Google Cloud

White paper
May 2021

Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning.

Authors:
Khalid Salama,
Jarek Kazmierczak,
Donna Schut

Table of Contents

Executive summary 3

Overview of MLOps lifecycle and core capabilities 4

Building an ML-enabled system 6
The MLOps lifecycle 7
MLOps: An end-to-end workflow 8
MLOps capabilities 9
- Experimentation 11
- Data processing 11
- Model training 11
- Model evaluation 12
- Model serving 12
- Online experimentation 13
- Model monitoring 13
- ML pipelines 13
- Model registry 14
- Dataset and feature repository 14
- ML metadata and artifact tracking 15

Deep dive of MLOps processes 15

ML development 16
Training operationalization 18
Continuous training 20
Model deployment 23
Prediction serving 25
Continuous monitoring 26
Data and model management 29
- Dataset and feature management 29
  - Feature management 30
  - Dataset management 31
- Model management 32
  - ML metadata tracking 32
  - Model governance 33

Putting it all together 34

Additional resources 36

Executive summary

Across industries, DevOps and DataOps have been widely adopted as methodologies to improve quality and reduce the time to market of software engineering and data engineering initiatives. With the rapid growth in machine learning (ML) systems, similar approaches need to be developed in the context of ML engineering, which handle the unique complexities of the practical applications of ML. This is the domain of MLOps. MLOps is a set of standardized processes and technology capabilities for building, deploying, and operationalizing ML systems rapidly and reliably.]

We previously published Google Cloud's AI Adoption Framework to provide guidance for technology leaders who want to build an effective artificial intelligence (AI) capability in order to transform their business. That framework covers AI challenges around people, data, technology, and process, structured in six different themes: *learn, lead, access, secure, scale,* and *automate*.

The current document takes a deeper dive into the themes of *scale* and *automate* to illustrate the requirements for building and operationalizing ML systems. *Scale* concerns the extent to which you use cloud managed ML services that scale with large amounts of data and large numbers of data processing and ML jobs, with reduced operational overhead. *Automate* concerns the extent to which you are able to deploy, execute, and operate technology for data processing and ML pipelines in production efficiently, frequently, and reliably.

We outline an MLOps framework that defines core processes and technical capabilities. Organizations can use this framework to help establish mature MLOps practices for building and operationalizing ML systems. Adopting the framework can help organizations improve collaboration between teams, improve the reliability and scalability of ML systems, and shorten development cycle times. These benefits in turn drive innovation and help gain overall business value from investments in ML.

This document is intended for technology leaders and enterprise architects who want to understand MLOps. It's also for teams who want details about what MLOps looks like in practice. The document assumes that readers are familiar with basic machine learning concepts and with development and deployment practices such as CI/CD.

The document is in two parts. The first part, an overview of the MLOps lifecycle, is for all readers. It introduces MLOps processes and capabilities and why they're important for successful adoption of ML-based systems.

The second part is a deep dive on the MLOps processes and capabilities. This part is for readers who want to understand the concrete details of tasks like running a continuous training pipeline, deploying a model, and monitoring predictive performance of an ML model.

Organizations can use the framework to identify gaps in building an integrated ML platform and to focus on the scale and automate themes from Google's AI Adoption Framework. The decision about whether (or to which degree) to adopt each of these processes and capabilities in your organization depends on your business context. For example, you must determine the business value that the framework creates when compared to the cost of purchasing or building capabilities (for example, the cost in engineering hours).

Overview of MLOps lifecycle and core capabilities

Despite the growing recognition of AI/ML as a crucial pillar of digital transformation, successful deployments and effective operations are a bottleneck for getting value from AI. Only one in two organizations has moved beyond pilots and proofs of concept. Moreover, 72% of a cohort of organizations that began AI pilots before 2019 have not been able to deploy even a single application in production.[1] Algorithmia's survey of the state of enterprise machine learning found that 55% of companies surveyed have not deployed an ML model.[2] To summarize: models don't make it into production, and if they do, they break because they fail to adapt to changes in the environment.

This is due to a variety of issues. Teams engage in a high degree of manual and one-off work. They do not have reusable or reproducible components, and their processes involve difficulties in handoffs between data scientists and IT. Deloitte identified lack of talent and integration issues as factors that can stall or derail AI initiatives.[3] Algorithmia's survey highlighted that challenges in deployment, scaling, and versioning efforts still hinder teams from getting value from their investments in ML. Capgemini Research noted that the top three challenges faced by organizations in achieving deployments at scale are lack of mid- to senior-level talent, lack of change-management processes, and lack of strong governance models for achieving scale.

The common theme in these and other studies is that ML systems cannot be built in an ad hoc manner, isolated from other IT initiatives like DataOps and DevOps. They also cannot be built without adopting and applying sound software engineering practices, while taking into account the factors that make operationalizing ML different from operationalizing other types of software.

Organizations need an automated and streamlined ML process. This process does not just help the organization successfully deploy ML models in production. It also helps manage risk when organizations scale the number of ML applications to more use cases in changing environments, and it helps ensure that the applications are still in line with business goals. McKinsey's Global Survey on AI found that having standard frameworks and development

[1] The AI-powered enterprise, CapGemini Research Institute, 2020.
[2] 2020 state of enterprise machine learning, Algorithmia, 2020.
[3] Artificial intelligence for the real world, Deloitte, 2017.
[4] The state of AI in 2020, McKinsey, 2020.

processes in place is one of the differentiating factors of high-performing ML teams.[4]

This is where ML engineering can be essential. ML engineering is at the center of building ML-enabled systems, which concerns the development and operationalizing of production-grade ML systems. ML engineering provides a superset of the discipline of software engineering that handles the unique complexities of the practical applications of ML.[5] These complexities include the following:

- Preparing and maintaining high-quality data for training ML models.
- Tracking models in production to detect performance degradation.
- Performing ongoing experimentation of new data sources, ML algorithms, and hyperparameters, and then tracking these experiments.
- Maintaining the veracity of models by continuously retraining them on fresh data.
- Avoiding training-serving skews that are due to inconsistencies in data and in runtime dependencies between training environments and serving environments.
- Handling concerns about model fairness and adversarial attacks.

MLOps is a methodology for ML engineering that unifies ML system development (the ML element) with ML system operations (the Ops element). It advocates formalizing and (when beneficial) automating critical steps of ML system construction. MLOps provides a set of standardized processes and technology capabilities for building, deploying, and operationalizing ML systems rapidly and reliably.

MLOps supports ML development and deployment in the way that DevOps and DataOps support application engineering and data engineering (analytics). The difference is that when you deploy a web service, you care about resilience, queries per second, load balancing, and so on. When you deploy an ML model, you also need to worry about changes in the data, changes in the model, users trying to game the system, and so on. This is what MLOps is about.

MLOps practices can result in the following benefits over systems that do not follow MLOps practices:

- Shorter development cycles, and as a result, shorter time to market.
- Better collaboration between teams.
- Increased reliability, performance, scalability, and security of ML systems.
- Streamlined operational and governance processes.
- Increased return on investment of ML projects.

In this section, you learn about the MLOps lifecycle and workflow, and about the individual capabilities that are re-

[5] Towards ML Engineering, Google, 2020.

quired for a robust MLOps implementation.

Building an ML-enabled system

Building an ML-enabled system is a multifaceted undertaking that combines data engineering, ML engineering, and application engineering tasks, as shown in figure 1.

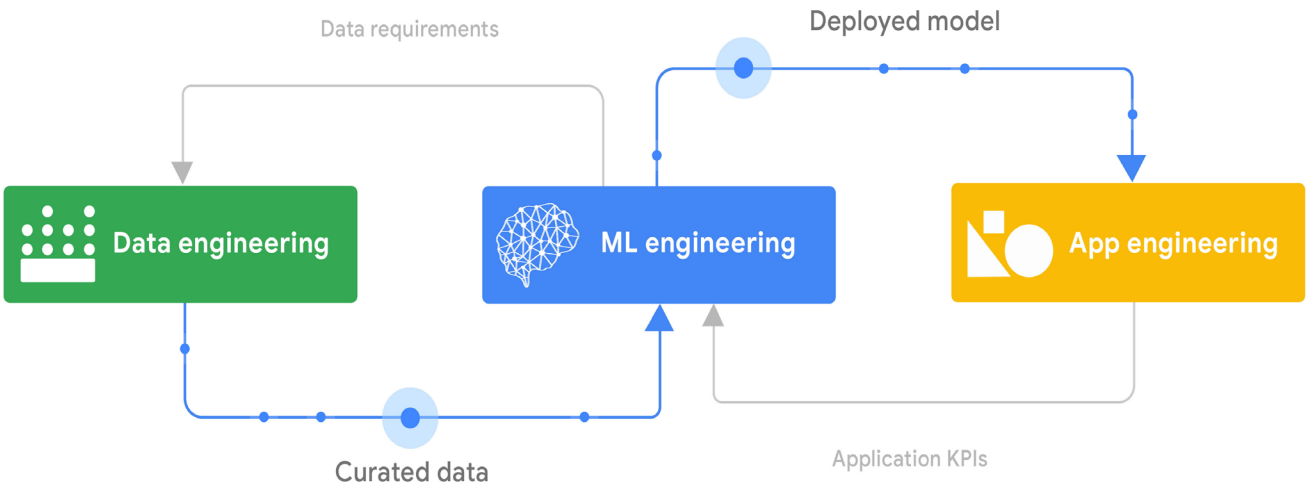

Figure 1. The relationship of data engineering, ML engineering, and app engineering

Data engineering involves ingesting, integrating, curating, and refining data to facilitate a broad spectrum of operational tasks, data analytics tasks, and ML tasks. Data engineering can be crucial to the success of the analytics and ML initiatives. If an organization does not have robust data engineering processes and technologies, it might not be set up for success with downstream business intelligence, advanced analytics, or ML projects.

ML models are built and deployed in production using curated data that is usually created by the data engineering team. The models do not operate in silos; they are components of, and support, a large range of application systems, such as business intelligence systems, line of business applications, process control systems, and embedded systems. Integrating an ML model into an application is a critical task that involves making sure first that the deployed model is used effectively by the applications, and then monitoring model performance. In addition to this, you should also collect and monitor relevant business KPIs (for example, click-through rate, revenue uplift, and user experience). This information helps you understand the impact of the ML model on the business and adapt accordingly.

The MLOps lifecycle

The MLOps lifecycle encompasses seven integrated and iterative processes, as shown in figure 2.

Figure 2. The MLOps lifecycle

The processes can consist of the following:

- **ML development** concerns experimenting and developing a robust and reproducible model training procedure (training pipeline code), which consists of multiple tasks from data preparation and transformation to model training and evaluation.
- **Training operationalization** concerns automating the process of packaging, testing, and deploying repeatable and reliable training pipelines.
- **Continuous training** concerns repeatedly executing the training pipeline in response to new data or to code changes, or on a schedule, potentially with new training settings.
- **Model deployment** concerns packaging, testing, and deploying a model to a serving environment for online experimentation and production serving.

- **Prediction serving** is about serving the model that is deployed in production for inference.
- **Continuous monitoring** is about monitoring the effectiveness and efficiency of a deployed model.
- **Data and model management** is a central, cross-cutting function for governing ML artifacts to support auditability, traceability, and compliance. Data and model management can also promote shareability, reusability, and discoverability of ML assets.

MLOps: An end-to-end workflow

Figure 3 shows a simplified but canonical flow for how the MLOps processes interact with each other, focusing on high-level flow of control and on key inputs and outputs.

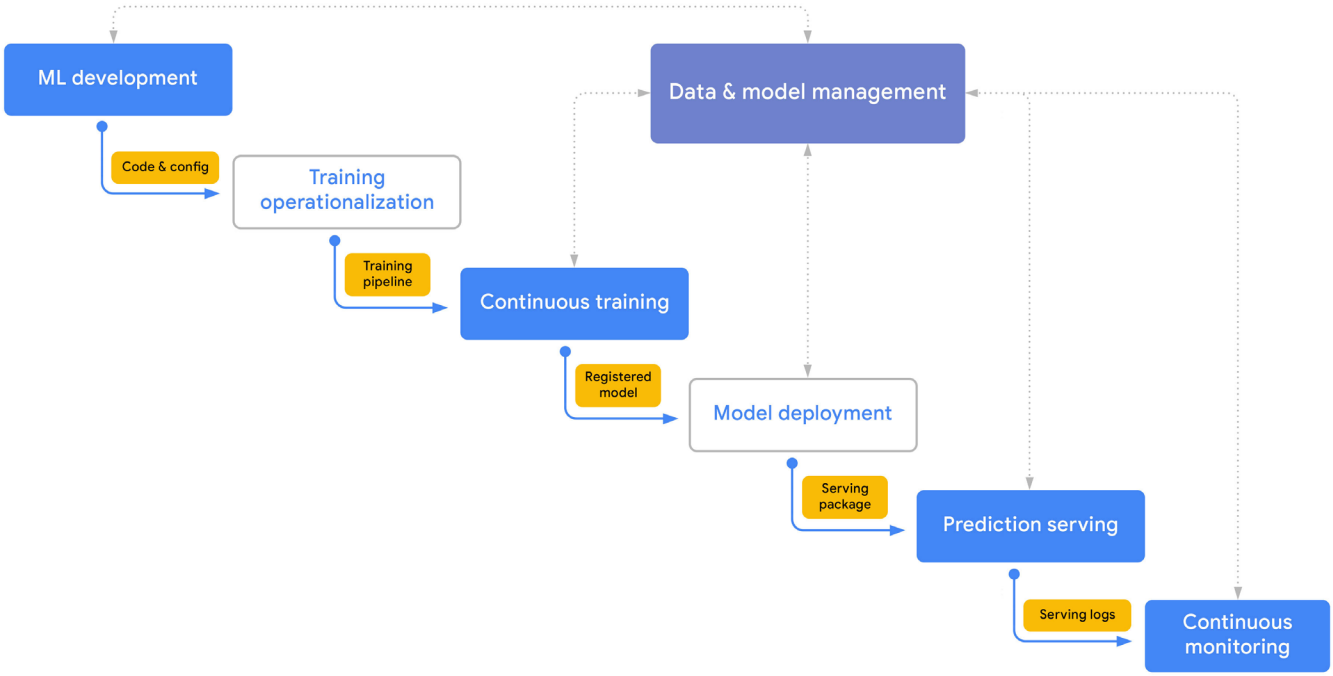

Figure 3. The MLOps process

This is not a waterfall workflow that has to sequentially pass through all the processes. The processes can be skipped, or the flow can repeat a given phase or a subsequence of the processes. The diagram shows the following flow:

1. The core activity during this ML development phase is experimentation. As data scientists and ML researchers prototype model architectures and training routines, they create labeled datasets, and they use features and other reusable ML artifacts that are governed through the data and model management process. The

primary output of this process is a formalized training procedure, which includes data preprocessing, model architecture, and model training settings.

2. If the ML system requires continuous training (repeated retraining of the model), the training procedure is operationalized as a training pipeline. This requires a CI/CD routine to build, test, and deploy the pipeline to the target execution environment.
3. The continuous training pipeline is executed repeatedly based on retraining triggers, and it produces a model as output. The model is retrained as new data becomes available, or if model performance decay is detected. Other training artifacts and metadata that are produced by a training pipeline are also tracked. If the pipeline produces a successful model candidate, that candidate is then tracked by the model management process as a registered model.
4. The registered model is annotated, reviewed, and approved for release and is then deployed to a production environment. This process might be relatively opaque if you are using a no-code solution, or it can involve building a custom CI/CD pipeline for progressive delivery.
5. The deployed model serves predictions using the deployment pattern that you have specified: online, batch, or streaming predictions. In addition to serving predictions, the serving runtime can generate model explanations and capture serving logs to be used by the continuous monitoring process.
6. The continuous monitoring process monitors the model for predictive effectiveness and service. The primary concern of effectiveness performance monitoring is detecting model decay—for example, data and concept drift. The model deployment can also be monitored for efficiency metrics like latency, throughput, hardware resource utilization, and execution errors.

MLOps capabilities

To effectively implement the key MLOps processes outlined in the previous section, organizations need to establish a set of core technical capabilities. These capabilities can be provided by a single integrated ML platform. Alternatively, they can be created by combining vendor tools that each are best suited to particular tasks, developed as custom services, or created as a combination of these approaches.

In most cases, the processes are deployed in stages rather than all at once in a single deployment. An organization's plan for adopting these processes and capabilities should align with business priorities and with the organization's technical and skills maturity. For example, many organizations start by focusing on the processes for ML development, model deployment, and prediction serving. For these organizations, continuous training and continuous monitoring might not be necessary if they are piloting a relatively small number of ML systems.

Figure 4 shows the core set of technical capabilities that are generally required for MLOps. They are abstracted as functional components that can have many-to-many mappings to specific products and technologies.

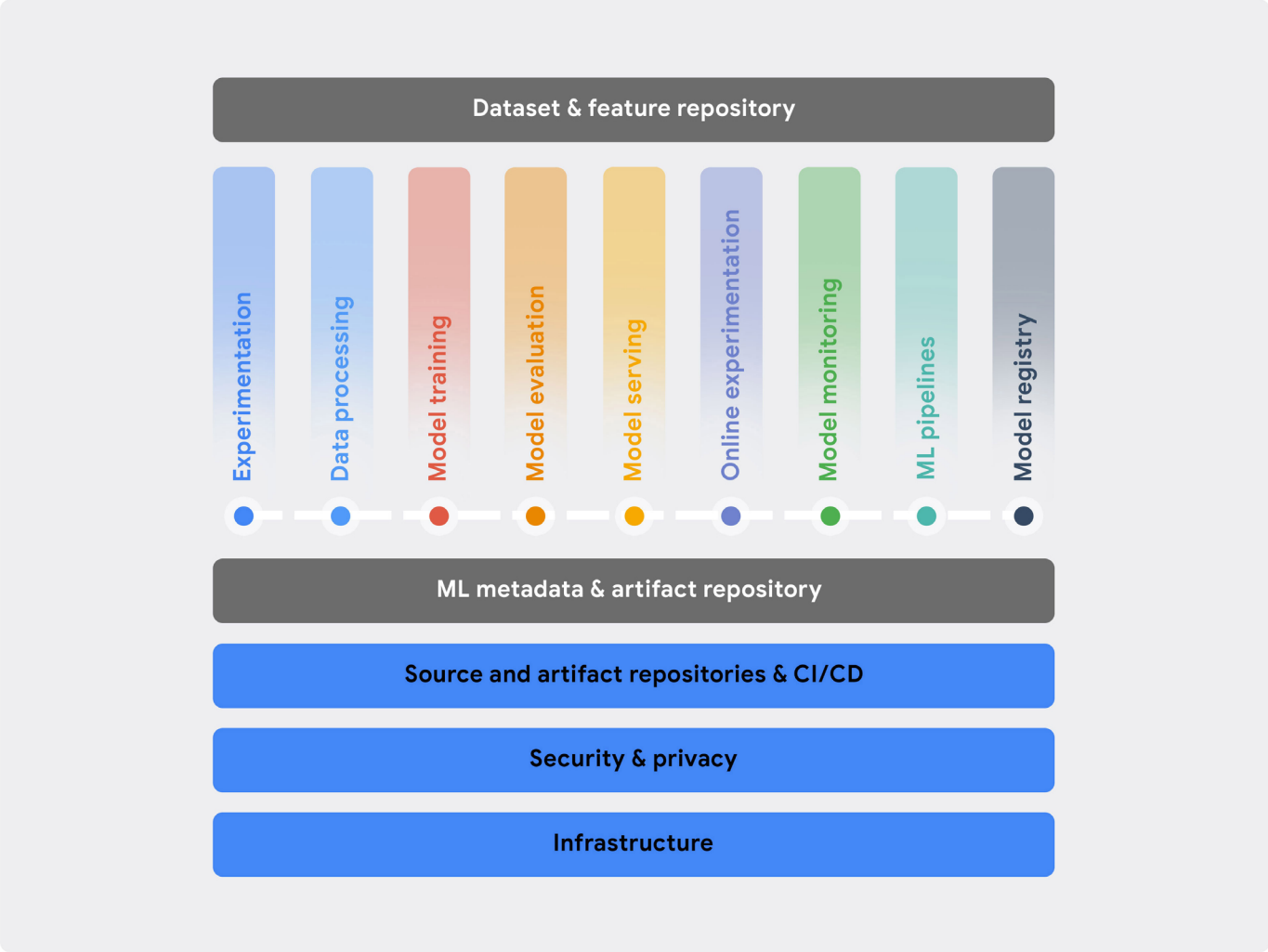

Figure 4. Core MLOps technical capabilities

Some foundational capabilities are required in order to support any IT workload, such as a reliable, scalable, and secure compute infrastructure. Most organizations already have investments in these capabilities and can benefit by taking advantage of them for ML workflows. Such capabilities might span multiple clouds, or even operate partially on-premises. Ideally, this would include advanced capabilities such as specialized ML accelerators.

In addition, an organization needs standardized configuration management and CI/CD capabilities to build, test, release, and operate software systems rapidly and reliably, including ML systems.

On top of these foundational capabilities is a set of core MLOps capabilities. These include experimentation, data processing, model training, model evaluation, model serving, online experimentation, model monitoring, ML pipeline, and model registry. Finally, two cross-cutting capabilities that enable integration and interaction are an ML metadata and artifact repository and an ML dataset and feature repository.

The following sections outline the characteristics of each of the MLOps capabilities.

Experimentation

The experimentation capability lets your data scientists and ML researchers collaboratively perform exploratory data analysis, create prototype model architectures, and implement training routines. An ML environment should also let them write modular, reusable, and testable source code that is version controlled. Key functionalities in experimentation include the following:

- Provide notebook environments that are integrated with version control tools like Git.
- Track experiments, including information about the data, hyperparameters, and evaluation metrics for reproducibility and comparison.
- Analyze and visualize data and models.
- Support exploring datasets, finding experiments, and reviewing implementations.
- Integrate with other data services and ML services in your platform.

Data processing

The data processing capability lets you prepare and transform large amounts of data for ML at scale in ML development, in continuous training pipelines, and in prediction serving. Key functionalities in data processing include the following:

- Support interactive execution (for example, from notebooks) for quick experimentation and for long-running jobs in production.
- Provide data connectors to a wide range of data sources and services, as well as data encoders and decoders for various data structures and formats.
- Provide both rich and efficient data transformations and ML feature engineering for structured (tabular) and unstructured data (text, image, and so on).
- Support scalable batch and stream data processing for ML training and serving workloads.

Model training

The model training capability lets you efficiently and cost-effectively run powerful algorithms for training ML models.

Model training should be able to scale with the size of both the models and the datasets that are used for training. Key functionalities in model training include the following:

- Support common ML frameworks and support custom runtime environments.
- Support large-scale distributed training with different strategies for multiple GPUs and multiple workers.
- Enable on-demand use of ML accelerators.
- Allow efficient hyperparameter tuning and target optimization at scale.
- Ideally, provide built-in automated ML (AutoML) functionality, including automated feature selection and engineering as well as automated model architecture search and selection.

Model evaluation

The model evaluation capability lets you assess the effectiveness of your model, interactively during experimentation and automatically in production. Key functionalities in model evaluation include the following:

- Perform batch scoring of your models on evaluation datasets at scale.
- Compute pre-defined or custom evaluation metrics for your model on different slices of the data.
- Track trained-model predictive performance across different continuous-training executions.
- Visualize and compare performances of different models.
- Provide tools for what-if analysis and for identifying bias and fairness issues.
- Enable model behavior interpretation using various explainable AI techniques.

Model serving

The model serving capability lets you deploy and serve your models in production environments. Key functionalities in model serving include the following:

- Provide support for low-latency, near-real-time (online) prediction and high-throughput batch (offline) prediction.
- Provide built-in support for common ML serving frameworks (for example, TensorFlow Serving, TorchServe, Nvidia Triton, and others for Scikit-learn and XGBoost models) and for custom runtime environments.
- Enable composite prediction routines, where multiple models are invoked hierarchically or simultaneously before the results are aggregated, in addition to any required pre- or post-processing routines.
- Allow efficient use of ML inference accelerators with autoscaling to match spiky workloads and to balance

cost with latency.
- Support model explainability using techniques like feature attributions for a given model prediction.
- Support logging of prediction serving requests and responses for analysis.

Online experimentation

The online experimentation capability lets you understand how newly trained models perform in production settings compared to the current models (if any) before you release the new model to production. For example, using a small subset of the serving population, you use online experimentation to understand the impact that a new recommendation system has on click-throughs and on conversation rates. The results of online experimentation should be integrated with the model registry capability to facilitate the decision about releasing the model to production. Online experimentation enhances the reliability of your ML releases by helping you decide to discard ill-performing models and to promote well-performing ones. Key functionalities in online experimentation include the following:

- Support canary and shadow deployments.
- Support traffic splitting and A/B tests.
- Support multi-armed bandit (MAB) tests.

Model monitoring

The model monitoring capability lets you track the efficiency and effectiveness of the deployed models in production to ensure predictive quality and business continuity. This capability informs you if your models are stale and need to be investigated and updated. Key functionalities in model monitoring include the following:

- Measure model efficiency metrics like latency and serving-resource utilization.
- Detect data skews, including schema anomalies and data and concept shifts and drifts.
- Integrate monitoring with the model evaluation capability for continuously assessing the effectiveness performance of the deployed model when ground truth labels are available.

ML pipelines

The ML pipelines capability lets you instrument, orchestrate, and automate complex ML training and prediction pipe-

lines in production. ML workflows coordinate different components, where each component performs a specific task in the pipeline. Key functionalities in ML pipelines include the following:

- Trigger pipelines on demand, on a schedule, or in response to specified events.
- Enable local interactive execution for debugging during ML development.
- Integrate with the ML metadata tracking capability to capture pipeline execution parameters and to produce artifacts.
- Provide a set of built-in components for common ML tasks and also allow custom components.
- Run on different environments, including local machines and scalable cloud platforms.
- Optionally, provide GUI-based tools for designing and building pipelines.

Model registry

The model registry capability lets you govern the lifecycle of the ML models in a central repository. This ensures the quality of the production models and enables model discovery. Key functionalities in the model registry include the following:

- Register, organize, track, and version your trained and deployed ML models.
- Store model metadata and runtime dependencies for deployability.
- Maintain model documentation and reporting—for example, using model cards.
- Integrate with the model evaluation and deployment capability and track online and offline evaluation metrics for the models.
- Govern the model launching process: review, approve, release, and roll back. These decisions are based on a number of offline performance and fairness metrics and on online experimentation results.

Dataset and feature repository

The dataset and feature repository capability lets you unify the definition and the storage of the ML data assets. Having a central repository of fresh, high-quality data assets enables shareability, discoverability, and reusability. The repository also provides data consistency for training and inference. This helps data scientists and ML researchers save time on data preparation and feature engineering, which typically take up a significant amount of their time. Key functionalities in the data and feature repository include the following:

- Enable shareability, discoverability, reusability, and versioning of data assets.
- Allow real-time ingestion and low-latency serving for event streaming and online prediction workloads.
- Allow high-throughput batch ingestion and serving for extract, transform, load (ETL) processes and model training, and for scoring workloads.
- Enable feature versioning for point-in-time queries.
- Support various data modalities, including tabular data, images, and text.

ML data assets can be managed at the entity features level or at the full dataset level. For example, a feature repository might contain an entity called customer, which includes features like age group, postal code, and gender. On the other hand, a dataset repository might include a customer churn dataset, which includes features from the customer and product entities, as well as purchase- and web-activity event logs.

ML metadata and artifact tracking

Various types of ML artifacts are produced in different processes of the MLOps lifecycle, including descriptive statistics and data schemas, trained models, and evaluation results. ML metadata is the information about these artifacts, including their location, types, properties, and associations to experiments and runs. The ML metadata and artifact tracking capability is foundational to all other MLOps capabilities. Such a capability enables reproducibility and debugging of complex ML tasks and pipelines. Key functionalities in ML metadata and artifact tracking include the following:

- Provide traceability and lineage tracking of ML artifacts.
- Share and track experimentation and pipeline parameter configurations.
- Store, access, investigate, visualize, download, and archive ML artifacts.
- Integrate with all other MLOps capabilities.

Deep dive of MLOps processes

This section describes each of the core MLOps processes in detail. It describes key tasks and flow of control between tasks, the key artifacts created by the tasks, and the relationship of tasks to other upstream and downstream processes. In this section, you learn about concrete details of tasks like running a continuous training pipeline, deploying a model, and monitoring predictive performance of the model.

MLOps processes take place on an integrated ML platform that has the required development and operations capabilities (described later). Infrastructure engineers can provision this type of platform in different environments (like development, test, staging, and production) using configuration management and infrastructure-as-code (IaC) tools like Terraform. Each environment is configured with its own set of required compute resources, data access, and subset of MLOps capability services.

ML development

Experimentation is the core activity in ML development, where your data scientists can rapidly try several ideas for data preparation and ML modeling. Experimentation starts when the ML use case is well defined, meaning that the following questions have been answered:

- What is the task?
- How can we measure business impact?
- What is the evaluation metric?

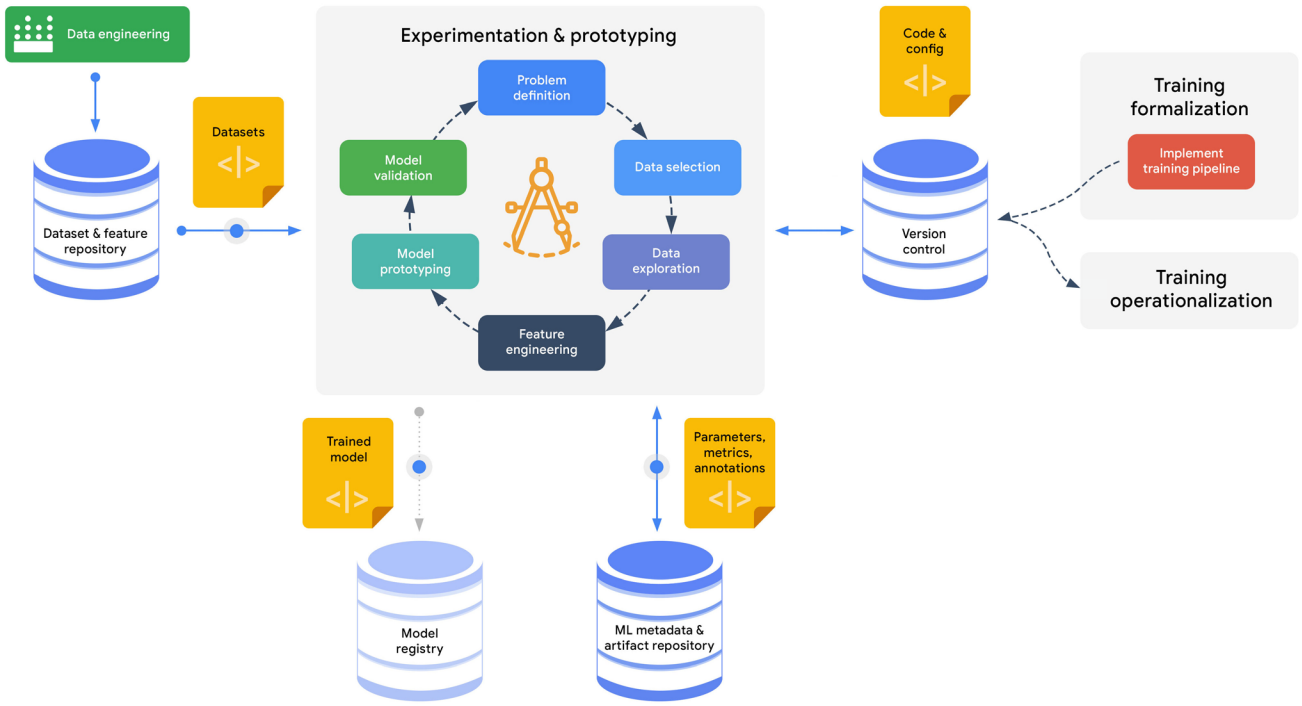

Figure 5. The ML development process

- What is the relevant data?
- What are the training and serving requirements?

Experimentation aims to arrive at an effective prototype model for the ML use case at hand. In addition to experimentation, data scientists need to formalize their ML training procedures. They do this by implementing an end-to-end pipeline, so that the procedures can be operationalized and run in production. Figure 5 shows the process of ML development.

During experimentation, data scientists typically perform the following steps:

- Data discovery, selection, and exploration.
- Data preparation and feature engineering, using interactive data processing tools.
- Model prototyping and validation.

Performing these iterative steps can lead data scientists to refining the problem definition. For example, your data scientists or researchers might change the task from regression to classification, or they might opt for another evaluation metric.

The primary source of development data is the dataset and feature repository. This repository contains curated data assets that are managed on either the entity-features level or the full dataset level.

In general, the key success aspects for this process are experiment tracking, reproducibility, and collaboration. For example, when your data scientists begin working on an ML use case, it can save them time if they can find previous experiments that have similar use cases and that reproduce the results of those experiments; data scientists can then adapt those experiments to the task at hand. In addition, data scientists need to be able to compare various experiments and to compare different runs of the same experiment so that they understand the factors that lead to changing the model's predictive behavior and to performance improvements.

To be able to reproduce an experiment, your data science team needs to track configurations for each experiment, including the following:

- A pointer to the version of the training code in the version control system.
- The model architecture and pretrained modules that were used.
- Hyperparameters, including trials of automated hyperparameter tuning and model selection.
- Information about training, validation, and testing data splits that were used.
- Model evaluation metrics and the validation procedure that was used.

If there is no need to retrain the model on a regular basis, then the produced model at the end of the experimentation is submitted to the model registry. The model is then ready to be reviewed, approved, and deployed to the target

serving environment. In addition, all the relevant metadata and artifacts that were produced during model development are tracked in the metadata tracking repository.

However, in most cases, ML models need to be retrained on a regular basis when new data is available or when the code changes. In this case, the output of the ML development process is not the model to be deployed in production. Instead, the output is the implementation of the continuous training pipeline to be deployed to the target environment. Whether you use code-first, low-code, or no-code tools to build continuous training pipelines, the development artifacts, including source code and configurations, must be version controlled (for example, using Git-based source control systems). This lets you apply standard software engineering practices to your code review, code analysis, and automated testing. It also lets you build a CI/CD workflow to deploy the continuous training pipeline to the target environment.

Experimentation activities usually produce novel features and datasets. If the new data assets are reusable in other ML and analytics use cases, they can be integrated into the feature and dataset repository through a data engineering pipeline. Therefore, a common output of the experimentation phase is the requirements for upstream data engineering pipelines (see Figure 1).

Training operationalization

Training operationalization is the process of building and testing a repeatable ML training pipeline and then deploying it to a target execution environment. For MLOps, ML engineers should be able to use configurations to deploy the ML pipelines. The configurations specify variables like the target deployment environment (development, test, staging, and so on), the data sources to access during execution in each environment, and the service account to use for running compute workloads. Figure 6 shows the stages for an approach for a training pipeline.

ML Development

Typical assets produced in this process include the following:

- Notebooks for experimentation and visualization
- Metadata and artifacts of the experiments
- Data schemas
- Query scripts for the training data
- Source code and configurations for data validation and transformation
- Source code and configurations for creating, training, and evaluating models
- Source code and configurations for the training-pipeline workflow
- Source code for unit tests and integration tests

Core MLOps capabilities:

- Dataset & feature repository
- Data processing
- Experimentation
- Model training
- Model registry
- ML metadata & artifact repository

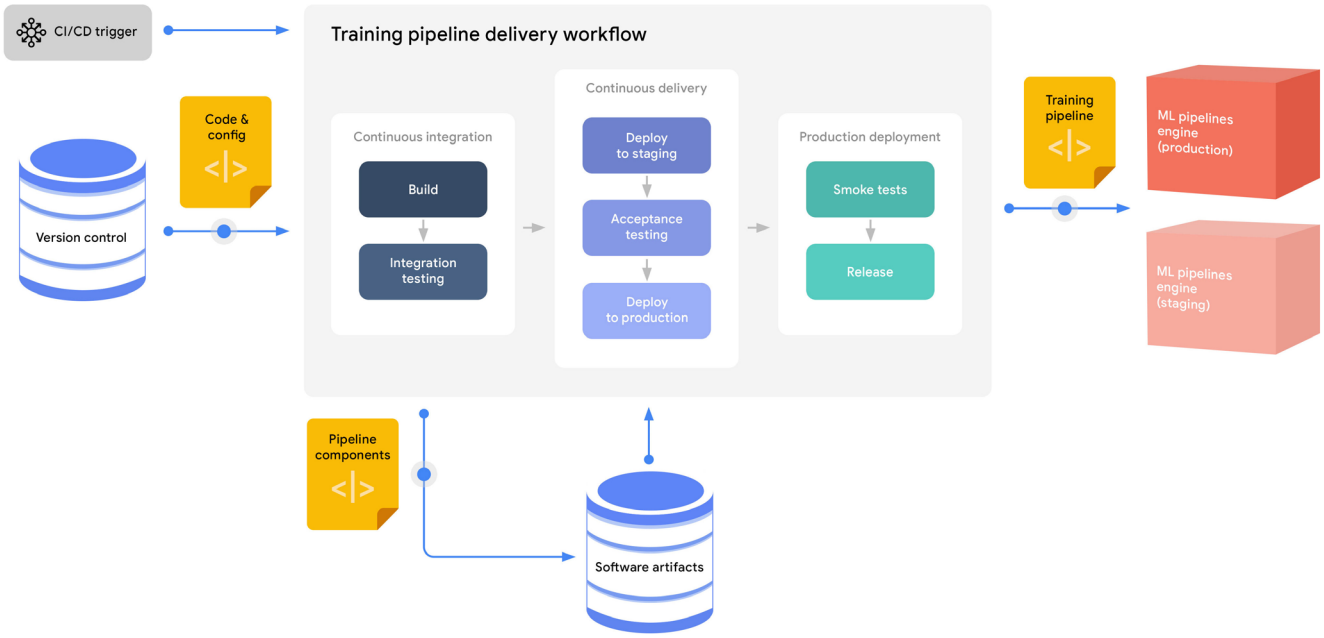

Figure 6. The training operationalization process

A pipeline typically goes through a series of testing and staging environments before it is released to production. The number of testing and staging environments varies depending on standards that are established in a given organization. Most organizations have at least one testing environment before production; some have more.

The specifics of the pipeline deployment process depend on the technology that is used to implement the pipeline. With some no-code solutions, data scientists and ML engineers don't handle or even see the details.

Alternatively, if you use a code-first technology to have more flexibility and control over the ML pipelines, ML engineers can deploy the pipeline using standard CI/CD processes and tools. This approach is what the diagram depicts. The diagram shows a standard CI/CD workflow, which consists of these stages:

1. In the CI stage, the source code is unit-tested, and the training pipeline is built and integration-tested. Any artifacts that are created by the build are stored in an artifact repository.

Training Operationalization

Typical assets produced in this process include the following:

- Training-pipeline executable components (for example, container images stored in a container registry)
- Training-pipeline runtime representation, which references the components stored in an artifacts repository

Core MLOps capabilities:

- ML pipelines

2. In the CD stage, the tested training pipeline artifacts are deployed to a target environment, where the pipeline goes through end-to-end testing before being released to the production environment. Typically, pipelines are tested in non-production environments on a subset of production data, while the full-scale training is performed only in production environments.
3. The newly deployed training pipeline is smoke-tested. If the new pipeline fails to produce a deployable model, the model serving system can fall back to the model that was produced by the current training pipeline.

Continuous training

The continuous training process is about orchestrating and automating the execution of training pipelines. How frequently you retrain your model depends on your use case and on the business value and cost of doing so. For example, if you are using an ML model that performs well in classifying images, and if there are no changes in the environment in which the images are generated and collected, there might be little benefit in retraining the model on new data every day. On the other hand, if you run a shopping site that has a recommendation system, user behavior changes continually, and retraining frequently on new data captures new trends and patterns. This investment in retraining can lead to higher click-through rates and therefore potential additional purchases.

Each run of the pipeline can be triggered in several ways, including the following:

- Scheduled runs based on jobs that you configure.
- Event-driven runs, such as when new data becomes available above a certain threshold, or when model decay is detected by the continuous monitoring process.
- Ad hoc runs based on manual invocation.

Figure 7 shows the flow of a canonical pipeline.

Typical ML training pipelines have workflows that include to the following:

1. Data ingestion. Training data is extracted from the source dataset and feature repository using filtering criteria such as the date and time of the most recent update.
2. Data validation. The extracted training data is validated to make sure that the model is not trained using skewed or corrupted data.
3. Data transformation. The data is split, typically into training, evaluation, and testing splits. The data is then transformed, and the features are engineered as expected by the model.
4. Model training and tuning. The ML model is trained and the hyperparameters are tuned using the training and evaluation data splits to produce the best possible model.

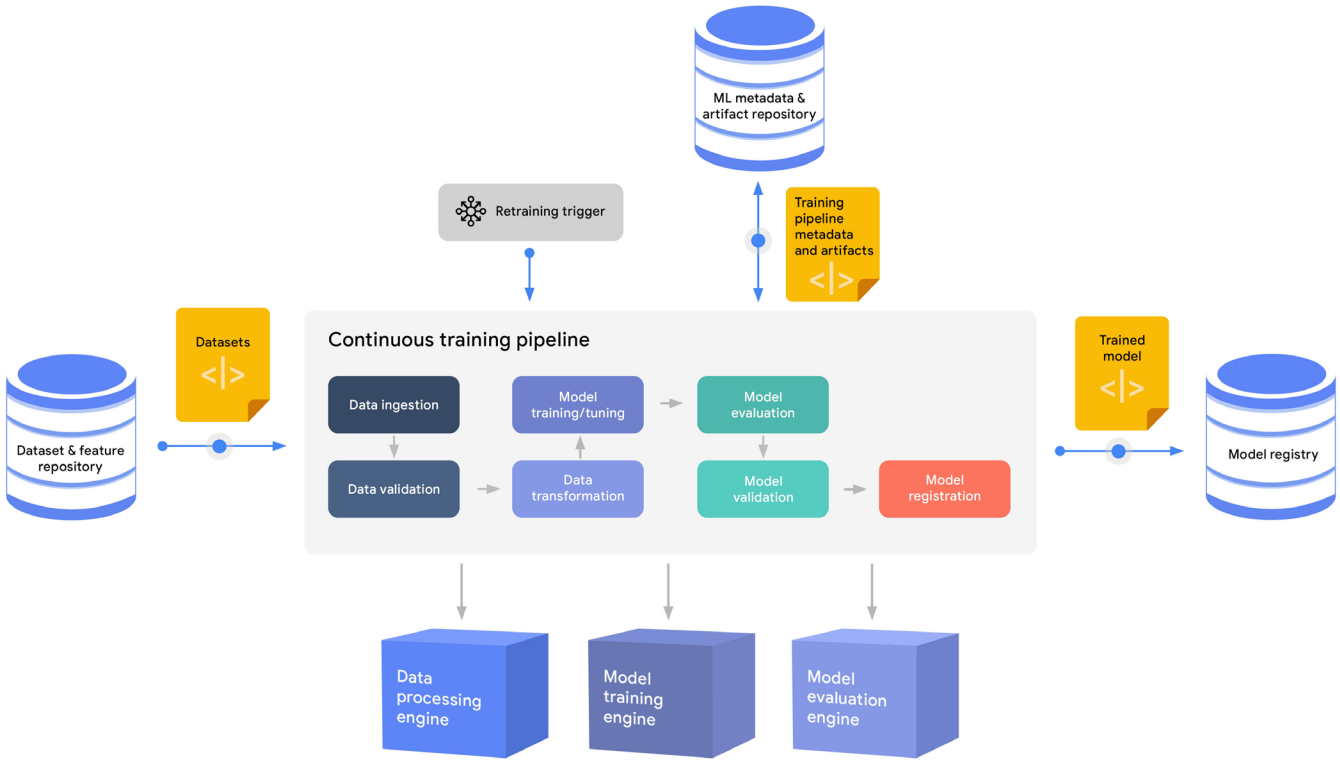

Figure 7. The continuous training process

5. Model evaluation. The model is evaluated against the test data split to assess the performance of the model using various evaluation metrics on different partitions of the data.
6. Model validation. The results of model evaluations are validated to make sure that the model meets the expected performance criteria.
7. Model registration. The validated model is stored in a model registry along with its metadata.

As the diagram shows, the continuous training pipeline runs based on a retraining trigger. When the pipeline starts, it extracts a fresh training dataset from the dataset and feature repository, executes the steps in the ML workflow, and submits a trained model to the model registry. All the run information and the artifacts that are produced throughout the pipeline run are tracked in the metadata and artifact repository.

An orchestrated and automated training pipeline mirrors the steps of the typical data science process that runs in the ML development phase. However, the automated pipeline has some differences. In an automated pipeline, data validation and model validation play a critical role in a way that they don't during experimentation; these steps are gatekeepers for the overall ML training process. Runs of an automated pipeline are unattended. Therefore, as the runs are executed, the training and evaluation data can evolve to a point where the data processing and training procedures that are implemented by the pipeline are no longer optimal, and possibly are no longer valid.

The data validation step can detect when data anomalies start occuring. These anomalies can include new features, new feature domains, features that have disappeared, and drastic changes in feature distributions. The data validation step works by comparing the new training data to the expected data schema and reference statistics. For details about how data validation works, see Analyzing and validating data at scale for machine learning with TensorFlow Data Validation.

The model validation phase detects anomalies when a lack of improvement or even a degradation in performance of new model candidates is observed. The pipeline can apply complex validation logic in this step, including several evaluation metrics, sensitivity analysis based on specific inputs, calibration, and fairness indicators.

An important aspect of continuous training, therefore, is tracking. Pipeline runs must track generated metadata and artifacts in a way that enables debugging, reproducibility, and lineage analysis. Lineage analysis means being able to track a trained model back to the dataset (snapshot) that was used to train that model, and to link all the intermediate artifacts and metadata. Lineage analysis supports the following:

- Retrieve the hyperparameters that are used during training.
- Retrieve all evaluations that are performed by the pipeline.
- Retrieve processed data snapshots after transformation steps, if feasible.
- Retrieve data summaries such as descriptive statistics, schemas, and feature distributions.

When a pipeline run has finished and the model has been validated, the pipeline can register a model candidate in the model registry.

An automated pipeline can include deployment steps, effectively acting as a unified, end-to-end training and deployment pipeline. In some use cases, where the model is trained and deployed several times per day (for example, every hour), training and deployment might be implemented and streamlined into a single pipeline. In these cases, additional validations are built into the pipeline, such as model size validation, serving runtime validation (for dependencies and accelerators), and serving latency evaluation.

Continuous Training

Typical assets produced in this process include the following:

- A trained and validated model stored in the model registry
- Training metadata and artifacts stored in the ML metadata and artifacts repository, including pipeline execution parameters, data statistics, data validation results, transformed data files, evaluation metrics, model validation results, and training checkpoints and logs

Core MLOps capabilities:

- Dataset & feature repository
- ML metadata & artifact repository
- Data processing
- Model training
- Model evaluation
- ML pipelines
- Model registry

However, creating a complete pipeline like this might not be practical in all organizations. For example, in some organizations, model training and model deployment are the responsibilities of different teams. Therefore, the scope of most of the training pipelines ends at registering a trained and validated model, rather than at deploying it for serving.

Model deployment

After a model has been trained, validated, and added to the model registry, it is ready for deployment. During the model deployment process, the model is packaged, tested, and deployed to a target serving environment. As with the training operationalization phase, the model deployment process can involve a number of testing steps and testing environments. The model might need to go through a model governance process before it is allowed to be deployed to a target environment.

Figure 8 shows a high-level view of the model deployment process.

When you use no-code or low-code solutions, the model deployment process is streamlined and abstracted from the

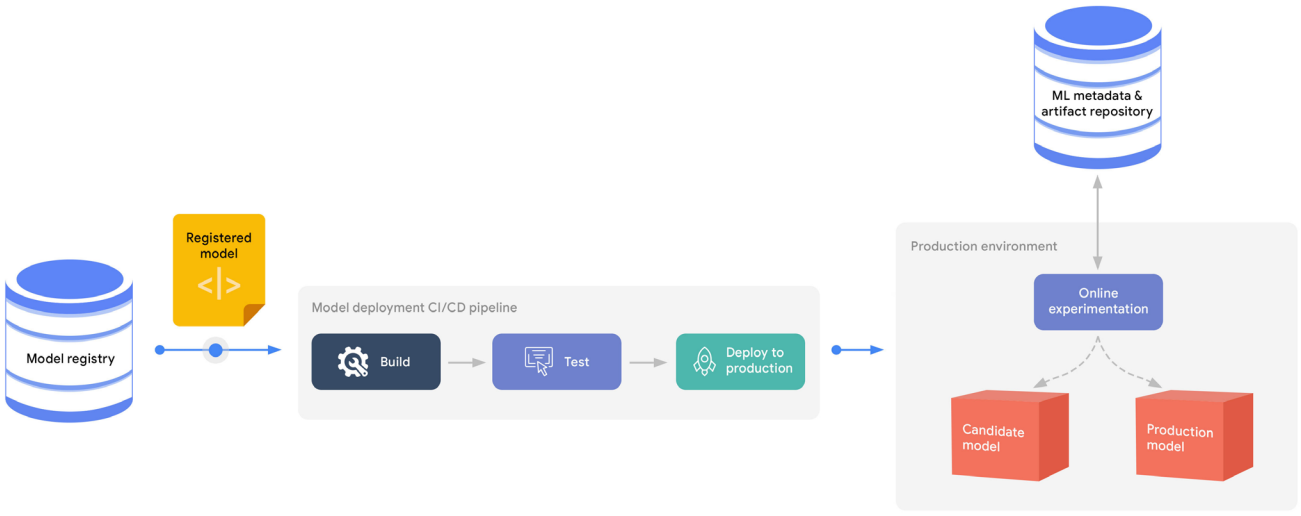

Figure 8. Model deployment progressive delivery workflow

perspective of the data scientists and ML engineers. Typically, you point to the entry for the model in the model registry, and the model is deployed automatically using the metadata and the artifacts that are stored for that model.

However, in other scenarios, you might want more control over the model deployment process, therefore the process

requires complex CI/CD routines. In that case, the CI/CD system reads the source code of the model serving component from the source repository and fetches the model from the model registry. The system integrates, builds, tests, and validates the model serving service, and then deploys the service through a progressive delivery process. Figure 9 shows this process.

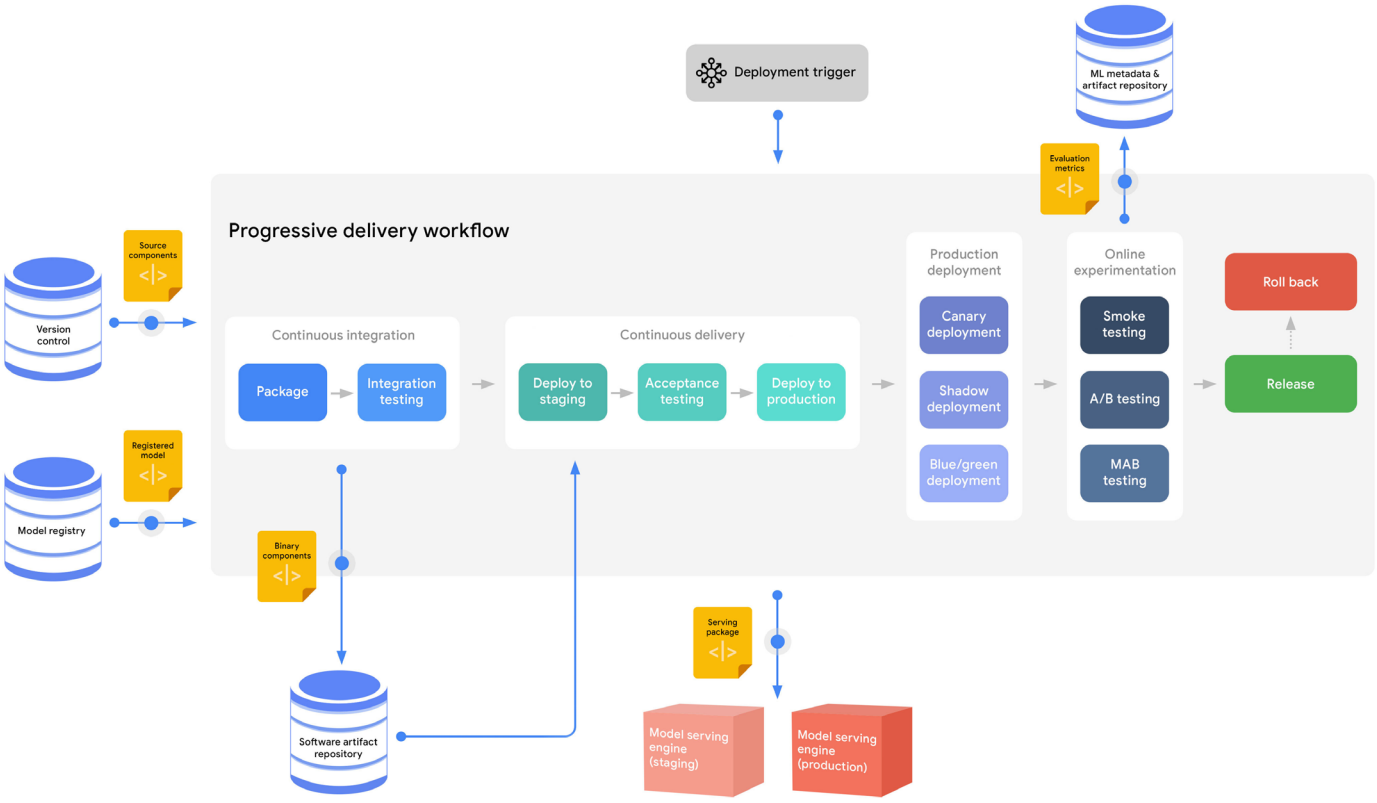

Figure 9. A complex CI/CD system for the model deployment process

In the CI stage of model deployment, tests might include testing your model interface to see if it accepts the expected input format and if it produces the expected output. You might also validate the compatibility of the model with the target infrastructure, such as checking for required packages and accelerators. During this stage, you might also check that the model's latency is acceptable.

In the CD stage of model deployment, the model undergoes progressive delivery. Canary deployments, blue-green deployments, and shadow deployments are often used to perform smoke testing, which usually focuses on model service efficiency like latency and throughput, as well as on service errors. After you verify that the model works technically, you test the model's effectiveness in production by gradually serving it alongside the existing model and running online experiments, which refers to testing new functionality in production with live traffic.

Online experimentation is particularly important in the context of ML. Deciding whether a new model candidate should replace the production version is a more complex and multi-dimensional task compared to deploying other software assets.

In the progressive delivery approach, a new model candidate does not immediately replace the previous version. Instead, after the new model is deployed to production, it runs in parallel to the previous version. A subset of users is redirected to the new model in stages, according to the online experiment in place. The outcome of the experiments is the final criterion that decides whether the model can be fully released and can replace the previous version. A/B testing and multi-armed bandit (MAB) testing are common online experimentation techniques that you can use to quantify the impact of the new model on application-level objectives. Canary and shadow deployment methods can facilitate such online experiments.

Prediction serving

In the prediction serving process, after the model is deployed to its target environment, the model service starts to accept prediction requests (serving data) and to serve responses with predictions. Figure 10 shows the elements of prediction serving.

Model Deployment

Typical assets produced in this process include the following:

- Model serving executable application (for example, a container image stored in a container registry or a Java package stored in an artifact repository)
- Online experimentation evaluation metrics stored in ML metadata and artifact repository

Core MLOps capabilities:

- Model serving
- Model registry
- Online experimentation
- ML metadata & artifact repository

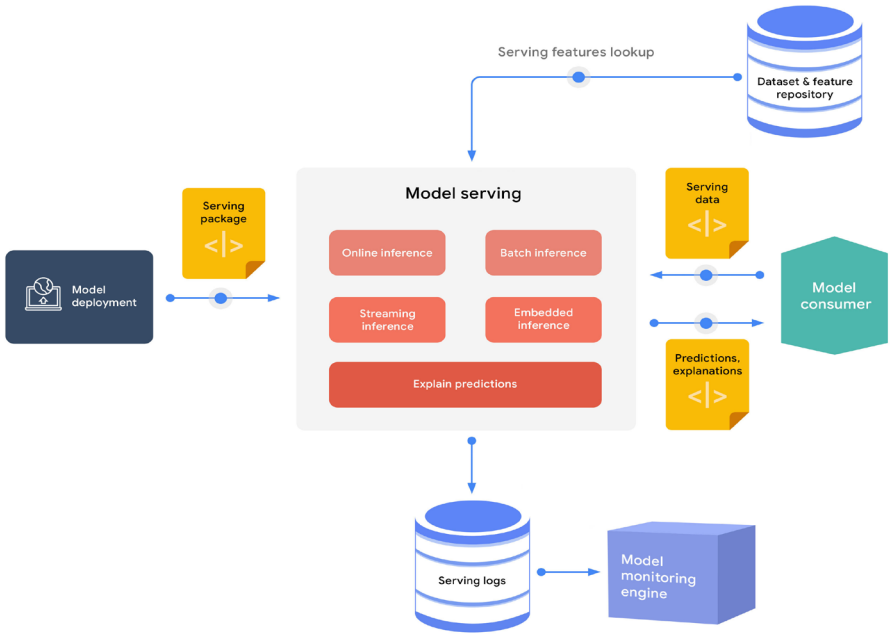

Figure 10. Elements of the prediction serving process

The serving engine can serve predictions to consumers in the following forms:

- Online inference in near real time for high-frequency singleton requests (or mini batches of requests), using interfaces like REST or gRPC.
- Streaming inference in near real time, such as through an event-processing pipeline.
- Offline batch inference for bulk data scoring, usually integrated with extract, transform, load (ETL) processes.
- Embedded inference as part of embedded systems or edge devices.

In some scenarios of prediction serving, the serving engine might need to look up feature values that are related to the request. For example, you might have a model that predicts the propensity of a customer to buy a particular product, given a set of customer and product features. However, the request includes only the customer and the product identifier. Therefore, the serving engine uses these identifiers to fetch the customer and the product feature values from a feature repository and then to feed them to the model to produce a prediction.

An important part of having confidence in ML systems is being able to interpret the models and provide explanations to their predictions. The explanations should provide insight into the rationale for the prediction—for example, by generating feature attributions for a given prediction. Feature attributions indicate in the form of scores how much each feature contributes to a prediction.

The inference logs and other serving metrics are stored for continuous monitoring and analysis.

Prediction Serving

Typical assets produced in this process include the following:

- Request-response payloads stored in the serving logs store
- Feature attributions of the predictions

Core MLOps capabilities:

- Dataset & feature repository
- Model serving

Continuous monitoring

Continuous monitoring is the process of monitoring the effectiveness and efficiency of a model in production, which is a crucial area of MLOps. It is essential to regularly and proactively verify that the model performance doesn't decay. As the serving data changes over time, its properties start

to deviate from the properties data that was used for training and evaluating the model. This leads to model effective performance degradation. In addition, changes or errors in upstream systems that produce the prediction requests might lead to changes to the properties of the serving data, and consequently produce bad predictions from the model.

The monitoring engine uses the inference logs to identify anomalies (skews and outliers), as shown in figure 11.

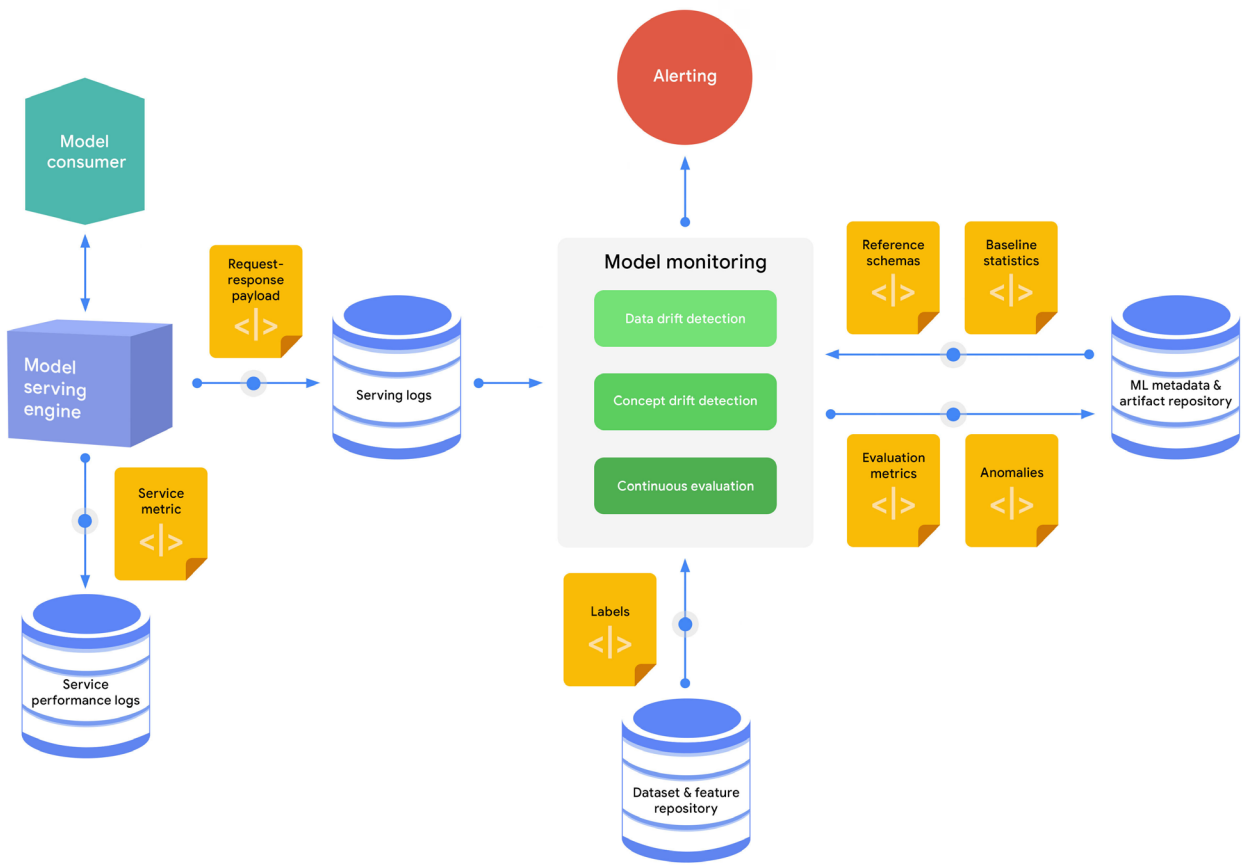

Figure 11. The continuous monitoring process

A typical continuous monitoring process consists of the following steps:

1. A sample of the request-response payloads is captured and stored in the serving logs store.
2. The monitoring engine periodically loads the latest inference logs, generates a schema, and computes statistics for the serving data.
3. The monitoring engine compares the generated schema to a reference schema to identify schema skews,

and compares the computed statistics to baseline statistics to identify distribution skews.

4. If the true labels (ground truth) for the serving data are available, the monitoring engine uses it to evaluate the model predictive effectiveness in hindsight on the serving data.
5. If anomalies are identified, or if the model's performance is decaying, alerts can be pushed through various channels (for example, email or chat) to notify the owners to examine the model or to trigger a new retraining cycle.

Effectiveness performance monitoring aims to detect model decay. Model decay is usually defined in terms of data and concept drift.

Data drift describes a growing skew between the dataset that was used to train, tune, and evaluate the model and the production data that a model receives for scoring. *Concept drift* is an evolving relationship between the input predictors and the target feature.

Data drift can involve two types of skews:

- *Schema skew* occurs when training data and serving data do not conform to the same schema.
- *Distribution skew* occurs when the distribution of feature values for training data is significantly different from the distribution for serving data.

In addition to identifying schema and distribution skews, other techniques for detecting data and concept drift include novelty and outlier detection, as well as feature attributions change. For more information, see ML model monitoring reference guides in the Google Cloud documentation.

In some scenarios, your system is able to store ground truth for your serving data. For example, you capture whether a customer bought a product recommended by your model, or you calculate the actual demand for a particular product by the end of the week compared to the demand that was forecasted by the model. You can use this information as true labels to your serving data, and the information can be stored and retrieved from the dataset and feature repository for continuous evaluation and for further model training cycles.

Continuous Monitoring

Typical assets produced in this process include the following:

- Anomalies detected in serving data during drift detection
- Evaluation metrics produced from continuous evaluation

Core MLOps capabilities:

- Dataset & feature repository
- Model monitoring
- ML metadata & artifact repository

Besides monitoring model effectiveness, monitoring model serving efficiency focuses on metrics like the following:

- Resource utilization, including CPUs, GPUs, and memory.
- Latency, which is a key metric in online and streaming deployments to indicate model service health.
- Throughput, which is a key metric in all deployments.
- Error rates.

Measuring these metrics is helpful not only in maintaining and improving system performance, but also in predicting and managing costs.

Data and model management

At the heart of the six processes outlined earlier is data and model management. This is a central function for governing ML artifacts in order to support auditability, traceability, and compliance, as well as for shareability, reusability, and discoverability of ML assets.

Dataset and feature management

One of the key challenges of data science and ML is creating, maintaining, and reusing high-quality data for training. Data scientists spend a significant amount of their ML development time on exploratory data analysis, data preparation, and data transformation. However, other teams might have prepared the same datasets for similar use cases but have no means for sharing and reusing them. This situation can lead not only to wasted time re-creating the datasets, but to inconsistent definitions and instances of the same data entities.

In addition, during prediction serving, a common challenge is discrepancies between serving data and training data. This is called *training-serving skew*, and can occur because the data is extracted from different sources in different forms during training and serving. Training-serving skew impacts the performance of the models in production.

Dataset and feature management helps mitigate such issues by providing

Data & Model Management

Core MLOps capabilities:

- Dataset & feature repository
- Model registry
- ML metadata & artifact repository

a unified repository for ML features and datasets. Figure 12 shows how the feature and dataset repository provides the same set of data entities for multiple uses in the MLOps environment.

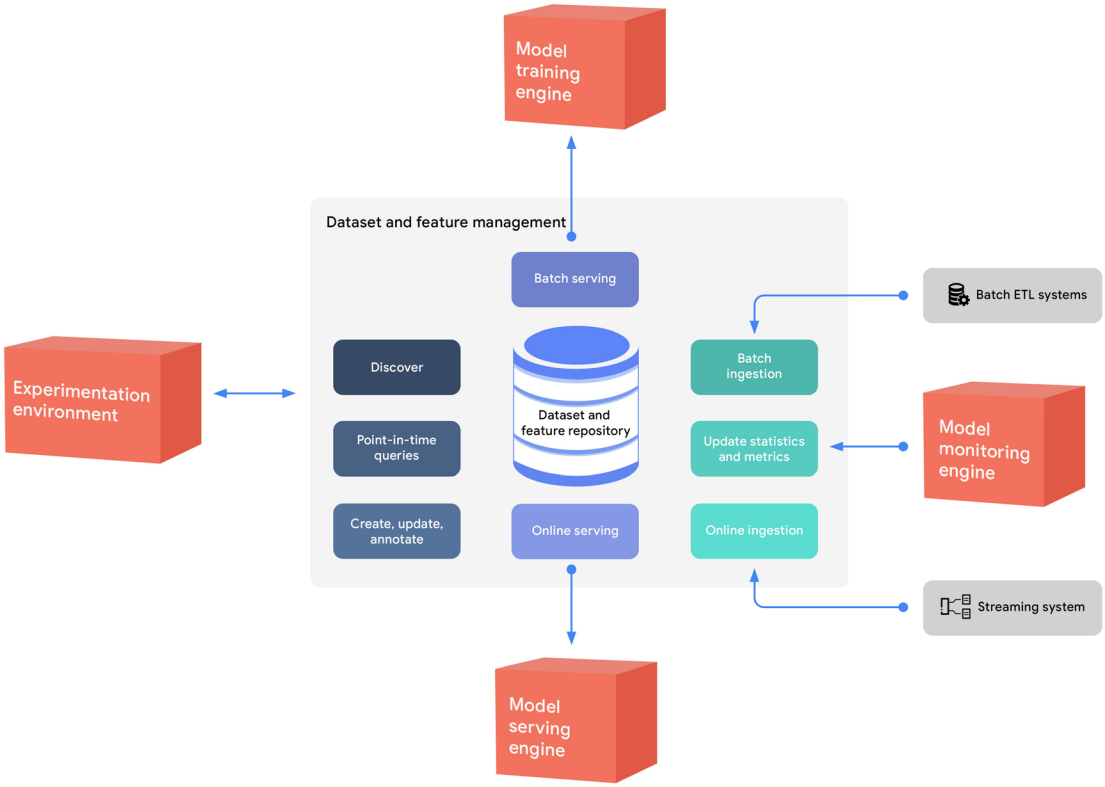

Figure 12. Using the dataset and feature repository to provide entities for multiple uses

As the diagram shows, the features and datasets are created, discovered, and reused in different experiments. Batch serving of the data is used for experimentation, continuous training, and batch prediction, while online serving of the data is used for real-time prediction use cases.

Feature management

Features are attributes of business entities that are cleansed and prepared based on standard business rules—aggregations, derivations, flags, and so on. Examples of entities include product, customer, location, and promotion. You can manage your data entities in a centralized repository to standardize their definition, storage, and access for training and serving. A feature repository helps data scientists and researchers do the following:

- Discover and reuse available feature sets for their entities instead of re-creating the entities in order to create

their own datasets.

- Establish a central definition of features.
- Avoid training-serving skew by using the feature repository as the data source for experimentation, continuous training, and online serving.
- Serve up-to-date feature values from the feature repository.
- Provide a way of defining and sharing new entities and features.
- Improve collaboration between data science and research teams by sharing features.

In batch ETL systems, the training pipeline can retrieve the features as a batch for the training task. For online serving, the serving engine can fetch the feature values that are related to the requested entity. Updates to the feature repository can be ingested from the batch ETL or streaming systems. In addition to those updates, the monitoring service can update statistics and metrics for these features.

Dataset management

Features can be used in many datasets for multiple ML tasks and use cases, while a dataset is used for a particular ML task or use case. More precisely, feature repositories typically don't include labeled data instances (instances with predictable targets). Instead, they include reusable feature values of various entities. The features of different entities can be combined and joined with other transactional data that contains labels in order to create a dataset.

For example, the feature repository might contain a customer entity that includes features that describe customer demographics, purchase behavior, social media interactions, sentiment scores, third-party financial flags, and so on. The customer entity can be used in several tasks, such as churn prediction, click-through rate prediction, customer lifetime value estimation, customer segmentation, and recommendations. Each task has its own dataset that contains the customer features and other features from the entities that are relevant to the task. In addition, in case of supervised learning tasks, each dataset has its own labels.

Dataset management helps with the following:

- Maintaining scripts for creating datasets and splits so that datasets can be created in different environments (development, test, production, and so on).
- Maintaining a single dataset definition and realization within the team to use in various model implementations and hyperparameters. This dataset includes splits (training, evaluation, test, and so on) and filtering conditions.
- Maintaining metadata and annotation that can be useful for team members who are collaborating on the same dataset and task.
- Providing reproducibility and lineage tracking.

Model management

As organizations add to the number of models in production at scale, it becomes difficult to keep track of all of them manually. Organizations need controls in order to manage risk and implement ML models responsibility, as well as to maintain compliance with regulations.

To help with this task, organizations need to establish robust model management. Model management is a cross-cutting process that is at the center of MLOps. It entails both ML metadata tracking and model governance. Having model management across the ML lifecycle helps ensure the following:

- The data that is being collected and used for model training and evaluation is accurate, unbiased, and used appropriately without any data privacy violations.
- The models are evaluated and validated against effectiveness quality measures and fairness indicators, so that they are fit for deployment in production.
- The models are interpretable, and their outcomes are explainable (if needed).
- The performance of deployed models is monitored using continuous evaluation and the models' performance metrics are tracked and reported.
- Potential issues in model training or prediction serving are traceable, debuggable, and reproducible.

ML metadata tracking

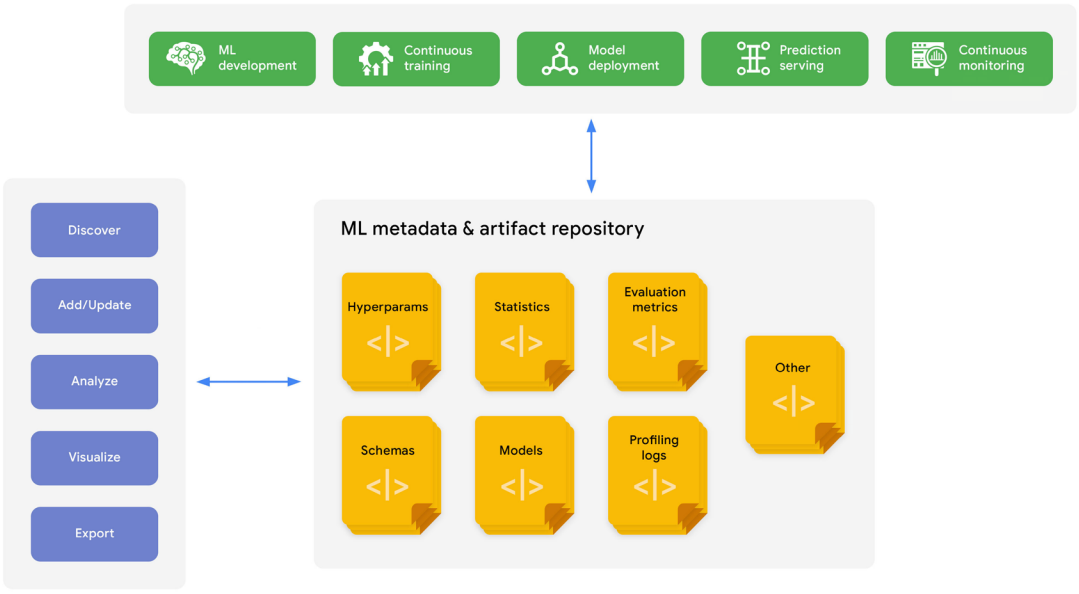

Figure 13. Metadata tracking

ML metadata tracking is generally integrated with different MLOps processes. The artifacts produced by the other processes are usually automatically stored in an ML artifact repository, along with the information about the process executions. ML metadata that is captured can include pipeline run ID, trigger, process type, step, start and end date-time, status, environment configurations, and input parameter values. Examples of artifacts that are stored include processed data splits, schemas, statistics, hyperparameters, models, and evaluation metrics or custom artifacts. Figure 13 shows metadata tracking.

ML metadata tracking lets data scientists and ML engineers track experimentation parameters and pipeline configurations for reproducibility and for tracing lineage. In addition, ML metadata tracking lets users search, discover, and export existing ML models and artifacts.

Data scientists and ML engineers can use ML metadata tracking to add and update annotations to the tracked ML experiments and runs. This facilitates discoverability. Moreover, ML metadata tracking provides the tools for analyzing, comparing, and visualizing the metadata and artifacts of different experiments and ML pipeline runs. This helps data scientists and ML engineers understand the behavior of the pipelines and to debug ML issues.

Model governance

Model governance is about registering, reviewing, validating, and approving models for deployment. Depending on the organization, on the regulatory requirements of the model, and on the particular use case, the process of model governance can differ. The process can be automated, semi-automated, or fully automated (with multiple release criteria in all cases) to determine whether ML models are ready to go to production. In addition, model governance should support reporting on the performance of deployed models.

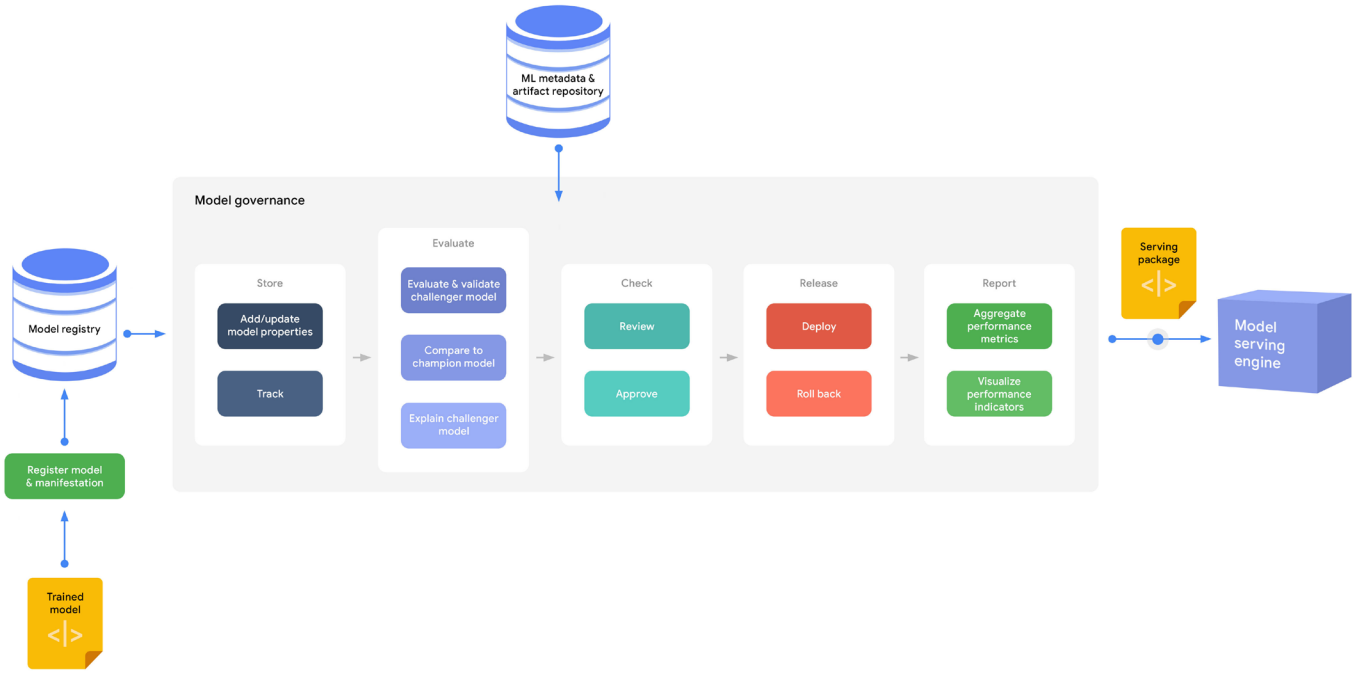

Figure 14. Tasks involved in model governance

Figure 14 shows the tasks that are involved in model governance.

Model governance can use information in the ML metadata and the model registry to do the following tasks:

- **Store:** Add or update model properties and track model versions and property changes. The model registry can store many model versions from the experimentation and continuous training phases, which lets data scientists easily reproduce significant models.
- **Evaluate:** Compare a new challenger model to the champion model by looking not only at evaluation metrics (accuracy, precision, recall, specificity, and so on) but also at business KPIs that are collected through online experimentation. Additionally, model owners need to be able to understand and explain the model predictions—for example, by using feature attribution methods. This ensures the quality of the model that is deployed in production.
- **Check:** Review, request changes, and approve the model to help control for risks, such as business, financial, legal, security, privacy, reputational, and ethical concerns.
- **Release:** Invoke the model deployment process to go live. This controls the type of the model release (for example, canary or blue-green) and the traffic split that is directed to it.
- **Report:** Aggregate, visualize, and highlight model performance metrics that are collected from the continuous evaluation process. This ensures the quality of the model in production.

Explainability is particularly important in the case of decision automation. The governance process should provide to risk managers and auditors a clear view of lineage and accountability. The process should also provide them the ability to review decisions in accordance with an organization's ethical and legal responsibilities.

Putting it all together

Delivering business value through ML is not only about building the best ML model for the use case at hand. Delivering this value is also about building an integrated ML system that operates continuously to adapt to changes in the dynamics of the business environment. Such an ML system involves collecting, processing, and managing ML datasets and features; training, and evaluating models at scale; serving the model for predictions; monitoring the model performance in production; and tracking model metadata and artifacts.

In this document, we discuss the core capabilities for building and operating ML systems, and we describe a comprehensive MLOps process to streamline the workflow from development to production. This can help organizations reduce time to market while increasing the reliability, performance, scalability, and security of their ML systems.

Figure 15 provides a summary of the end-to-end MLOps process.

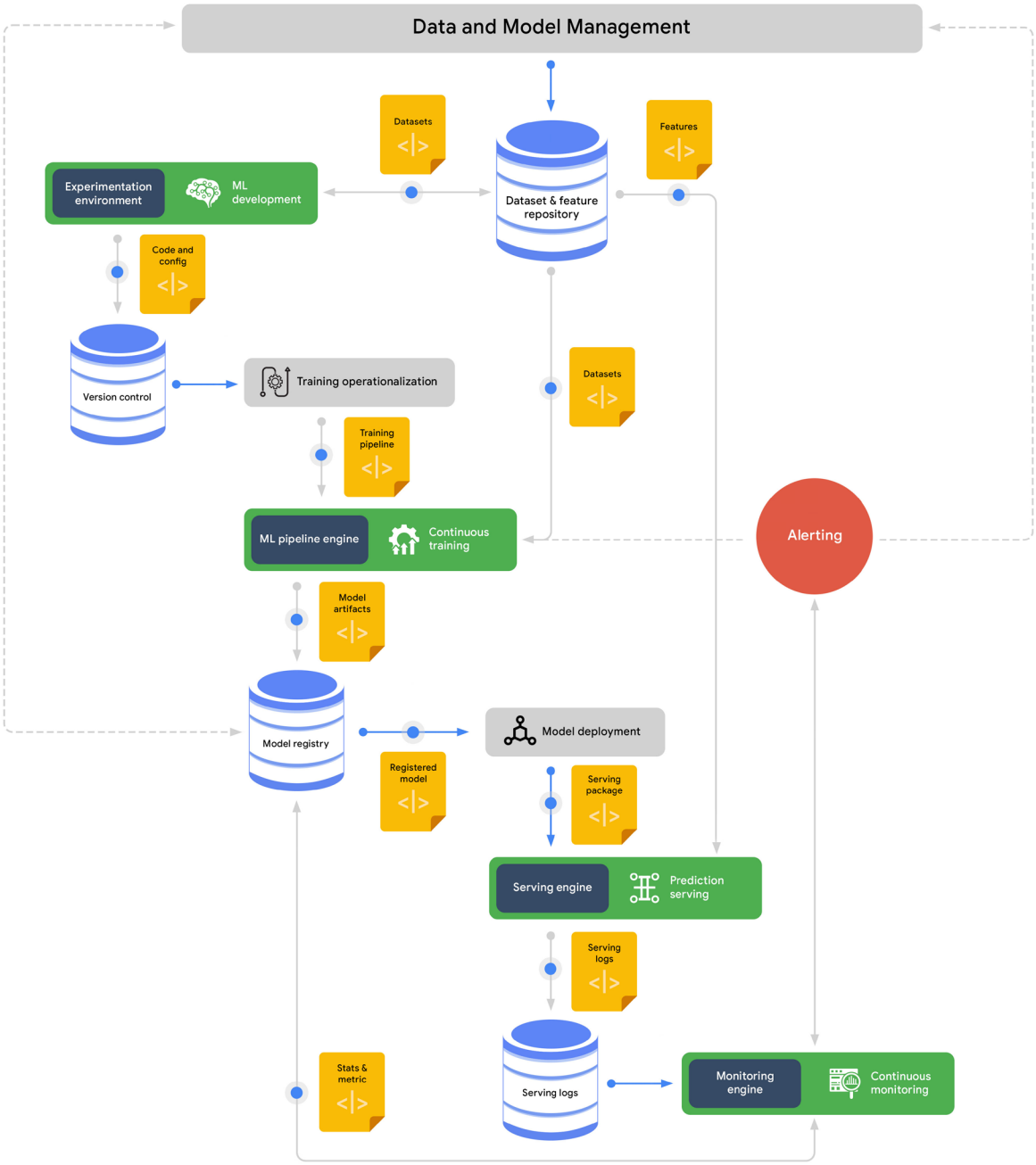

Figure 15. End-to-end MLOps workflow

Additional resources

For more information about how to get started with MLOps on Google Cloud, see the following books, guides, courses, articles, and videos:

- Best Practices for Implementing Machine Learning on Google Cloud
- Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps
- An introduction to MLOps on Google Cloud
- MLOps: Continuous delivery and automation pipelines in machine learning
- Architecture for MLOps using TFX, Kubeflow Pipelines, and Cloud Build
- Setting up an MLOps environment on Google Cloud
- MLOps (Machine Learning Operations) Fundamentals on Coursera

Google Cloud