

Launch an Effective Machine Learning Monitoring System

Published 1 August 2023 - ID G00783956 - 69 min read

By Analyst(s): Joe Antelmi

Initiatives: [Analytics and Artificial Intelligence for Technical Professionals](#); [Drive Quantifiable Value With D&A Solutions for the Business](#); [IT Operations and Cloud Management for Technical Professionals](#)

Machine learning monitoring in production is challenging due to production environment diversity, model integration complexity and best-practice scarcity. Data and analytics technical professionals can use this research to define benefits, pillars and best practices of monitoring machine learning.

Overview

Key Findings

- Machine learning (ML) models are essential to monitor because they are prone to drift that can cause performance degradation. However, anecdotal evidence from inquiry calls with Gartner clients shows that many large organizations are not monitoring ML models sufficiently to account for risks such as data, concept and model drifts.
- ML monitoring designs must account for three tiers: the data tier (e.g., feature verification), the model tier (e.g., concept drift) and the infrastructure tier (e.g., logging). Omitting any one of these in the design will result in a monitoring “blind spot.”
- Holistic ML monitoring integrates the following monitoring dimensions: upstream data, feature verification, training-serving skew, KPI score, concept drift, model metrics, ecosystem monitoring, logging and tracing.
- The major cloud vendors offer native machine learning monitoring capabilities that provide a useful starting point for ML monitoring in the cloud. A wide variety of third-party open-source and commercial off-the-shelf monitoring tools also offer sophisticated features.

Recommendations

Data and analytics technical professionals focused on ML monitoring should:

- Adopt some form of ML monitoring (open-source software, third party, cloud provider) as soon as you deploy ML models into production. Without visibility into model quality, technical professionals may miss insidious silent failures.
- Build a “detect and suggest” monitoring system by implementing capabilities that not only detect an anomaly, but also suggest what to do next. Build your monitoring system such that it gives you clues as to what to do next.
- Monitor crucial model system inputs and outputs, such as upstream data pipelines, model scores and system telemetry. As your ML deployments grow in complexity and automation, add more monitoring capability (for example, methods to detect concept drift) to enable agility and responsiveness to sudden negative changes.
- Scale your team as you scale your monitoring system as your ML systems grow in complexity by augmenting the initial core of data scientist, ML engineer, and data engineer with other roles. Mature monitoring requires the cooperation of a broader set of roles, including business domain experts, IT operations teams, application teams, model owners and architects.

Analysis

When an ML model is deployed into production, it becomes part of a ML system, which has all the challenges of traditional software and some additional ones related to their unique nature.

The core problem is that even when ML models are monitored and regularly retrained, they are subject to sudden and drastic declines in performance over time. In a recent research paper, Microsoft shared that the ML models it has deployed are prone to frequent and drastic accuracy drops (up to 40%) despite frequent retraining.¹

Generally, this decline is caused by something changing. In research by Microsoft, this change was most often related to data drift, often caused by a change in the upstream data pipeline, but there are many other factors that can degrade ML model performance.¹ These include model drift, concept drift, errors, outages and beyond. ML systems are complex and their performance is volatile and generally degrades.

Organizations must do something about this problem: The solution, most often, is monitoring the model and retraining it frequently. However, in scenarios where an incident occurs and model performance degrades, organizations must build the competency and capability to understand why performance degraded, resolve the issue and take corrective action.

This report helps data and analytics technical professionals get started with ML monitoring by:

- Identifying machine learning monitoring challenges
- Defining three essential monitoring tiers: data tier, model tier and ecosystem tier
- Analyzing nine monitoring capabilities that are most important for monitoring a model in production
- Describing the process of getting started with monitoring from a scalability perspective, as well as in the major cloud platforms

Machine Learning Monitoring Challenges

Multiple Things to Monitor

ML systems are composed of code (and configuration), models and data, as well as the infrastructure that supports them and the applications they integrate into. ML models also typically have some form of customer, user or use-case goal that they are attempting to accomplish and deliver business KPI gain. Moreover, ML model performance very often will degrade, making it crucial to have visibility into the complex workings of these systems. In complex environments, multiple ML models may be deployed in a variety of deployment models that together support an application – and all must be monitored. Testing in production (an antipattern for deterministic software) is a reality for machine learning systems because it is impossible to account for every failure node predeployment.

Multiple Monitoring Signals to Track

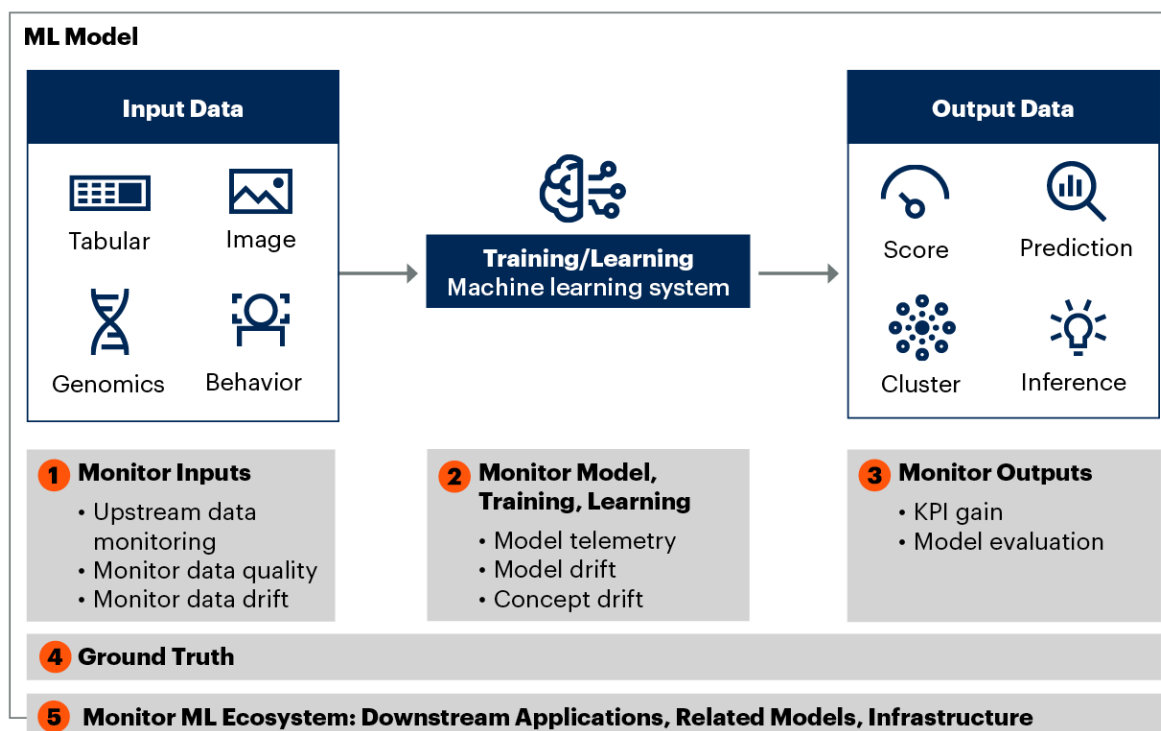
Depending on the production ML system, you may need to monitor the following, as depicted in Figure 1:

1. Model inputs (upstream data, features, code, configuration).

2. The model, and model training and learning. Looking for changes in external conditions that might indicate model/business drift.
3. Model outputs (downstream scores, predictions, inferences). Evaluating model accuracy based on the nature of the developed ML model. This includes model performance and model behavior from a functional (mathematical) and operational (business) perspective.
4. Outputs of the model compared to ground truth (or to a simpler heuristic). Ground truth in this context is the real-world truth, outcome or score that you can use to evaluate and validate the model prediction.
5. Measures of other components of the model system — for example, the performance and behavior of ML models that have some link or dependency with your model. Also monitor ML model system infrastructure via traditional monitoring signals (latency, traffic, errors, saturation).

Figure 1: Monitoring an ML System

Monitoring an ML System



Source: Gartner
783956_C

Ground truth is often difficult to collect. This crucial information about real-world outcomes can help evaluate model scoring/classifier performance, but not every use case makes this possible. For example, a churn prediction is hard to validate, especially if the organization takes a customer who is likely to churn and tries to stop them from churning. Much attention goes into monitoring ML inputs — that is, data or code or infrastructure — but it is similar to in other realms of monitoring, if you can monitor model outputs, you are likely to get the most important insights. Outputs include business impact, scores, the inferences and the applications the models contribute to. The outputs will tell you whether the model is actually doing what you wanted it to do, especially if you can compare model outputs to reality, or other measures of model value.

This essential practice of tracking model outputs and comparing them to ground truth data will give you a great picture of model performance, if it's possible.

Drift Happens

When drift takes place, something changes in the model system or external conditions that causes model performance to degrade.

There are three major types of drift:

- **Business** — This may include changes in business operation conditions and processes or shifts in business understanding and concepts that were set forth when the project was originally conceived.
- **Mathematical (or model)** — Your model is trained to approximate the mathematical function that captures the relationships or patterns inherent in the training data. However, when real-world data deviates from the training set in such a way that affects the underlying relationship and patterns the model learns, we call it “mathematical drift” or “model drift.” This can result in your trained model degrading in performance or precision slowly over time (model decay) or becoming entirely ineffective (model stale) in the new problem space, thus requiring retraining.
- **Data** — This accounts for any shift from the original training dataset to the actual data seen in production that may render the model ineffective. Examples of data drift include changes in data format or structure, data range and distribution. When the drift occurs with the input features, it is called “feature drift.” When the drift occurs on the output side, it is called “label drift.”

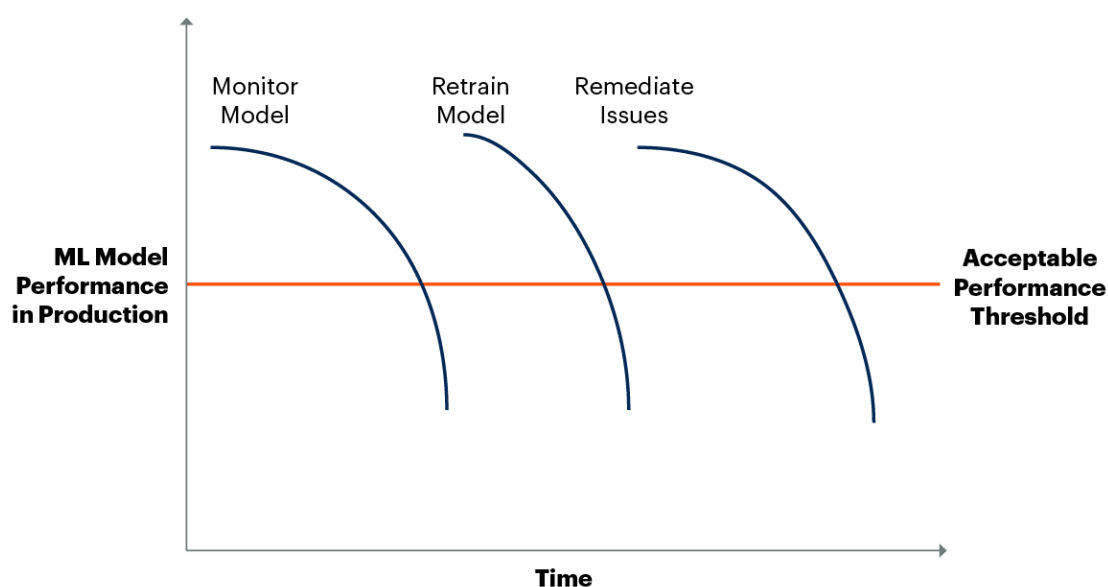
Figure 2 shows you can monitor model performance and retrain the model to keep its performance within certain thresholds. The downward curve indicates a degradation of the model performance, which is captured due to monitoring. Upon detection, the corrective action is taken to retrain the model, which brings model performance back to where it is supposed to be.

In theory, this seems straightforward. In the real world, monitoring model performance, and figuring out which corrective action to take, is challenging.

Figure 2: Monitor and Retrain Model to Improve ML Model Performance

Monitor and Retrain Model to Improve ML Model Performance

Illustrative



Source: Gartner
783956_C

Gartner

Data Issues

Data is a crucial input to ML system behavior, and data issues are common with machine learning.

ML system behavior is governed not just by rules specified in the code, but also by model behavior learned from data. Moreover, those data inputs are often unstable and subject to change as time progresses. If you can't capture the changes to the inputs of the ML system, you can't understand the system.

There is a long list of potential problems you can have with data in machine learning pipelines. These could include:

- Version conflicts
- Out-of-date permissions
- Broken infrastructure
- Faulty queries
- Schema changes
- Data loss

In general, inputs that get changed in upstream data systems can cause schema drift to occur. Inconsistencies between feature training and serving pipelines, perhaps caused by two different languages being used for development and production systems, cause training/serving skew.

Entanglement

Machine learning system components all depend on each other. For example, when an input feature is changed, then the importance, weights or use of the remaining features may all shift as well.

Complex systems, like weather, economies or traffic, are inherently unpredictable. As ML systems usually rely on digital representations of the real world, tiny variations in features, settings or measurement may dramatically shift results. Thus, as a baseline, feature engineering and selection code must be carefully tested. Moreover, large production deployments of ML tend to involve many models, which may have dependencies with each other. This is another dimension of entanglement that organizations must account for.

MLOps Immaturity and Complexity

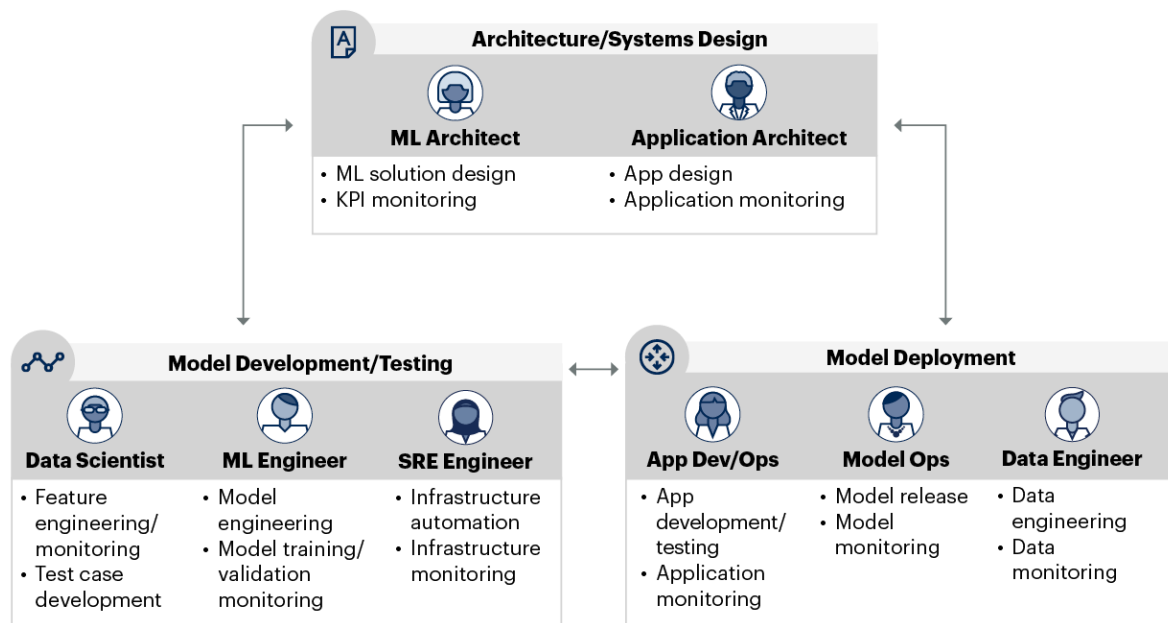
Organizations are often just starting out with enabling the machine learning operations (MLOps) capabilities that they need for robust ML systems. Without good versioning, model management and configuration management, model monitoring becomes very difficult. Often, organizations will need to make small changes to models in production, and they need robust MLOps capabilities beyond monitoring to enable this. (For more on MLOps, and its relationship to model operationalization, or ModelOps, see Note 1 and Note 2.)

Overlapping Responsibilities Among Teams

Monitoring, ideally, is a shared responsibility. It includes roles such as ML engineer, ML architect, IT architect, data engineer, application architect, and infrastructure and operations (I/O) professional. But in practice, teams often don't prioritize adding monitoring features to their ML systems. In addition, the different roles (especially across groups/departments) each have a different view on what they need to monitor for, which, without sharing, creates tunnel vision as everybody is focused on their own (see Figure 3).

Figure 3: Shared Team Responsibilities for ML Monitoring

Shared Team Responsibilities for ML Monitoring



Source: Gartner
783956_C

Tool Complexity and Immaturity

Monitoring requires an understanding of why to monitor, what to monitor and how to monitor. However, tool complexity complicates the what and the how. This is illustrated in the [Getting Started With Machine Learning Monitoring In the Cloud](#) section later in this research.

Scaling Monitoring Is Difficult

Due to the above challenges, scaling monitoring requires teams to adopt better tooling, define responsibilities (and hire MLOps engineers) in a cross-functional team and incentivize different roles to prioritize building monitoring features. Prioritizing what needs to be built, and how to go from monitoring to observability, become priorities for large teams.

ML Monitoring Goals

To solve these challenges, technical professionals must first define what they are trying to achieve. Then, they must determine what gaps exist in their current capability and what will be necessary to achieve these goals.

Determining what to monitor is critical to designing your monitoring system. As tempting as it may be to err on the side of monitoring as many different things as possible, that turns out to be counterproductive. Not only does the collection and storage of unnecessary data place unnecessary demands on technology resources and personnel, but it also slows down problem resolution and can cause unnecessary alerts.

Use the following five foundational principles to guide your ML monitoring system design.

Principle 1: Define Meaningful and Measurable KPI Gain

The primary value of ML in production is not about accuracy or precision. Instead, it is about solving a business problem. Key performance indicators quantify the business value for your project. Measure the KPI gain over time. You will find that the more time you spend on understanding the goals of your ML project and defining the problem at hand, the easier it will be to determine what to monitor. This is because during the model evaluation phase, you will have some specific time-bound goals that you wish to achieve with an understanding of the factors that affect these goals.

The high-level business goals your project aligns to may be related to increasing revenue, decreasing cost or reducing risk, for example. But you often have to go deeper to define the problem properly. Moreover, you'd want to look to measure model lift, which is the improvement of the business KPI that you can attribute to the model.

Principle 2: Detect Anomalies and Suggest Actions

Resilient ML models can continue to operate and provide a score, even when data patterns change and thus mask an underlying problem. ML monitoring must detect when there is a serving data pattern change that affects KPI gain (e.g., missing features, concept drift) and must suggest what to do next (e.g., retrain the model, promote a challenger, etc.). Design your monitoring system such that it gives you clues as to what to do next.

Principle 3: Support Model Explainability

The inherent inability to interpret the behavior of a model increases risk and reduces trust in ML model usage. We are at a point where the ability to explain the predictions and interpret the models is not an option but a necessity (see [Incorporate Explainability and Fairness Within the AI Platform](#)). While most of the model development focus is on predicting the “what” and “why” behind the prediction, it is equally important to trust the “how.” Organizations should evaluate ML explainability toolkits and product features and align them with use cases (regulatory, end-user or customer-centric, or troubleshooting).

Principle 4: Support Ongoing Model Deployment

Rarely will an organization train and deploy a model that is never retrained or replaced. The opposite is more likely to be true. Therefore, you must design your ML monitoring system to support a process where there is ongoing model deployment. The design must include the ability to simultaneously support many models that are in various stages of operation. The design must also be able to support the retraining process by collecting data for model training. See [Machine Learning Training Essentials and Best Practices](#) for more information on model training.

Principle 5: Support Multiple Roles

ML monitoring involves the collaboration of cross-domain expertise and therefore must be designed to support multiple roles. Designing an effective ML monitoring system often requires a range of expertise. Plan to combine the knowledge of these roles to design and deploy your ML monitoring system:

- Data scientist (e.g., data science platforms, algorithms, metrics)
- ML engineer/MLOps engineer
- Subject matter expert (SME; e.g., manufacturing process engineering, financial risk assessment)
- Application architect (e.g., microservices, APIs, packaging)

- Operations team (e.g., storage, compute and networking systems)

What and How to Monitor

As you design this system, take first principles into account. Three pillars of data science influence the architecture of an ML monitoring system:

- There must be data (e.g., training and testing data).
- There must be observable data patterns.
- The data patterns must be stable over some time period.

Data scientists design models that exploit stable data patterns to make inferences based upon new (i.e., serving) data. ML monitoring must be able to detect anomalies (e.g., missing features, training-serving feature distribution skew, concept drift) that are indicators of a disruption to data pattern stability.










In addition, ML monitoring systems must suggest what action to do next. For example, if missing data is a problem, the monitoring system could surface information on this change to the upstream data pipeline. The more the monitoring system goes to reactive, to proactive, with remediation, the easier it is to handle large numbers of models deployed in production.

Anomalies can emerge from three sources: the serving data, the deployed models or the operational infrastructure that serves the models. Therefore, technical professionals must architect their ML monitoring system into three tiers that focus on these anomaly sources (see Figure 4):

- **Data tier** — Focuses on the serving data being fed into the model and includes monitoring upstream data, feature verification and training-serving skew.
- **Model tier** — Focuses on operating characteristics of the model and includes monitoring KPI scores, model metrics and concept drift.
- **Ecosystem tier** — Focuses on broader dependencies, which may include infrastructure that supports model operations, the applications that the model supports downstream and the overall system. In practice, it typically includes different signals from distributed systems, which may include metrics, traces and logs.

Figure 4: Three Tiers of ML Monitoring

Three Tiers of ML Monitoring

Data Monitoring	 Upstream Data	 Feature Quality	 Feature Skew
Model Monitoring	 KPI Score	 Model Metrics	 Concept Drift
Ecosystem Monitoring	 System Dependencies	 Event Logging	 Inference Latency

Source: Gartner
783956_C

Verification vs. validation: *Verification and validation are closely related terms that define different aspects of quality assurance. In the context of this research, we define the terms as follows:*

- **Verification:** *The evaluation of whether a system complies with the system requirements. Example: Did we build what we intended to build, irrespective of whether it was the right thing to build?*
- **Validation:** *The evaluation of whether a system satisfies the needs of the stakeholders that imposed the system requirements. Example: Did we build a system that solves the problem we intended to solve?*

Within the context of machine learning, we use the following terms throughout this research:

- **Feature verification:** *The evaluation of whether the ML features used by the training/serving systems satisfy the system requirements. Example: Was each feature correctly formatted (e.g., INT64 versus STRING)?*
- **Feature validation:** *The evaluation of whether the ML features used by the training/serving systems satisfy the needs of the stakeholders that imposed the system requirements. Example: Did we use the correct features (e.g., FICO score) to solve the business problem (e.g., predict loan default)?*

Data Tier

The data tier focuses on the data flowing along the serving data pipeline. Data tier monitoring provides the following capabilities: upstream data monitoring, serving feature verification and training-serving skew detection.

Upstream Data Monitoring

Traditional batch data pipelines perform data quality checks to determine whether data was lost or whether expected values are out of range. Similar data quality checks are required along the machine learning data pipeline because data changes and/or errors can impact model scoring. For example, changes to feature encoding (e.g., Fahrenheit to Celsius) or null value features ² can impact the resultant score produced by the trained model running on an inference server.

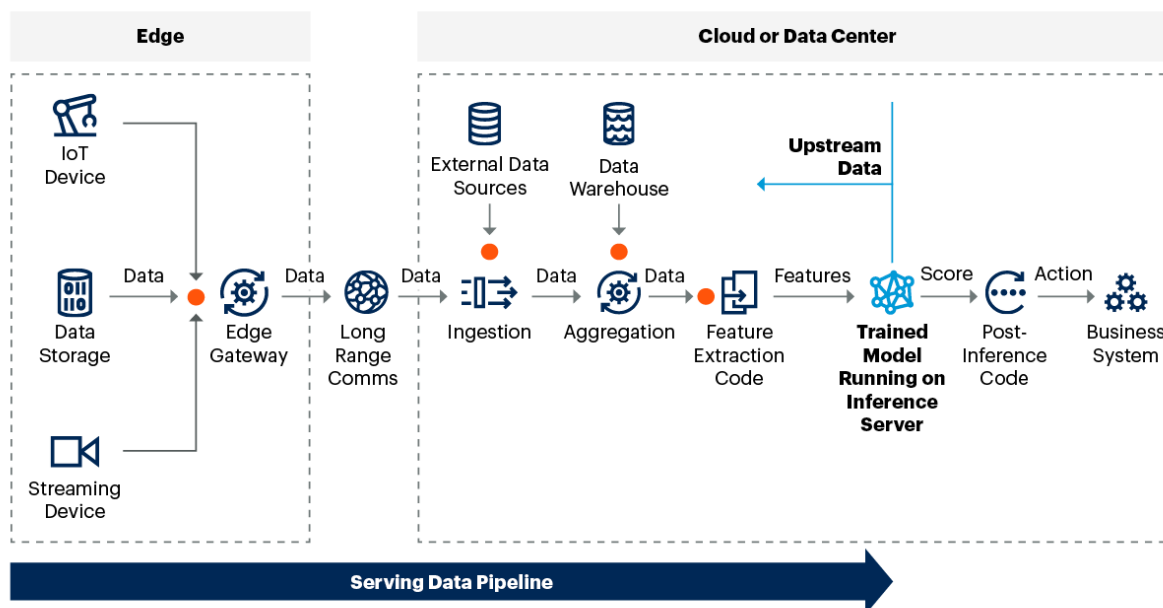
Figure 5 shows that the upstream data pipeline includes Internet of Things (IoT) devices, edge data storage systems and streaming devices. These devices generate data that flows along the pipeline to the ML model:

- The edge gateway ingests, aggregates and transforms this data before forwarding it along the pipeline.
- The serving data pipeline also includes data from external data sources (e.g., weather data) and from the logical data warehouses (e.g., historical data).
- Finally, the feature extraction code prepares the data (e.g., features) before passing it to the trained ML model.

Figure 5: Upstream Data Monitoring

Upstream Data Monitoring

● Data Monitoring



Source: Gartner

451229_C

Gartner

Recommendations: Upstream Data Monitoring

- **Identify data sources:** ³ Identify which upstream systems feed data to your model. Ask:
 - Where do the systems reside in the pipeline?
 - What data do these systems provide to your model?
 - Who owns/manages the systems?
 - Are certain data sources more important to your model performance than others?
 - What trust level will we set for each data source? While a data source of low trust may be usable, what expectations do we have for this source and how closely do we need to monitor it?
- **Identify data assumptions:** Your data assumptions form the list of the items that you must monitor. For example:

- Does your feature extraction code or ML model make any assumptions about this data? For instance, must the data be within a certain range (e.g., 1 to 100) or of a certain type (e.g., string)?
- For streaming data, must it arrive at a predetermined rate (e.g., every two to three seconds) or in a certain format (e.g., XML)?
- **Monitor data close to the source:** The semantic meaning of the data may be lost as it flows through the pipeline. Therefore, monitoring as close to the data source as possible increases the likelihood of identifying and repairing the problem. In addition, data transformations along the pipeline can amplify even a small change and affect model scoring in ways that may not be easy to diagnose. How will you monitor those transformations?
- **Monitor upstream system changes:** Make sure you are aware of changes to upstream hardware and software that may affect the data flow through the pipeline. Configuration changes may have a subtle effect on the pipeline data that may result in a change to model scoring.

Serving Feature Verification

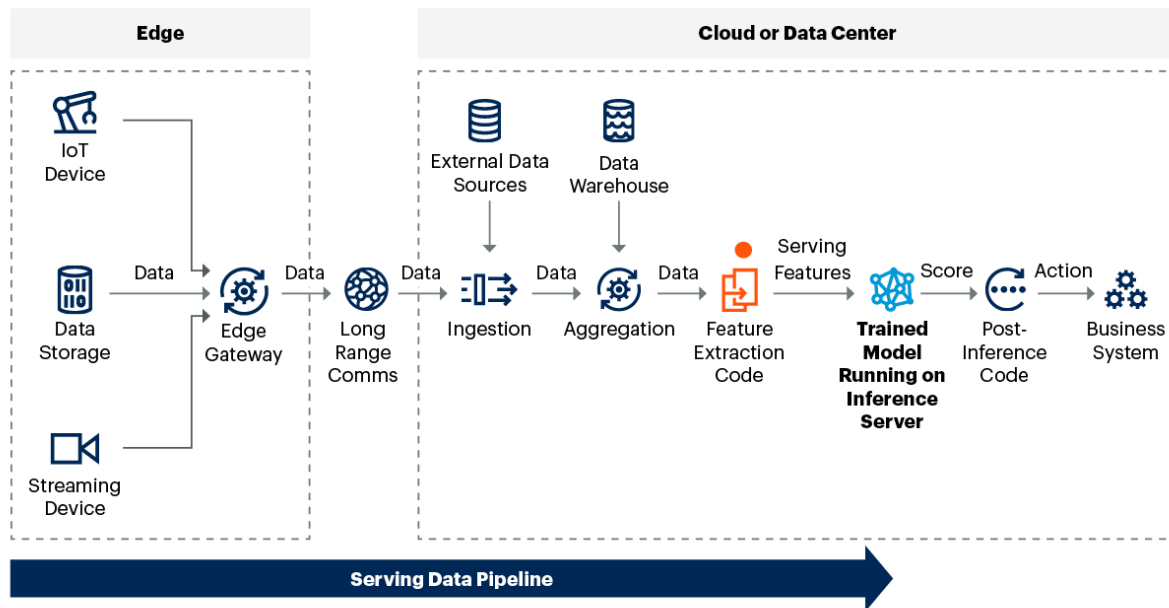
Data flows along the serving data pipeline and eventually reaches software that extracts the necessary features from the data and passes them to the trained model. The model uses the features to generate a prediction, a classification, cluster, and so on (aka a “score”). If some of the serving features are missing, out of range or otherwise in error, these errors can affect the score.

A monitoring system must be able to verify the serving features. Feature verification will occur in the serving data pipeline at a location prior to the trained model (see Figure 6). The feature extraction code will commonly pass features to the verification code as a confirmation check prior to (or in parallel with) passing those features to the model.

Figure 6: Serving Feature Verification

Serving Feature Verification

● Feature Verification



Source: Gartner

451229_C

Gartner

For example, LendingClub has developed its fifth-generation credit model, which leverages machine learning and 10 years of LendingClub data to assess and price credit risk. The fintech marketplace bank makes its data available for free download and provides API access to its platform. The model score helps LendingClub lenders to facilitate loans to people whom traditional lending programs miscategorize as too risky but who are actually good borrowers. The model also screens out borrowers who seem safe but are risky based on the credit model and historical data.

A model monitoring system for the LendingClub model should provide feature verification that detects errors, such as:

- Wrong feature type (e.g., model expects FICO score feature, but actual value is a Boolean).
- Missing features (e.g., model expects borrower's last name, but last name is missing),
- Out-of-range feature (e.g., model expects a loan duration feature to be in the range zero to 60 months, but the actual value is 4,000).

These feature errors could cause the model to predict that borrowers are good credit risks when in fact they are not (or vice versa). If you do not verify the serving features, then how would you know if the model score was affected by “bad features”?

The problem of feature errors can impact not only the model currently in production, but also future models not yet trained or deployed. The authors of “Data Validation for Machine Learning” explain the issue below. ⁴

“Consider an ML pipeline that trains on new training data arriving in batches every day, and pushes a fresh model trained on this data to the serving infrastructure. The queries to the model servers (the serving data) are logged and joined with labels to create the next day’s training data. This setup ensures that the model is continuously updated and adapts to any changes in the data characteristics on a daily basis.

“Now, let us assume that an engineer performs a (seemingly) innocuous code refactoring in the serving stack, which, however, introduces a bug that pins the value of a specific int feature to -1 for some slice of the serving data (e.g., imagine that the feature’s value is generated by doing a remote procedure call (RPC) into a back-end system and the bug causes the RPC to fail, thus returning an error value). The ML model, being robust to data changes, continues to generate predictions, albeit at a lower level of accuracy for this slice of data.

“Since the serving data eventually becomes training data, this means that the next version of the model gets trained with the problematic data. Note that the data looks perfectly fine for the training code, since -1 is an acceptable value for the int feature. If the feature is important for accuracy, then the model will continue to underperform for the same slice of data. Moreover, the error will persist in the serving data (and thus in the next batch of training data) until it is discovered and fixed.”

Feature verification is thus a critical capability in a robust ML monitoring system.

Recommendations: Feature Verification

- **Capture feature expectations in a schema:** Recent research recommends that technical professionals encode subject matter expert insights about feature data in a schema, to help them verify features prior to passing them to a production model. ^{4,5} For example, if a loan borrower's age is one of the model features, then one can certainly assume that the input value for this feature must always be in the range 1 to 200. Start by visualizing your training data (e.g., use an open-source tool such as [Facets](#)) and then analyze the value distribution for each feature.
- **Develop feature verification code:** Consider using open-source feature verification libraries to reduce your development investment. For instance, [TensorFlow \(Google\) Data Validation](#) (TFDV) is a library for exploring and verifying machine learning data. The library automates [data-schema generation](#) and [detects anomalies](#) such as missing features. It is instructive to review these libraries, even if you are not using TensorFlow. The open-source code can provide examples of feature verification that you can copy and use in your application. (Note: The libraries use the term "data validation," but as defined here, they are talking about data verification.)
- **Regression test feature extraction code:** ⁵ An important step in verifying each feature is to carefully test the code that prepares and submits the features to the operational ML model. Subtle bugs in this code could have a significant impact on model scoring and may be difficult to debug. Be sure to perform regression testing to the feature verification and creation code when you make changes to support new models. This is especially important when you "roll back" an ML model to an earlier version. A model rollback may expose a bug in the feature verification code or feature extraction code that the developer introduced when making changes to support a new model. Regression bugs are difficult to diagnose because the code used to work with the old model, but now it doesn't.
- **Consider using a declarative API for feature verification:** One thing that companies that have many ML models in production do is look for ways to scale services. In line with that reasoning, a more generalized approach to data verification could be achieved by defining a declarative API, which combines common quality constraints with user-defined verification code, and thereby enables "unit tests" for data. ⁶

Training-Serving Skew

Training-serving skew occurs when the training system differs from the serving system in such a way that the differences affect model scoring. A common example is feature distribution skew. This type of training-serving skew occurs when the feature distribution used for model training substantially differs from the feature distribution observed during model serving.

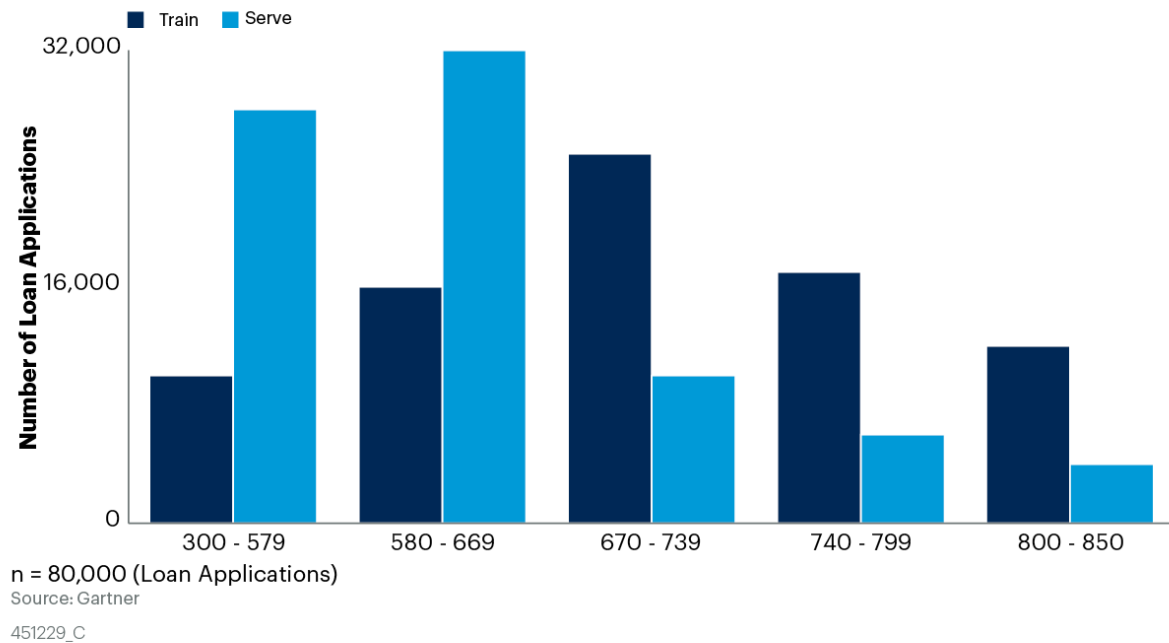
Let's continue with the LendingClub example on predicting credit risk to illustrate what we mean. Assume that a data scientist designed the credit risk prediction model with features that included the borrower's FICO score, current salary and requested loan amount. Further assume that the data scientist used a training dataset that included 80,000 loan applications. Figure 7 shows the FICO score distribution used for training (dark blue) and the distribution observed during serving (light blue). The FICO score training and serving distributions look very different. The training data had a normal distribution. In contrast, the serving data distribution contains many more low FICO score applications than the training data.

A feature distribution skew indicates that the training data does not reflect "real world" serving data. When there is a large and persistent training-serving feature distribution skew, this may indicate that it is time to retrain the model. This is especially true when the skew occurs with features that have the most impact on model scoring. ⁷

Figure 7: Training-Serving Feature Skew

Training-Serving Feature Skew

FICO Score Distribution



Gartner

In addition to feature distribution skew, training-serving skew can occur for other reasons:

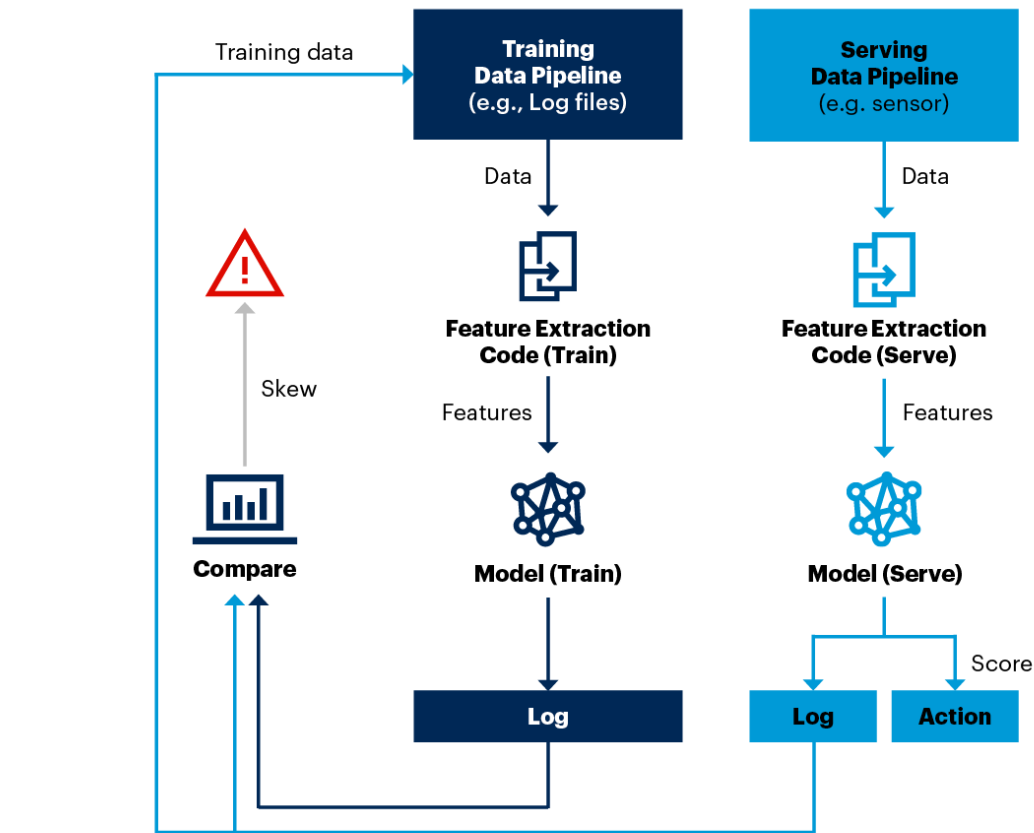
- **Feature extraction skew:** ⁸ This occurs when there is a discrepancy between training and serving feature extraction code. This could occur when the developer must simulate some training data to generate features for model training, because the data only becomes available in real time for model serving. Another example is that the developer makes a change to the feature extraction code for the serving path but fails to make that same change for the training path. Feature skew can impact model scoring and can introduce training bias if the scored data is used to train new models.

- **Scoring-labeling skew:** ⁸ Organizations will often use the scores from an active model to train a new model. Scoring-labeling skew occurs when only a subset of the scored examples is used by the production system and thus only a subset of the examples is labeled and available for training the new model. Scoring-labeling skew can introduce training bias. For example, imagine a website that sells cat photos. The website uses customer search terms and previously viewed cat photos as input to an ML algorithm. The ML algorithm generates 10 photos that it predicts the customer will like. But the web developer only displays five of the photos. When the client selects one of the photos, that photo will be logged and labeled as “selected.” The other four photos will be logged and labeled as “not selected.” The remaining five photos that the algorithm generated (but the developer did not display) will be logged, but will not be labeled. If the log data is used to train future models, the unlabeled data will not be used for training the new model. This may introduce a training bias — perhaps one of the cat photos not displayed would have been selected.

Figure 8 conceptually illustrates how to design a system to monitor for feature distribution skew. Note the training data pipeline on the left and serving data pipeline on the right. The monitoring system logs input features and output scores for every prediction request. This data is then fed into a data visualization tool to compare the training and serving feature distribution. It is not necessary to compare the distributions of every feature. Focus on the features that have the greatest impact to the model scoring. Alternatively, run a two-sample test to compare the distributions.

Figure 8: Training-Serving Skew Monitoring

Training-Serving Skew Monitoring



Source: Gartner
451229_C

Gartner

Recommendations: Training-Serving Skew Monitoring

- **Reuse code between your training pipeline and your serving pipeline whenever possible:** ⁸ Training code can be different from serving code for a host of reasons. For example, the training code is in a Jupyter Notebook and then is converted into a Python script, and this creates divergence. Over time, different codebases will diverge, introducing a source of subtle behavioral differences between the training and serving models. Try to minimize the code differences. Also, if possible, use the same programming languages and development tools (such as a single DSML platform) for both training and serving code.

- **Log serving data, features and score:** ⁸ Log the data, features and score for your serving data pipeline. The log data will be used to measure the various forms of training-serving skew. It may not be practical to log everything. You can reduce the volume by logging a percentage of the data. You can reduce the logging volume even further by only logging the features that have the biggest influence on the score. You need to make this decision carefully. If you need to be able to reproduce a specific result for a given period, you will need specific retention policies.
- **Compare training and serving feature extraction code:** Use the log data to monitor feature skew. Feed the log data into the feature extraction code in the training data pipeline. Did the code produce the same features as the feature extraction code in the serving data pipeline? If not, you have feature skew. You should compare the training and serving code to determine why there is a discrepancy and fix the problem.
- **Compare training and serving feature distribution:** It is important to monitor the change in the distribution of model features. A change in distribution may indicate that serving data is drifting from training data, in which case the model may need to be retrained. Some ModelOps systems can monitor for feature distribution skew for you. Pure-play ML monitoring tools (e.g., Arize AI) are also available to help you to monitor feature distribution skew. It is important to correlate feature distribution skew with feature importance. Not all features contribute equally to predicting the score. Therefore, not all instances of feature distribution skew require the same attention as others. Focus first on those features that exhibit training-serving skew and that have the greatest contribution to the score.
- **Deploy a feature store to improve feature management and monitoring:** Feature management is critical for the success of ML initiatives within organizations. Storing features and related metadata facilitates sharing across projects, decreases ML model development times and promotes collaboration (reusability). Time stamps and feature versioning enable the exact conditions under which a model was developed to be recreated (reproducibility). This promotes model explainability, audit and retraining. Providing a consistent view of features across development and production, and monitoring features to detect and remediate issues, preserves ML model performance over time (reliability). Feature stores aim to bring these capabilities to ML workloads while scaling with the organization's maturing ML portfolio. For more information, see [The Logical Feature Store: Data Management for Machine Learning](#).

Model Tier

The model tier focuses on operating characteristics of the model. It monitors the following three dimensions: KPI gain, concept drift and model metrics.

KPI Gain

The value of machine learning is not necessarily about accuracy or [Area Under the Curve](#) (AUC). The value of ML is about solving a business problem. [Model Scoring](#) (aka prediction) is the outcome resulting from the execution of a machine learning model. Scoring is the method by which we understand whether the machine learning model is doing what we intend it to do. The semantic meaning of a model score depends upon the business problem you are trying to solve.

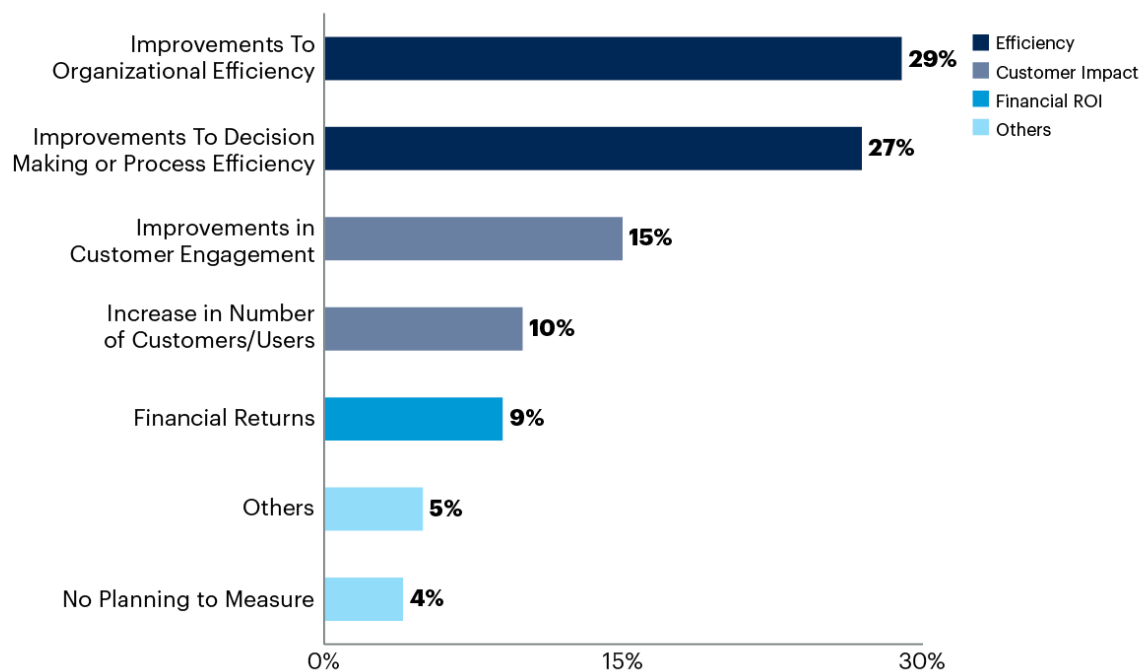
To generate a score, the ML inference server must accept the input features, load the trained model, execute the model and return the score to the code that provides postinference processing. The scoring may occur in real time (e.g., selecting the best videos to watch right now) or it may occur as part of a batch process (e.g., analyzing historical behavior of video selection for the past 30 days).

A model may perform well mathematically, but not solve a business problem. This is why so many ML projects never achieve their desired business outcome.

Model scores must be aligned with KPIs that measure business value — hence the term “KPI score.” Measures of ML project success may include KPIs such as “improve process efficiency” or “improve customer engagement” (see Figure 9).

Figure 9: Measures of Success for AI and ML Initiatives

Measures of Success for AI and ML Initiatives



n = 78

Base: Gartner Research Circle members who have currently deployed/are deploying projects in the next zero to 12 months, excluding "unsure"
 Q. What is the primary measure that your organization is currently using/planning to use to determine the success of its AI or ML initiatives?

Select all that apply.

Source: Gartner AI and ML Development Strategies Survey

451229_C

Gartner

Recommendations: KPI Score Monitoring

- **Define your business problem as you define your KPI:** Determine the specific business problem you are trying to solve as you define the KPIs for your ML initiative. Often, determining the specific business problem that can be measured by a specific KPI is an iterative process born out of a vaguely defined idea or an observation of patterns in your training data. This may require collaboration among a variety of roles such as a domain expert (e.g., manufacturing expert, data scientist, business analyst, application architect, data engineer, etc.). Define the KPI in a way that succinctly measures the business problem you are solving (e.g., improve customer churn rate). Define your current state (e.g., current customer churn rate = 5% per month). Set a KPI goal that captures how you hope to change the current state (e.g., future customer churn rate = 3% per month). Seek out SMART objectives (specific, measurable, achievable, relevant, and time-bound). Once you clearly understand the business problem and have defined the KPIs, the data science team can complete the process of feature engineering and model training.

- **Create your KPI score by integrating domain knowledge and data science knowledge:** The tendency of a data scientist will be to view KPI as model accuracy or some other type of model metric. But the model that is the most accurate is not necessarily the best. The tendency of a domain expert will be to view KPI as an operational problem. But they may know nothing about data science or where to get training data. You must integrate the domain expert's deep knowledge (e.g., chemical process manufacturing, mortgage risk analysis, customer churn indicators, etc.) with the data scientist's deep knowledge (e.g., linear regression models, deep learning, data science platforms, etc.) to create the KPI score. Business-oriented KPIs tend to provide a "top-down" view, whereas model-oriented KPIs tend to provide a "bottom-up" view. The process of defining KPIs usually iterates between these high-level and bottom-up viewpoints. In addition, you must look for other factors (e.g., [model explainability](#)) that may also be important characteristics of your KPI.
- **Deploy a model with a simple KPI score definition, then refine:** Often, a simple measure of KPI score is good enough to get you started. Don't wait to try and create the "perfect" KPI score. Deploy the model with a "good enough" measure of KPI. Then, as you retrain and deploy your model, refine and deploy your KPI score.
- **Define and monitor KPI gain/loss:** KPI gain/loss, which can be also viewed as model lift, is the improvement/degradation in the KPI score when comparing scores from the active ML model to the baseline. The baseline could be the KPI score calculated prior to running machine learning (e.g., current customer churn rate = 5% per month) Or, the KPI gain/loss could compare the active ML model to the previously active model or to shadow models (see the [Ecosystem Tier](#) section). You will need to define how you will measure KPI gain/loss and establish a mechanism to monitor KPI gain/loss. Some MLOps tools provide a way for you to define cost metrics and measure KPI gain/loss.
- **Monitor KPI score distribution:** Monitoring systems often view model scores as a probability that a prediction is true (e.g., the model predicts that the manufactured product contains a defect with 95% certainty). Over time, the model will produce many scores. The distribution of these scores represents the behavior of the model over time. If the KPI score distribution changes when compared to previous model behavior or compared to the behavior of a canary model (see the next section below, Concept Drift), then this could indicate a potential problem. For example, the input data may have significantly changed, or the feature extraction code may have changed. Monitor the KPI score distribution and look for shifts from the baseline.

Concept Drift

Concept drift occurs when your understanding of the data used to build a model changes, even though the data may not change. Put another way, concept drift is a drift of scoring labels over time for essentially the same data.⁹ Concept drift is a difficult machine learning problem. The reason is that the concept of interest may depend on some hidden context, not given explicitly in the form of predictive features.¹⁰ Changes in the hidden context can induce radical changes in the target concept.

A good example of concept drift is how word sentiment has varied dramatically over historical time periods.¹¹ In the 1860s, “lean” used to be associated with “weakness” or “frailty,” but over time it became more associated with “fitness” and “muscularity.” Clearly, the data (i.e., the English characters that form the word “lean”) did not change. But the sentiment of the meaning of the word (i.e., the “concept”) did change. There is a hidden relationship between the input data (“lean”) and the output predictions (“weak,” “muscular”) that has changed over time.

Feature distribution skew vs. concept drift

Feature distribution skew is different from concept drift. Feature distribution skew occurs when the distribution of feature values over time in the training path differs from the distribution of feature values over time in the serving path (see the [Training-Serving Skew](#) section). In contrast, concept drift refers to a change in the underlying model concept even though the training/serving data may not change. Concept drift and feature distribution skew may occur independently or simultaneously.

Measuring concept drift for a deployed model using live data is a difficult problem. This difficulty arises because we need access to both the predictions the model generated (i.e., the score) and the ground truth (i.e., the labeled data used for training a supervised model).

Concept drift may exhibit several patterns:¹²

- Temporal consistency to the change (e.g., concept drift occurs every summer)
- A gradual change over time
- A recurring or cyclical change (e.g., concept drift occurs every 28 days)
- A sudden or abrupt change

Recommendations: Concept Drift Monitoring

- **Monitor for KPI gain changes over time:** If your KPI gain degrades over time, it may be because the real world has changed. If so, then it is time to look for different data patterns and retrain your model. The data patterns you used to train the model may have changed, and thus your model concept may have drifted. To monitor for KPI gain changes, you will need to know whether your model predictions were correct (so that you can calculate gain). This may not always be possible to do in practice because the true outcome of your prediction may not be known for a long time (or at all).
- **Monitor for feature distribution skew for most important features:** Feature distribution skew indicates that the “real world” feature distribution is drifting from the training data feature distribution. The “most important” features are the features that have the greatest influence on your model predictions. Feature distribution skew may indicate that your model concept has drifted. If so, then it is time to look for different data patterns and retrain your model. Refer to the [Training-Serving Skew](#) section for additional details on feature distribution skew.
- **Compare active model KPI gain against canary model KPI gain:** Often, an organization will serve an active model and a canary model. The canary model (aka baseline model) is typically a trusted, resilient model. If you begin to see that the active model KPI gain is diverging from the canary model, then the active model may be experiencing concept drift. If so, it's time to look for different data patterns and retrain your model.
- **Adapt your concept drift strategy to the unique nature of your data workload:** For example, for streaming data, adaptive windowing is a useful method; whereas for batch data, the Kolmogorov-Smirnov test, the chi-squared test or adversarial validation are common approaches.

Model Metrics

Model metrics (e.g., accuracy, precision, root mean square error, etc.) measure mathematical model performance. Metrics help the data scientist design, train and refine machine learning models. The ML monitoring system must provide visibility into the most important metrics for the specific use case.

However, there are so many different metrics, how does the reader understand what they all mean? And how does the reader determine which metrics are the “most important”?

The good news is that while the model designer needs to be an expert on model metrics, the monitoring system designer does not.

The monitoring system designer should simply have a general understanding of model metrics in order to intelligently communicate with the model designer. (See Note 4 for a brief introduction to classification model metrics and Note 5 for regression model metrics.)

The model designer will often indicate which metrics are most important. But the monitoring system designer must understand why these metrics are important for their use case.

For example: Imagine a data scientist designed a model to classify when brain scan images are “cancerous” or “not cancerous.” When the model classifies an image as cancerous, it is important that it be correct because an incorrect prediction tells the physician that the patient has cancer when they do not. It is also important that the model identifies all the cancerous images because an incorrect prediction tells the physician that the patient does not have cancer, when they do.

The model designer will use the language of model metrics to state that the model must be highly precise and highly sensitive (precision and sensitivity metrics can be derived from the confusion matrix). The model designer will probably want the model monitoring system to monitor the F1 metric (the F1 metric combines precision and sensitivity by providing the harmonic mean of precision and sensitivity). The monitoring system designers do not need to understand the mathematical intricacies of the F1 metric. But they do need to understand how to monitor this metric and why it is important for the particular use case.

Recommendations: Model Metrics

- **Don’t confuse model metrics with KPI scores:** Metrics analyze model performance using mathematical equations (e.g., accuracy, precision, root mean square error, etc.). In contrast, KPI scores measure model performance against business benchmarks, such as churn reduction.

- **Work with your data science team to determine the best metrics for your application:** Gain a basic understanding of the most relevant model metrics for your use case, but you need not become an expert in model metrics to successfully monitor your production models. Ask the members of the data science team that designed, trained and tested the model to help you understand what metrics are most important to monitor and why. In particular, learn how monitoring metrics will help you to achieve your overarching goal of improving the KPI score and what kind of information your data science team needs to improve models.
- **Use the model metrics provided by your MLOps platform or the metrics available in a specific ML monitoring tool:** The activity of bringing ML into production, also known as machine learning operationalization (MLOps), helps technical professionals operationalize their ML models. DSML platforms such as DataRobot integrate MLOps capabilities and offer model monitoring metrics. Also, ML monitoring/observability solutions offer increased depth of insight into model performance in production (vendors include Arize AI, Aporia, Superwise (Blattner Technologies), Fiddler and TruEra).

Ecosystem Tier

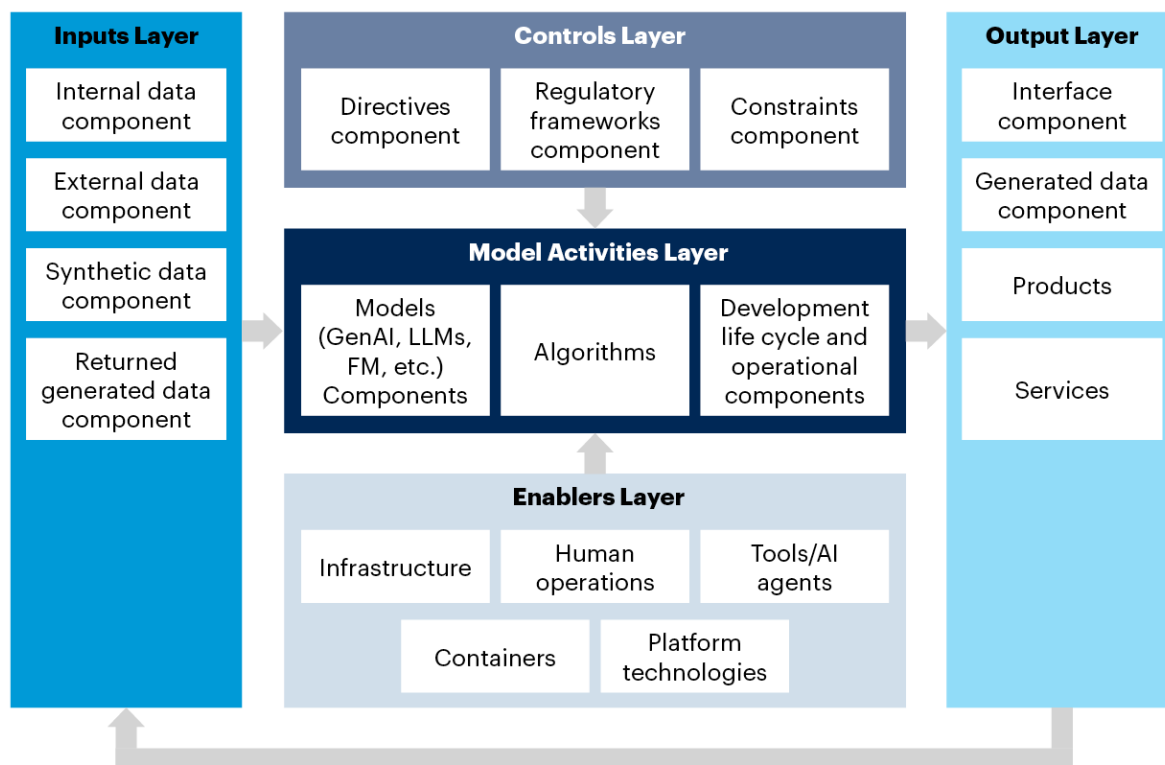
The ecosystem tier focuses on dependencies and scenarios. It monitors the following three dimensions: system dependencies, event logging and inference latency.

System Dependencies

The ML system is complex and requires a sophisticated approach to testing and monitoring. Taking a systems approach and mapping out inputs, outputs, model activities, controls and enablers, can be a successful approach to understanding the functioning of the entire ecosystem. Some simple applications might be possible to test before production, and monitor in production, but in ML systems, testing and monitoring must both happen in preproduction and production. See Figure 10.

Figure 10: ML-Based System Monitoring and Testing

ML-Based System Monitoring and Testing



Source: Gartner
783956_C

Gartner

Recommendations: Improving Your ML Monitoring Maturity

To effectively monitor this system, technical professionals must put these pieces together.

Align ML monitoring with your MLOps maturity level: As the complexity and scale of your production ML environment increases, both the methods and sophistication of your monitoring must improve.

At low levels of MLOps maturity, collect cost, health and system telemetry:

- If you are in the proof-of-concept phase, try to get a model into production.
- If you have a few models in production, but your MLOps retraining and deployment and integration processes are still mostly manual, you are at a relatively lower level of MLOps maturity.

- At a low level of maturity, focus on collecting cost, health and system telemetry as well as version metadata from the different components of your architecture.
- Following the principle that “the first rule of doing ML is to build a non-ML heuristic,” figure out useful heuristics that you can develop to create a baseline against which you can compare model performance. These monitoring signals will be helpful to the data scientist and data engineer who are getting started. You may be able to adapt existing monitoring capabilities inside your AutoML platform, or leverage basic service capabilities from your cloud provider.

At medium levels of MLOps maturity, build out a complete monitoring portfolio of drifts and model metrics and KPIs using native and best-of-breed tools:

- If your MLOps maturity is in the middle, you have developed some pipelines and you have a large number of models with a robust versioning, and you are doing automated retraining.
- At this stage of maturity, many organizations look to make ML monitoring more comprehensive and also a native part of ML services that get deployed.
- Build sophisticated monitoring capabilities for inputs and outputs and training processes. These monitoring capabilities should support the ML engineer and MLOps engineer persona. You are likely to use the full spectrum of native cloud monitoring capabilities native capabilities in your third-party DSML/MLOps platform. For a list of different categories of vendors, see [Getting Started With Machine Learning Monitoring in the Cloud](#).

At high levels of MLOps maturity, focus on automation, standardization and scale:

- At a high degree of MLOps maturity, everything (training, deployment, data) is a CI/CD pipeline, and you have the full spectrum of MLOps components. You are also likely a logical feature store adopter, due to the architectural advantages of feature stores for data management for machine learning.
- Focus on implementing data and ML observability, ontologies that enable you to do dependency analysis
- Use a sophisticated distributed monitoring platform with a focus on usability and different user personas.

- AI observability should be baked into the process of operationalization as much as possible, so that these capabilities can scale.

Many organizations at a high level of scale have their own monitoring system built from open-source or homegrown components. However, organizations just entering this area of sophistication may find that buying a COTS solution for monitoring ML may be worthwhile to save them effort. These implementations often have a sophisticated feature store and a sophisticated data engineering/data observability toolset. Moreover, new personas such as model owners may require them.

Event Logging and Inference Latency

Log data is a key part of an ML monitoring strategy. Many organizations imagine collecting data from every possible source and storing it indefinitely. In practice, this approach tends to lead to failure because the cost of storing and managing this log data far outstrips its utility. Instead, focus on identifying and collecting high-value log data. In every log record, keep a unique identifier (UID) for the model, so you can recover a specific record later if you're trying to debug a specific model. OpenTelemetry offers a [Log Data Model](#) that can be used to standardize log data collection.

For example, Logstash is a widely used cloud-scale log ingestion tool. Part of the Elastic Stack, it has an extensive library of input plug-ins that allow it to accept many different types of logs, perform some transformation and forward them to a destination. (Refer to [Implementing an Enterprise Open-Source Machine Learning Stack](#) for additional information on open-source ML tools.) Organizations also often look at Splunk and the Amazon Web Services (AWS) implementation of Elastic Stack for logging.

Recommendations: Logging and Tracing Decisions

Log model data:

- Model metrics, such as:
 - Accuracy
 - Precision
 - Sensitivity
- Unique ID and version number of the model. A best practice for generating a unique model identifier is to use a GitHub SHA (see Note 3).

- Tag each model with a human-readable identifier.
- Unique ID and version of the inference server that is running the model.
- Model configuration changes:
 - For example, challenger to champion.
 - Identity of who made configuration.

Log infrastructure data:

- Inference latency:
 - Latency is the difference between response and arrival rate.
 - Log request arrival rate.
 - Log scoring response rate.
 - Log request arrival rate and scoring response rate as separate events so that you can distinguish between failure to score and lack of requests.
 - Measure inference latency distribution for champion, challengers and canary models; you may identify anomalies.

Log sample of serving data:

- Feature verification errors, such as:
 - Missing features.
 - Features out of range.
 - Incorrect feature type.

- Log group of features for a single request:
 - Unique ID and version of the inference server that loaded and ran the model.
 - Unique ID and version of the model that served the request.
 - KPI scores generated by the model for this request.
 - If available, the actual labels associated with the predicted scores.

Utilize monitoring tools to collect performance metrics:

Inference server performance:

- CPU utilization
- Memory utilization
- ML cache hit rate for the inference server
- Unique ID and version of the inference server
- Infrastructure server errors
- Configuration changes and identification of who made the changes

Adopt tracing capabilities to trace ML-enabled applications

Tracing is another important monitoring tool:

- A trace encapsulates a single operation within an end-to-end ML application. A trace is usually represented as a series of spans.
- Each traceable unit of work within the operation generates a span. This could include an API gateway, feature extraction code, ML inference server, postinference processing, etc.
- Distributed tracing is used in microservices-based or other distributed applications because a single operation may touch many services.
- Distributed tracing allows collection of timing and other metadata as requests pass through different ML services.

The next section describes how to get started with ML monitoring in these tiers to see how they might work in a real deployment.

Getting Started With Machine Learning Monitoring in the Cloud

To implement machine learning monitoring using three major cloud providers, you must first familiarize yourself with the available options. Table 1 offers a summary.

Table 1: Cloud Machine Learning Monitoring Options

(Enlarged table in Appendix)

Vendor/Solution	Model Quality	Data Drift	Model Bias/Explainability	Data Quality	Infrastructure
Amazon SageMaker	SageMaker Model Monitor	SageMaker Model Monitor	SageMaker Model Monitor/ SageMaker Clarify	SageMaker Model Monitor/ SageMaker Pipelines	Amazon CloudWatch
Microsoft Azure Machine Learning (ML)	Azure Machine Learning Model Monitoring**	Azure Machine Learning Model Monitoring**	Azure Machine Learning Model Monitoring**	Azure Machine Learning Model Monitoring**	Azure Monitor/ Azure Monitor Application Insights
Google Cloud Vertex AI	Vertex AI Model Monitoring	Vertex AI Model Monitoring	Vertex Explainable AI	Vertex AI Pipelines*	Google Cloud Platform (GCP) Cloud Monitoring
*Requires some work to create this monitoring capability.					
**Feature in preview at time of writing.					

Source: Gartner (July 2023)

Monitoring Machine Learning in Amazon Web Services

As listed in Table 1, AWS offers several services that are a useful starting point for implementing ML monitoring. SageMaker Model Monitor provides tools to identify drift in data quality, model quality, model bias and feature attribution drift. SageMaker Model Monitor also integrates with SageMaker Clarify to identify ML model bias and feature attribution drift (explainability).

Monitoring requires effort and several implementation steps. For organizations starting their ML monitoring journey in AWS, the five steps are described here.

Step 1: Enable Data Quality Monitoring

Enable monitoring during model training to develop a baseline of data quality. Data quality monitoring profiles the input data during model training, then continuously compares incoming data with the existing data profile.

Tool: SageMaker Model Monitor allows you to monitor training and retraining jobs and create a baseline of data quality.

Step 2: Monitor Data Drift

If there's a significant difference between the original data profile and the incoming data profile, then a drift in the input data is likely happening.

Tool: Amazon SageMaker Model Monitor offers a data drift monitoring task that continuously profiles the input data and compares it with the baseline, with results captured in CloudWatch, Amazon SageMaker Model Dashboard, as well as the SageMaker Model Monitor UI in SageMaker Studio. This can be enabled via Amazon SageMaker drift detection template for real-time deployment. This task runs on its own computation resources using Deequ.

Guidance: The task is designed to be lightweight, to ensure the monitoring job does not slow down your ML inference flow, and to scale with the data. However, this assumption may not hold as you scale. Also, consider how often you want to run this task, based on how much cost you wish to incur as well as how rapidly data is likely to change.

Step 3: Add Model Quality Monitoring

Rather than looking at the input data in model quality monitoring, you must capture actual values (aka ground truth data) that can be compared with the predictions.

Tool: The Model Quality Monitoring task computes model performance metrics from actuals and predicted values. To succeed with model quality monitoring, generate a baseline set of predictions from a validation dataset, and then collect predictions made by the endpoint and have actual data points (ground truth data) to compare to.

Guidance: Depending on the use case, it can be quite difficult to collect ground truth data/actual values in many scenarios. For example, how can you know whether a customer churn model's customer churn prediction is accurate (especially because you might be doing new things to prevent churn)? But for some simpler predictions, like demand forecasting, ground truth data may be available. Sometimes you need to get feedback or inputs from your business unit partners to get ground truth data, or something close to it.

Step 4: Use SageMaker Clarify Features in SageMaker Model Monitor

To get more insight and visibility into your training data and models (most organizations try to reach the point of having some model quality monitoring in place), use SageMaker Clarify in SageMaker Model Monitor. This is one additional step you can pursue, as a best practice, to support model explainability and fairness. This tool enables you to identify and limit bias and explain predictions.

You may want to monitor feature bias and drift. Trained models may consider some features more strongly than others when generating predictions. If different model versions dramatically change in the features that they weight to generate predictions, this can be a bias that can cause problems. Comparing feature importance and bias between model-provided versions can help you understand and explain your models better from the perspective of bias.

Tool: SageMaker Clarify detects model drift/model bias and feature attribution drift, and offers other visualizations for model explainability.

Guidance: Model explainability and fairness is important, as described in [Incorporate Explainability and Fairness Within the AI Platform](#). But because of the effort involved using native capabilities, you will want to use the document linked above to make sure the organization understands the value of AI explainability initiatives and uses them within development and business processes.

Monitoring Machine Learning in Microsoft Azure Machine Learning

Table 1 links to Microsoft Azure Machine Learning's monitoring capabilities. Note that Azure Machine Learning Model Monitoring was historically mostly done through Windows SDK, but now there is a feature available in preview, Azure Machine Learning Model Monitoring.

To enable Azure Machine Learning Model Monitoring, follow these steps:

Step 1: Enable Data Collection

- If you deploy a model to an Azure Machine Learning online endpoint, enable production inference data collection using Azure Machine Learning Model Data Collector.
- If you deploy a model outside of Azure Machine Learning, you are responsible for collecting production inference data via your own tooling.

Step 2: Set Up Model Monitoring

- Use the SDK/Command Line Interface (CLI) 2.0 or the studio UI to configure model monitoring. During the setup, specify the desired monitoring signals and metrics, as well as set alert thresholds.
- Familiarize yourself with the available native monitoring capabilities. For example, for data drift, Azure Machine Learning data drift detector offers the following capabilities:
 - Analyzes data drift in your data
 - Monitors new data for differences between training and serving datasets
 - Monitors any differences between baseline and target data, profile features in data to track change in statistical properties over time
 - Set up data drift alerts

Guidance:

- Collaborate with data scientists who are familiar with the model to set up model monitoring. Their expertise in the model and its use cases will help you recommend appropriate monitoring signals with the correct alerting thresholds.
- Initiate model monitoring immediately after deploying your model to production.
- Determine the monitoring frequency based on the growth of your production data over time. If your production model receives substantial daily traffic and the daily data accumulation is sufficient to be monitored, set the frequency to daily. Otherwise, consider weekly or monthly monitoring frequencies based on the data's growth.
- Use model training data as the baseline dataset for comparison. Azure Machine Learning allows you to choose recent past production data or historical data (such as training or validation data) for comparison. Gartner recommends using the training data as baseline for data drift and data quality comparisons, and validation data as the baseline for prediction drift.

Step 3: View and Analyze Model Monitoring Results

- Once you have configured model monitoring, a monitoring job is scheduled to run at the frequency you've specified.

- Each run computes and evaluates metrics for all selected monitoring signals, triggering alert notifications when any threshold is exceeded.
- Any alert notification will include a link that will take you to the Azure Machine Learning workspace to view and analyze results.

Guidance:

- Incorporate multiple monitoring signals into your setup to gain a comprehensive and detailed understanding of model performance. Combining data drift and feature attribution drift can help provide early warnings about potential issues. Granular insights can be obtained through signals like data drift cohort analysis for specific data segments.
- Monitor the topmost important features. Azure Machine Learning monitors data drift or data quality for the top 10 important features when using training data as a baseline. Monitoring a subset of features in models that have a large number of features will reduce computation cost and reduce monitoring noise.

Step 4: Enrich ML Monitoring With Explainability and System Health Monitoring

AzureML monitoring, via a preview feature, supports explainability via data drift layering over feature importance and feature attribution drift. With data drift layering over feature importance, clients can set up monitoring that can combine these two signals; for example, “Monitor top 5 features for data drift.” With feature attribution drift, customers can monitor feature importance drift over time. This signal can be valuable for identifying the occurrence of model drift. Microsoft has also released an open-source explainability toolkit, the [Responsible AI Toolbox \(GitHub\)](#).

Azure Machine Learning SDK for Python also provides a monitoring class, which contains functionality to collect model data, including inputs and predictions.

Guidance:

- Deploy additional services in the Azure ecosystem to enable broader system and application monitoring. Azure Machine Learning relies on Azure Monitor — a full-stack monitoring service in Azure — to monitor cloud services and infrastructure, and Azure Application Insights for monitoring web service endpoints.

- By releasing new monitoring capabilities this year, Azure Machine Learning is providing more options for ML monitoring via the platform than were previously available. However, as many of these features are in preview, they may not be ready for enterprise-grade production use cases, so proceed with caution.

Monitoring Machine Learning in Google Cloud Platform

As described in Table 1, Google Vertex AI model monitoring runs jobs periodically to monitor deployed models and detect abnormal behaviors. Capabilities include analyzing data distributions between baseline (or training) data and comparison datasets. The service monitors for data drift and training-serving skew when the feature data distributions used in production differ from the feature distributions of data used in training. Users can also monitor TensorBoards.

Step 1: Set Up Vertex AI Model Monitoring

Vertex AI Model Monitoring offers feature skew and drift detection for categorical and numerical input features.

With Model Monitoring enabled, prediction requests get logged by a BigQuery table in the customer's Google Cloud project. The input feature values are then analyzed for skew or drift.

Vertex AI Model Monitoring also has a preview feature that enables monitoring of batch predictions. It provides a simplified experience for collecting and comparing batch prediction input data and comparing it to training data.

Guidance:

- Vertex AI Model Monitoring can sometimes parse the schema of model input data automatically, for example, for models that have been created using AutoML. However, for some custom models, you may need to provide this schema when you set up a monitoring job.
- Initiate model monitoring immediately after deploying your model to production.
- To monitor more efficiently from a cost perspective, set a prediction request sampling rate to monitor a sample of the production inputs to a model.
- You can enable skew detection if you provide the original training dataset for your model; otherwise, you should enable drift detection.

- Understand how fresh you need this model to be, and how much data will be coming in, and how sensitive you want to set monitoring thresholds. If the model receives a lot of daily traffic, then set the monitoring frequency to be more frequent than the default. Similarly, the default is that every categorical and numerical feature is monitored, with threshold values of 0.3. But you may want to relax or tighten the thresholds, depending on the scenario.

Step 2: View and Analyze Model Monitoring Results

Vertex AI Model Monitoring enables you to select an endpoint and visualize the most recent results for monitoring jobs. You can also visualize the results of batch prediction monitoring.

Step 3: Enrich ML Monitoring With Explainability Monitoring

Vertex AI Model Monitoring includes integrations with Vertex Explainable AI to offer monitoring for feature distributions and feature attributions.

Guidance:

- In a stable machine learning system, features' relative importance generally remains relatively stable over time. If an important feature drops in importance, it might signal that something about that feature has changed. Thus, monitoring and tracking when these changes occur can be very valuable to identify silent failures, where the model keeps working, but it has degraded in some way.
- New monitoring capabilities for batch scoring improve the experience of monitoring for batch use cases. However, as these features are in preview, they may not be ready for enterprise-grade production use cases, so proceed with caution.

Third-Party Options for Monitoring Machine Learning

Cloud providers offer native capabilities, but for richer monitoring capabilities you may wish to look at dedicated monitoring vendors or open-source toolsets (see Table 2).

Table 2: Sample Third-Party Options for Monitoring Machine Learning

(Enlarged table in Appendix)

Type	Sample Vendors
Pure play: Vendors that are primarily focused on ML monitoring and explainability. Many of these vendors are also rolling out NLP and Computer Vision monitoring as well as Generative AI observability features.	Aporia Arize AI Arthur Census Fiddler Mona New Relic Qualdo Superwise TruEra Verta WhyLabs
Part of a broader DSML engineering platform: These vendors bundle monitoring capabilities but also offer a broader set of platform capabilities for MLOps/ModelOps	DataRobot Datatron Domino Data Labs H2O.ai Modzy Seldon
Open source: Open-source offerings in ML monitoring	ClearML Evidently MLWatcher Whylogs

Source: Gartner (July 2023)

Sophisticated monitoring capabilities from dedicated vendors are both important and necessary as organizations scale up and mature their MLOps initiatives. Although the cloud vendors offer native capabilities, there is a complicated balance between native and best-of-breed that organizations must strive to attain.

Too many best-of-breed tools are difficult to integrate, but native capabilities do not always provide the capability and consolidation needed. At first, organizations may find that this calculus leads different departments to pursue different monitoring solutions. Over time, ML architects should work to consolidate the portfolio into a series of clear options that balance cost and implementation effort.

Strengths

When designing ML monitoring systems, technical professionals should consider the following benefits. A robust ML monitoring system can:

- **Provides visibility into system operation, performance and failures.** Without this visibility, technical professionals are blind to model behavior and performance.

- **Responds to data anomalies:** such as upstream data changes, training-serving, skew, and so on. ML monitoring systems can detect failures, which are often related to drifts, and provide an indication as to how to fix the system. This will improve overall system reliability and availability.
- **Supports model retraining:** A robust ML monitoring system will help the process of training newer versions of models by logging scoring data that can be used for training. This provides a feedback loop for iterative model development and improvement. The monitoring system will also support the monitoring of models that are scoring live data but are not yet in production (e.g., challenger models).

Weaknesses

When designing ML monitoring systems, technical professionals should consider the following cautions. Tools and practices for monitoring and managing machine learning:

- **Continue to lag behind advancements in machine learning.** As large language models (LLMs) are democratizing computer vision (CV) and natural language programming (NLP), monitoring capabilities must catch up to handle these use cases. In many cases, different approaches to monitoring will be necessary. There are very few best practices for monitoring of LLM-based systems. Some of the components of an ML monitoring system can leverage common tools (e.g., logging systems), but NLP/CV monitoring will often require new tools.
- **Require a steep learning curve.** Someone who is not a data scientist involved in developing an ML monitoring system must invest the time to attain a rudimentary understanding of machine learning. Otherwise, they will lack the basic skills needed to understand the technical problem and to communicate with a data scientist. Conversely, a nondomain expert must invest the time to thoroughly understand the business problem to design and train models that provide a measurable and meaningful KPI gain.
- **Risk obsolescence.** The ML market is undergoing tremendous change. Model development tools are changing rapidly as the LLM/generative AI space is taking off. There is an ever-increasing availability of off-the-shelf models. Anything you build now runs the risk of requiring material changes in 18 to 24 months. On the other hand, like any rapidly evolving technology, you learn by doing (and sometimes failing).

- **Require complex integration.** ML inference servers can now run models in constrained, distributed and mobile environments, which increases the complexity of ML monitoring. The monitoring systems must also have visibility into many, possibly distributed, sources of scoring data.

Guidance

The focus of ML monitoring must be on maximizing business-oriented KPI gain. Failure to demonstrate a measurable business benefit is why so many ML projects fail.

With that said, one of the other reasons that ML projects fail is that they fail to account for and implement monitoring. The barriers to entry for monitoring ML models are lower than ever before. Most ML monitoring providers — cloud, third-party and open-source — offer solutions that are very low-cost or free. These offerings, available via open-source software, community editions or cloud-native services, enable practitioners to achieve more ML observability in a straightforward manner.

As technical professionals, it is important to prioritize problem definition at the beginning of ML projects, and monitoring the moment production deployment occurs.

The following information provides general guidance for developing an effective ML monitoring system:

- **Define KPIs that are meaningful and measurable:** Designing meaningful and measurable KPIs will require that you establish a cross-domain team that includes the data scientist, ML engineer, ML architect, SME, application architect and operations team. Plan to combine the knowledge of the data scientist (e.g., data science platforms, algorithms, metrics), SME (e.g., manufacturing process engineering, financial risk assessment), application architect (e.g., microservices, APIs, packaging) and operations team (e.g., storage, compute and networking systems). The ML monitoring system must measure KPI gain and guide the organization to using models that optimize KPI gain.

- **Detect anomalies and suggest next best action:** Resilient ML models can continue to operate and provide a score, even when data patterns change and thus mask an underlying problem. ML monitoring must detect when there is a data pattern change that affects KPI gain (e.g., feature or concept drift) and organizations should strive to refine their incident resolution capabilities so that model insights can lead to remediation more quickly. Most of the time, the next action will be to retrain a model or promote a challenger model. Design a monitoring system that gives you clues as to what to do next.
- **Design a monitoring system that supports ongoing model deployment:** Rarely will an organization train and deploy a model that is never retrained or replaced. The opposite is more likely to be true. Therefore, design your ML monitoring system to support a process where there is continuous model deployment. The design must include the ability to simultaneously support many models that are in various stages of operation. The design must also be able to support the retraining process by collecting data for model training.
- **Support multiple roles:** ML monitoring involves the collaboration of cross-domain expertise and therefore must be designed to support multiple roles. Designing an effective ML monitoring system will require expertise that includes the data scientist, subject matter expert, application architect, data engineer and operations team. Plan to combine the knowledge of the data scientist (e.g., data science platforms, algorithms, metrics), SME (e.g., manufacturing process engineering, financial risk assessment), application architect (e.g., microservices, APIs, packaging) and operations team (e.g., storage, compute and networking systems) to design and deploy your ML monitoring system.
- **Architect an agile monitoring system that can adapt to ML and monitoring technology changes:** ML monitoring technology is maturing, but the ML space is evolving more quickly than the MLOps capabilities that are available. A convergence of complementary technology trends (e.g., generative AI, composite AI, edge computing, distributed cloud, hyperautomation) will force rapid evolution in the design, deployment and operation of ML monitoring systems. Design a flexible system that can adapt to changes.

Evidence

¹ [Matchmaker: Data Drift Mitigation in Machine Learning for Large-Scale Systems](#), Microsoft.

² [Productionizing Machine Learning: From Deployment to Drift Detection](#), Databricks.

This blog post defines three types of machine learning drift and discusses how to detect and address model drift.

³ [Instrumentation, Observability & Monitoring of Machine Learning Models](#), InfoQ.

Transcript of a talk by Josh Willis, a Slack software engineer, at [QCon](#). The talk provides detailed advice based on Willis' real-world experience.

⁴ [Data Validation for Machine Learning](#), MLSys. Proceedings of the second SysML

Conference, 2019. This paper presents a data validation system that is designed to detect anomalies specifically in data fed into machine learning pipelines. This system is deployed in production at Google. It is used by hundreds of product teams to continuously monitor and validate several petabytes of production data per day.

⁵ [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt](#)

[Reduction](#), Google Research. Eric Breck and others, IEEE, 2017. The paper presents a rubric for ML production readiness. It provides a set of 28 actionable tests and offers a scoring system to measure the readiness of an ML system for production. This paper details actionable advice drawn from practice and verified with extensive interviews with the maintainers of 36 real-world systems.

⁶ [Automating Large-Scale Data Quality Verification](#), VLDB Endowment. This paper

presents a system for automating the verification of data quality at scale for production deployments. The system provides a declarative API, which combines common quality constraints with user defined validation code, and thereby enables "unit tests" for data.

⁷ Google researchers have published widely on the topic of skew. Two examples of

Google researchers' work are [Data Validation for Machine Learning](#) and [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction](#).

⁸ [Automating Large-Scale Data Quality Verification](#), VLDB Endowment. This paper

presents a system for automating the verification of data quality at scale for production deployments.

⁹ [Concept Drift and Model Decay in Machine Learning](#), Data Exploration. This detailed

and technical blog explains concept drift and provides code snippets to simulate concept drift.

¹⁰ [The Problem of Concept Drift: Definitions and Related Work](#), TU Dortmund University. This paper considers different types of concept drift, peculiarities of the problem, and gives a critical review of existing approaches to the problem.

¹¹ [How to Detect Drifting Models](#), RapidMiner. The concept of model drift is difficult to describe. The RapidMiner blog nicely describes this concept using sentiment analysis of the word “terrific.” This approach is used in this note.

¹² [A Gentle Introduction to Concept Drift in Machine Learning](#), Machine Learning Mastery. Brownlee defines concept drift and describes seven actions to deal with concept drift.

Note 1: MLOps

MLOps enables the operationalization of the end-to-end pipeline that supports the continuous delivery and continuous integration of models in a production environment. Core capabilities include feature curation, feature management (store), model governance (in some cases through ModelOps), model release, activation, monitoring, performance tracking, management, logging, reuse and maintenance.

Note 2: ModelOps

ModelOps is focused primarily on the governance and life cycle management of a wide range of operationalized AI and decision models (including machine learning, knowledge graphs, rules, optimization, linguistic and agent-based models). Core capabilities include CI/CD integration, model development environments, champion-challenger testing, model versioning, model store and rollback.

ModelOps also enables the governance and procedures for re-tuning, retraining or rebuilding of AI models; aimed at providing an uninterrupted flow between the development, operationalization and full maintenance of AI models. ModelOps provides business domain experts autonomy to assess the quality (interpret the outcomes and validate KPIs) of AI models in production. MLOps also facilitates the ability to promote or demote AI models for inferencing without a full dependency on data scientists or ML engineers.

Note 3: GitHub SHA

A GitHub SHA is a unique hash that is created every time a change is committed to a GitHub repository. It allows you to keep a record of what changes were made, when the change occurred and who made the change.

Note 4: Classification Metrics

Before delving into model metrics, it is important to understand the difference between a classification model and a regression model because the model metrics are different for each type of model. A classification model accepts input data and predicts the output class whereas a linear regression model produces a numerical answer.

For example, let's say that a bank receives thousands of mortgage applications per year. It needs an automated way to determine which applicants it should accept and which it should not. It decides to create an ML model to help make this decision. Based upon the input data (e.g., FICO score, loan amount, employment status, current debt, etc.) the model classifies the applications into three groups "qualifies for credit," "does not qualify for credit" and "unsure." The model it created was a classification model.

A common way to analyze classification models is to use a [Confusion Matrix](#). A confusion matrix, also known as an error matrix, is a table that visualizes the performance of a classification algorithm (see Figure 12). The matrix is typically a table with two rows and two columns that report the number of false positives, false negatives, true positives and true negatives. The confusion matrix provides many important derivative metrics, such as:

- Accuracy: The proportion of the total number of predictions that were correct.
$$\text{Accuracy} = (TP + TN)/n$$
- Precision: The proportion of positive predictions that were correct. In other words, when the classifier predicts YES, how often is it right?
$$\text{Precision} = TP/(TP + FP)$$
- Recall (aka sensitivity): The proportion of actual YES samples that were predicted YES. In other words, when the classifier predicts YES, what proportion of all the actual YES samples did it predict?
$$\text{Recall/Sensitivity} = TP/(TP + FN)$$
- Specificity: The proportion of actual NO samples that were predicted NO. In other words, when the classifier predicts NO, what proportion of all the actual NO samples did it predict?
$$\text{Specificity} = TN/(FP + TN)$$
- F1 Score: In some applications, it might be important to have high precision and high recall. For example, a model that classifies brain scan images as "cancerous" or "not cancerous" must be correct (i.e., precise) and identify all the cancerous images (i.e., high recall/sensitivity). The F1 Score is the harmonic mean of precision and recall.
$$\text{F1 Score} = 2 * \text{Precision} * \text{Recall}/(\text{Precision} + \text{Recall})$$

Figure 12: Confusion Matrix

Confusion Matrix



Source: Gartner
451229_C

Note 5: Linear Regression Metrics

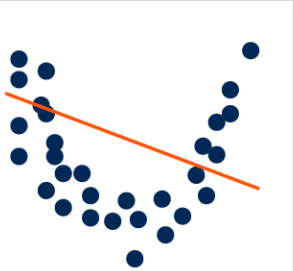
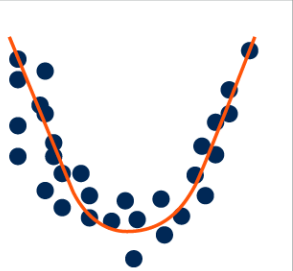
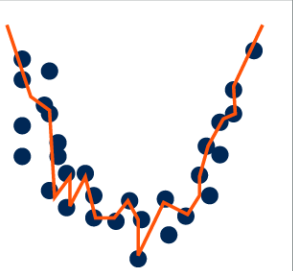
When your machine learning model must make predictions that do not fit into discrete classes, then you will need a linear regression model. ¹² A linear regression model produces a numerical answer rather than predicting a class.

For instance, let’s say a real estate agency needs to develop a model that predicts housing prices. The input to the model includes features such as number of bedrooms, number of bathrooms, living space area, size of yard, ZIP Code or postal code, and so on. The output of the model is the predicted sale price of the home. The model they created was a linear regression model.

The goal of training a linear regression model is to predict a numerical value by fitting the output of the model (i.e., the scores) to the training data curve. To do this training, data scientists will often use the root-mean-square error (RSME) to evaluate their linear regression models. The RMSE measures the standard deviation of the prediction errors (aka residuals) compared to the actual values observed. The data scientist will train the model to fit the curve of the predicted values (e.g., predicted home price) to the actual values (e.g., actual home values). The data scientist attempts to fit that model to the training data without underfitting or overfitting (see Figure 13).

Figure 13: Linear Regression: Underfitting and Overfitting

Linear Regression: Underfitting and Overfitting

	Underfitting	Just Right	Overfitting
Symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error• High bias	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Very low training error• Training error much lower than test error• High variance
Regression Illustration			

Source: Adapted From Stanford University
783956_C

For more information, refer to the following blog posts:

- [The Proper Way to Use Machine Learning Metrics](#), Medium.
- [Complete Guide to Machine Learning Evaluation Metrics](#), Medium.
- [12 Important Model Evaluation Metrics for Machine Learning Everyone Should Know](#), Analytics Vidhya.
- [Metrics to Evaluate Your Machine Learning Algorithm](#), Medium.
- [Lift Analysis — A Data Scientist’s Secret Weapon](#), KDnuggets.

- [20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics](#), Medium.
- [How to Monitor Machine Learning Models in Real Time](#), KDnuggets.

Document Revision History

[Getting Started With Machine Learning Monitoring in Production - 11 June 2020](#)

Recommended by the Author

Some documents may not be available as part of your current Gartner subscription.

[How to Securely Design and Operate Machine Learning](#)

[Roles and Skills to Support Advanced Analytics and AI Initiatives](#)

[Toolkit: Delivery Metrics for DataOps, Self-Service Analytics, ModelOps and MLOps](#)

[The Logical Feature Store: Data Management for Machine Learning](#)

[Machine Learning Playbook for Data and Analytics Professionals](#)

[Incorporate, Test, Deploy and Maintain Machine Learning Models in Production Applications](#)

[A Guidance Framework for Operationalizing Machine Learning](#)

© 2023 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates. This publication may not be reproduced or distributed in any form without Gartner's prior written permission. It consists of the opinions of Gartner's research organization, which should not be construed as statements of fact. While the information contained in this publication has been obtained from sources believed to be reliable, Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information. Although Gartner research may address legal and financial issues, Gartner does not provide legal or investment advice and its research should not be construed or used as such. Your access and use of this publication are governed by [Gartner's Usage Policy](#). Gartner prides itself on its reputation for independence and objectivity. Its research is produced independently by its research organization without input or influence from any third party. For further information, see "[Guiding Principles on Independence and Objectivity](#)." Gartner research may not be used as input into or for the training or development of generative artificial intelligence, machine learning, algorithms, software, or related technologies.

Table 1: Cloud Machine Learning Monitoring Options

Vendor/Solution	Model Quality	Data Drift	Model Bias/Explainability	Data Quality	Infrastructure
Amazon SageMaker	SageMaker Model Monitor	SageMaker Model Monitor	SageMaker Model Monitor/ SageMaker Clarify	SageMaker Model Monitor/ SageMaker Pipelines	Amazon CloudWatch
Microsoft Azure Machine Learning (ML)	Azure Machine Learning Model Monitoring**	Azure Machine Learning Model Monitoring**	Azure Machine Learning Model Monitoring**	Azure Machine Learning Model Monitoring**	Azure Monitor/ Azure Monitor Application Insights
Google Cloud Vertex AI	Vertex AI Model Monitoring	Vertex AI Model Monitoring	Vertex Explainable AI	Vertex AI Pipelines*	Google Cloud Platform (GCP) Cloud Monitoring
*Requires some work to create this monitoring capability.					
**Feature in preview at time of writing.					

Source: Gartner (July 2023)

Table 2: Sample Third-Party Options for Monitoring Machine Learning

Type	Sample Vendors
Pure play: Vendors that are primarily focused on ML monitoring and explainability. Many of these vendors are also rolling out NLP and Computer Vision monitoring as well as Generative AI observability features.	Aporia Arize AI Arthur Censius Fiddler Mona New Relic Qualdo Superwise TruEra Verta WhyLabs
Part of a broader DSML engineering platform: These vendors bundle monitoring capabilities but also offer a broader set of platform capabilities for MLOps/ModelOps	DataRobot Datatron Domino Data Labs H2O.ai Modzy Seldon
Open source: Open-source offerings in ML monitoring	ClearML Evidently MLWatcher Whylogs

Source: Gartner (July 2023)