neural Information Processing Systems 30 (2017). Source

[109] Long Ouyang et al. “Training Language Models to Follow Instructions with Human Feedback.” Advances in Neural Information Processing Systems 35 (2022): 27730–27744. Source

[110] Ethan Perez et al. “Red Teaming Language Models with Language Models.” Proceedings of EMNLP 2022, 3419–3448. Source

[111] Lianmin Zheng et al. “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.” Advances in Neural Information Processing Systems 36 (2023): 46595–46623. Source

[112] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1, July 2024. doi:10.6028/NIST.AI.600-1. Source

[113] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models.” ICLR 2023. Source

[114] Noah Shinn et al. “Reflexion: Language Agents with Verbal Reinforcement Learning.” Advances in Neural Information Processing Systems 36 (2023): 8634–8652. Source

[115] Carlos E. Jimenez et al. “SWE-bench: Can Language Models Resolve Real-World GitHub Issues?” ICLR 2024. Source

[116] Liming Dong, Qinghua Lu, and Liming Zhu. “AgentOps: Enabling Observability of LLM Agents.” arXiv:2411.05285, 2024. Source

[117] Cornelius Emde et al. “MASEval: Extending Multi-Agent Evaluation from Models to Systems.” Proceedings of ACL 2026, System Demonstrations, 345–356. doi:10.18653/v1/2026.acl-demo.34. Source

[118] James Newton-King. “Inside the LLM Call: GenAI Observability with OpenTelemetry.” OpenTelemetry Blog, 14 May 2026. Source

[119] OpenTelemetry Authors. OpenTelemetry Semantic Conventions, release v1.26.0 (21 May 2024), and release v1.42.0 (12 June 2026). OpenTelemetry GenAI Semantic Conventions (status: Development), accessed 23 September 2026. GenAI repository

[120] David Lazer, Ryan Kennedy, Gary King, and Alessandro Vespignani. “The Parable of Google Flu: Traps in Big Data Analysis.” Science 343, no. 6176 (2014): 1203–1205. doi:10.1126/science.1248506. Source

[121] Peter Lee. “Learning from Tay’s Introduction.” The Official Microsoft Blog, 25 March 2016. Source

[122] National Transportation Safety Board. Collision Between Vehicle Controlled by Developmental Automated Driving System and Pedestrian, Tempe, Arizona, March 18, 2018. Highway Accident Report NTSB/HAR-19/03, 2019. Source

[123] Laure Wynants et al. “Prediction Models for Diagnosis and Prognosis of COVID-19: Systematic Review and Critical Appraisal.” BMJ 369 (2020): m1328. Initial review published 7 April 2020; subsequently updated as a living review. doi:10.1136/bmj.m1328. Source

[124] Mata v. Avianca, Inc., 678 F. Supp. 3d 443 (S.D.N.Y. 2023), Opinion and Order on Sanctions, 22 June 2023. Source

[125] OpenAI. “Sycophancy in GPT-4o: What Happened and What We’re Doing About It.” 29 April 2025. Source

[126] OpenAI. “Expanding on What We Missed with Sycophancy.” 2 May 2025. Source

[127] Andrew Gelman and Donald B. Rubin. “Inference from Iterative Simulation Using Multiple Sequences.” Statistical Science 7, no. 4 (1992): 457–472. doi:10.1214/ss/1177011136. Source

[128] Andrew Gelman, Xiao-Li Meng, and Hal Stern. “Posterior Predictive Assessment of Model Fitness via Realized Discrepancies.” Statistica Sinica 6 (1996): 733–807, including discussion and rejoinder. Source

[129] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. “Concrete Problems in AI Safety.” arXiv:1606.06565, 2016. Source

[130] Miroslav Dudík, John Langford, and Lihong Li. “Doubly Robust Policy Evaluation and Learning.” Proceedings of ICML, 2011, 1097–1104. Source

[131] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc G. Bellemare. “Deep Reinforcement Learning at the Edge of the Statistical Precipice.” Advances in Neural Information Processing Systems 34 (2021). Source

[132] Ian J. Goodfellow et al. “Generative Adversarial Nets.” Advances in Neural Information Processing Systems 27 (2014). Source

[133] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium.” Advances in Neural Information Processing Systems 30 (2017). Source

[134] Alexander D’Amour et al. “Underspecification Presents Challenges for Credibility in Modern Machine Learning.” Journal of Machine Learning Research 23, no. 226 (2022): 1–61. Source

[135] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks.” Computer Aided Verification, LNCS 10426 (2017): 97–117. doi:10.1007/978-3-319-63387-9_5. Source

[136] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. “Membership Inference Attacks Against Machine Learning Models.” IEEE Symposium on Security and Privacy, 2017, 3–18. doi:10.1109/SP.2017.41. Source

[137] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. “BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain.” arXiv:1708.06733, 2017. Source

[138] Paulius Micikevicius et al. “Mixed Precision Training.” ICLR 2018. Source

[139] Benoit Jacob et al. “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference.” IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, 2704–2713. doi:10.1109/CVPR.2018.00286. Source

[140] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. “Not What You’ve Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection.” Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, 2023, 79–90. doi:10.1145/3605764.3623985. Source

[141] Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. “Language Models Don’t Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting.” Advances in Neural Information Processing Systems 36 (2023). Source

[142] Dmitry Vostokov. Theoretical Software Diagnostics: Collected Articles. Fourth edition. OpenTask, 2024. Source

[143] Peter J. Rousseeuw. “Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis.” Journal of Computational and Applied Mathematics 20 (1987): 53–65. doi:10.1016/0377-0427(87)90125-7. Source