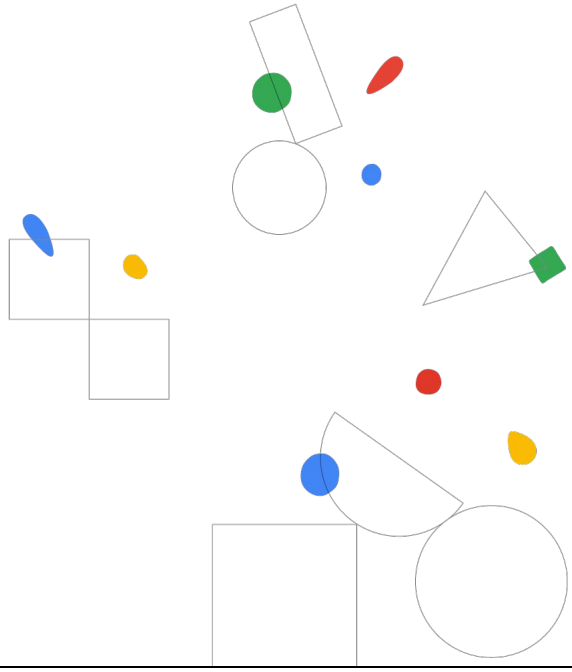
Google Cloud

Architecting ML systems



Module 01

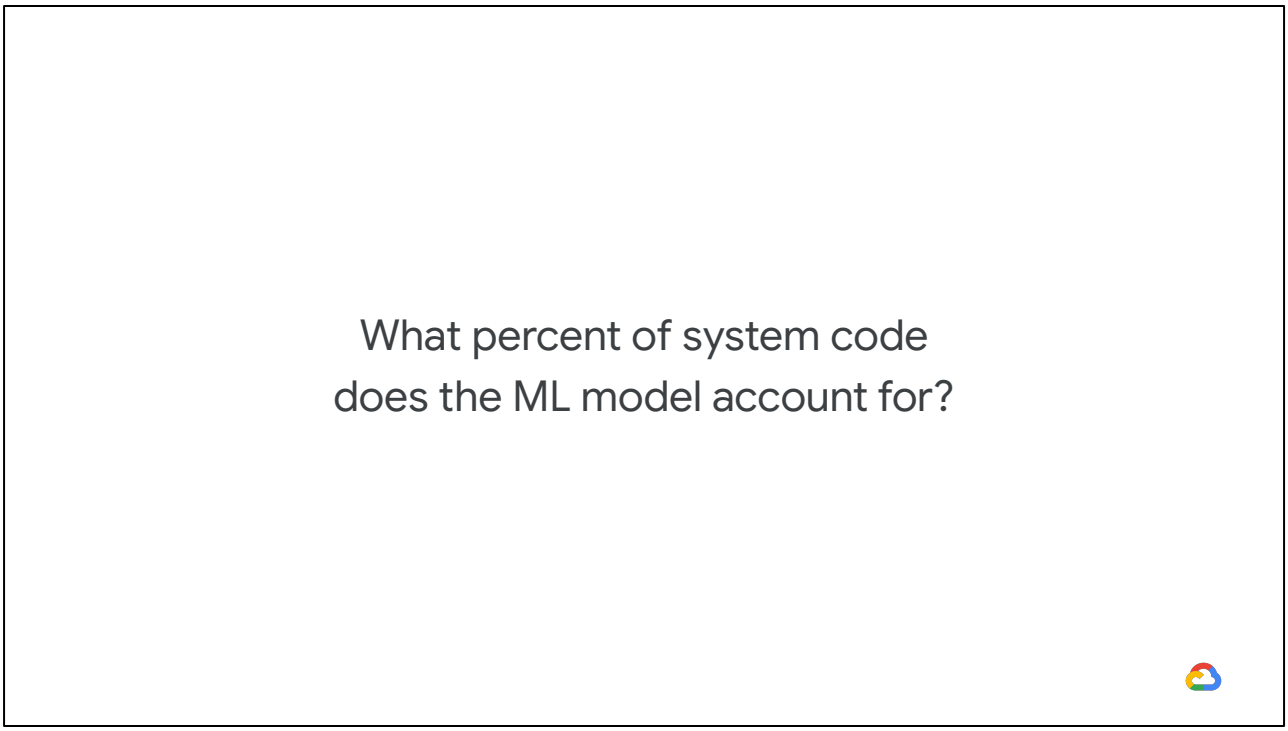
Architecting production ML systems

Welcome to Architecting ML systems, the second module of the Production Machine Learning Systems course.

Objectives

1. What makes up an architecture
2. Why and how to make good systems design decisions

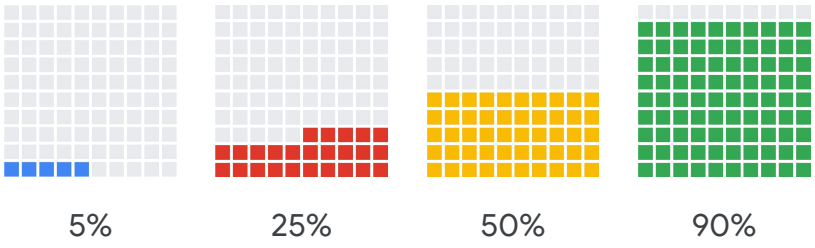
In this module, we'll explore what makes up an architecture as well as why and how to make good systems design decisions.



Let me ask you a question.

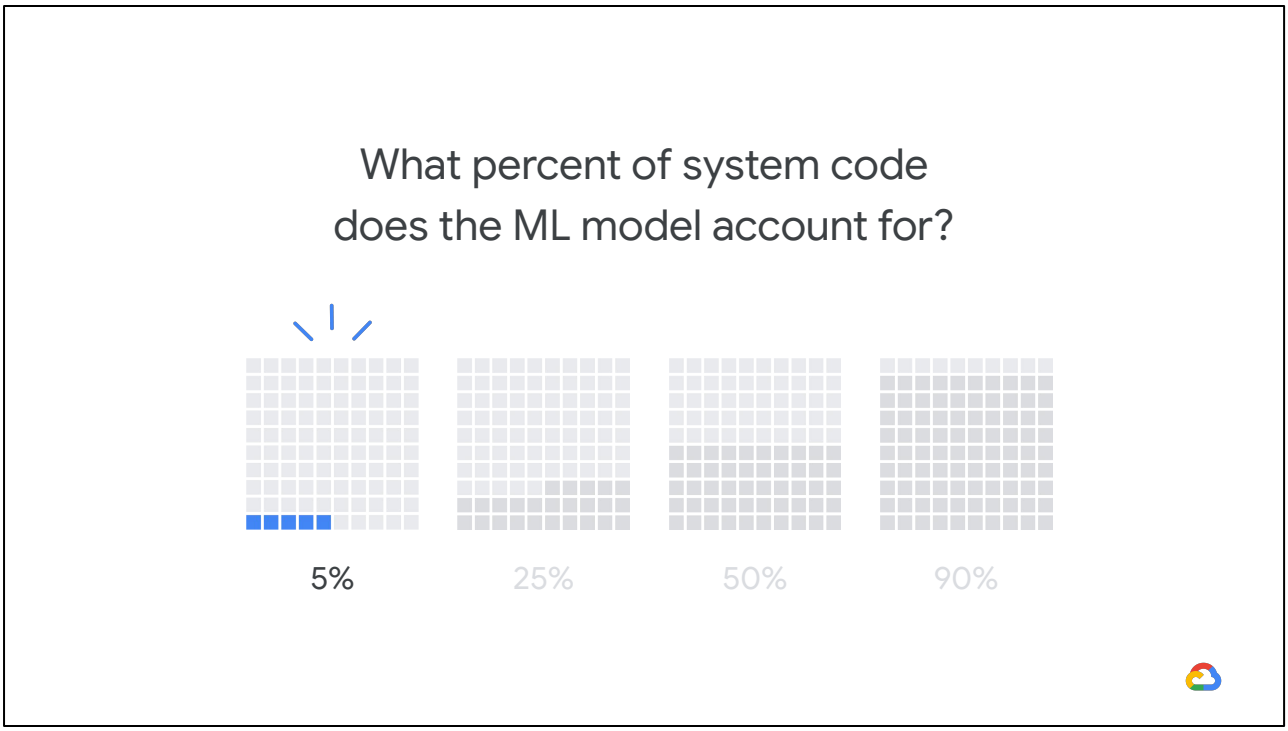
What percent of system code does the ML model account for?

What percent of system code does the ML model account for?

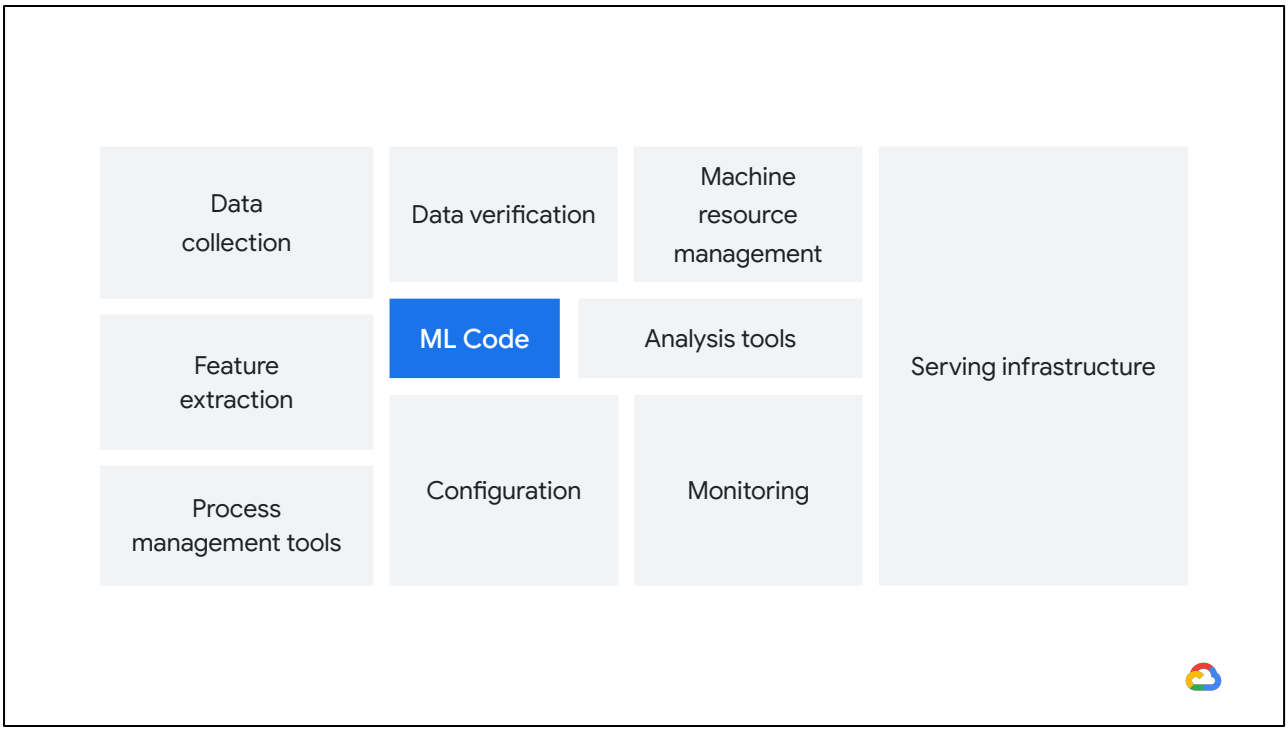


(a) 5%
(b) 25%
(c) 50%
(d) 90%

You'll recall from earlier in this specialization, we showed how time gets distributed among the different tasks necessary to launch an ML model and, surprisingly, modeling accounted for far less than most people expect. The same is true with respect to the code.



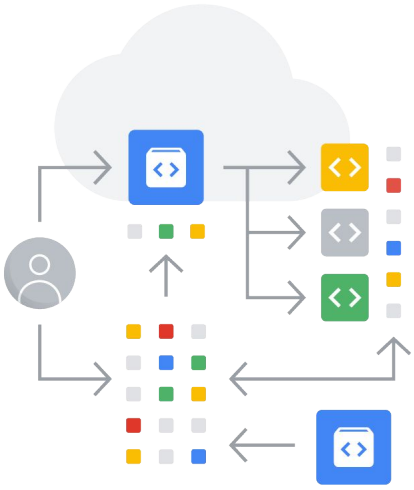
So, the answer is that ML model code typically accounts for about 5% of the overall code base.



The reason that ML models account for such a small percentage is that to keep a system running in production requires doing a lot more than just computing the model's outputs for a given set of inputs.

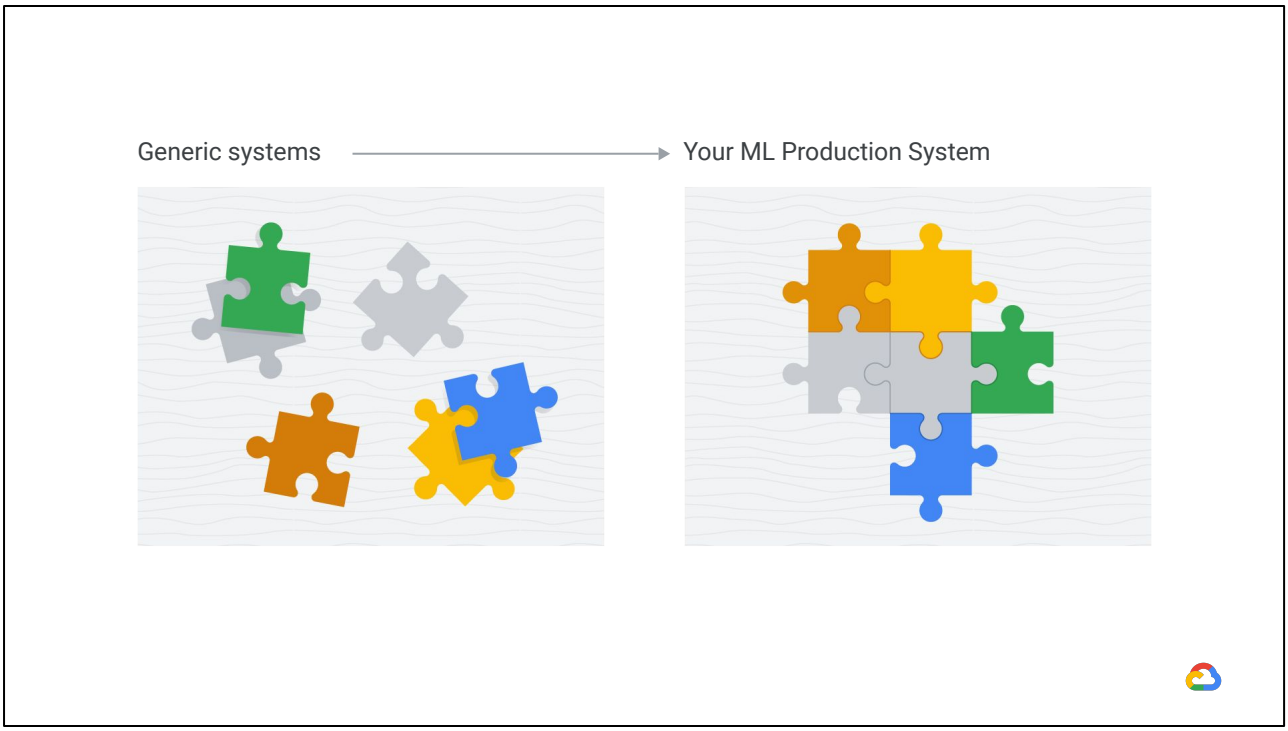
Learn how to...

- Choose a training and serving paradigm
- Serve ML models scalably
- Design an architecture from scratch

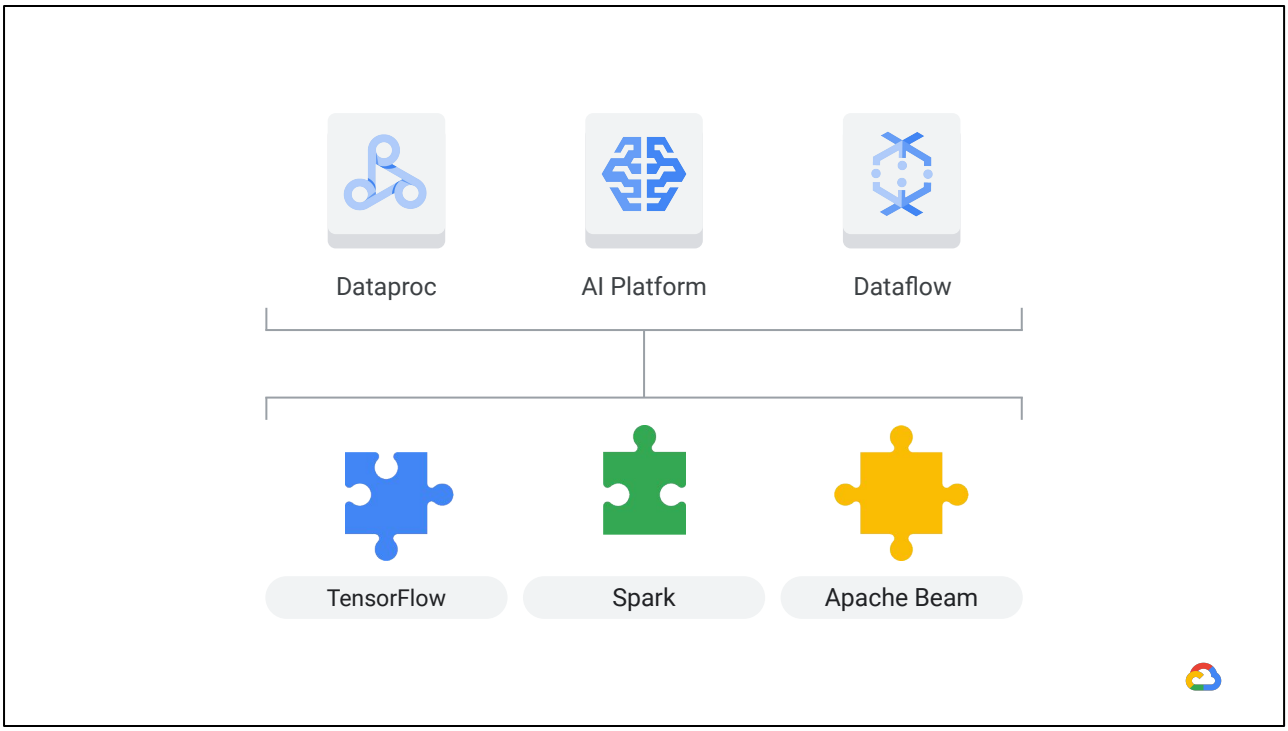


In this module, you'll see what else a production ML system needs to do and how you can meet those needs.

Upon completing this module, you should acquire the knowledge to choose an appropriate training and serving paradigm, serve ML models scalably, and design an architecture from scratch.



And while our focus is on Google Cloud, it's important that you always try and reuse generic systems when possible--many of which are open-source frameworks.

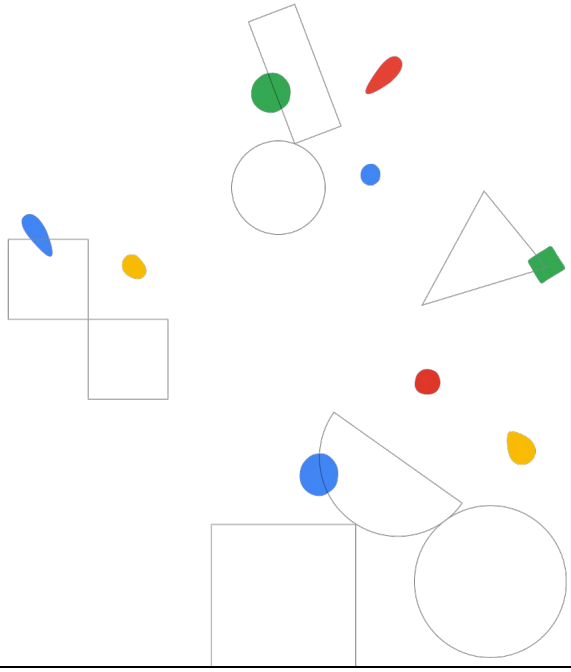


What's true of software frameworks like TensorFlow, Spark, or Apache Beam is also true of the underlying infrastructure on which you execute them.

Rather than spend time and effort provisioning infrastructure, you can use managed services such as Dataproc, AI Platform, or Dataflow to execute your Spark, TensorFlow, and Beam code.

Google Cloud

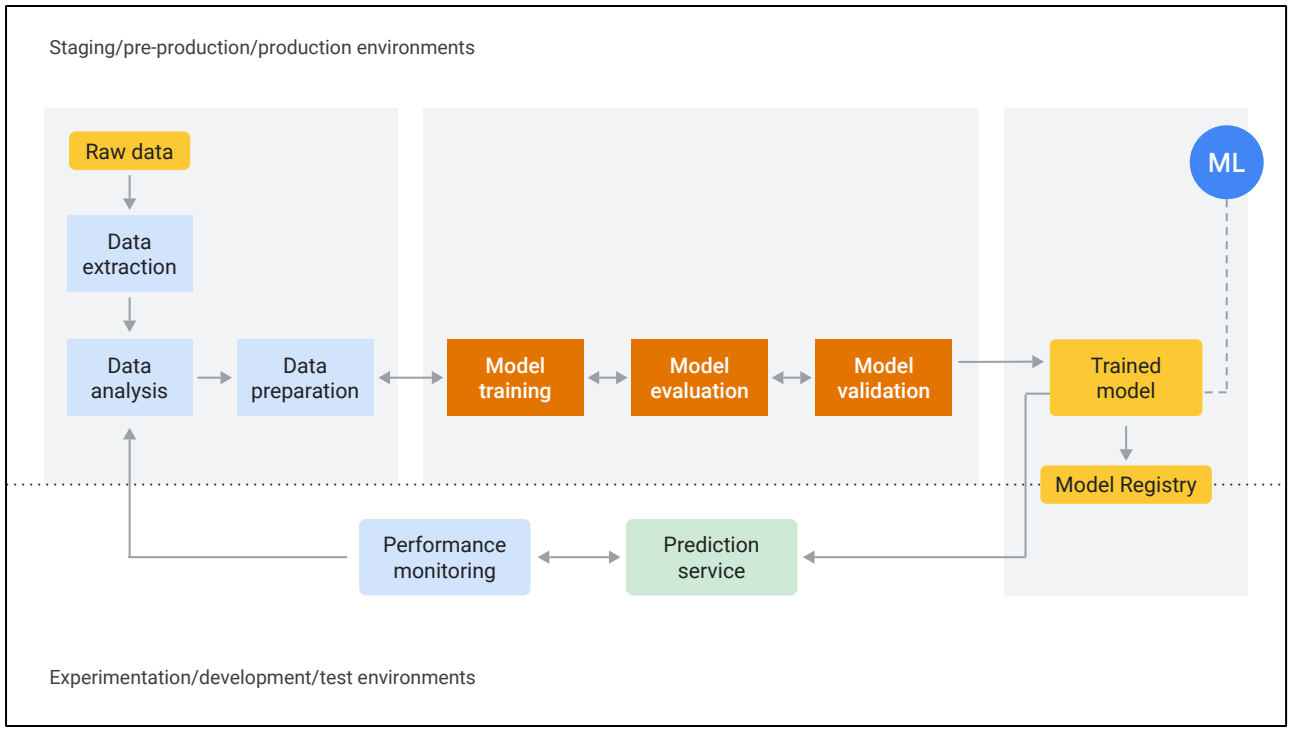
Data extraction, analysis, and preparation



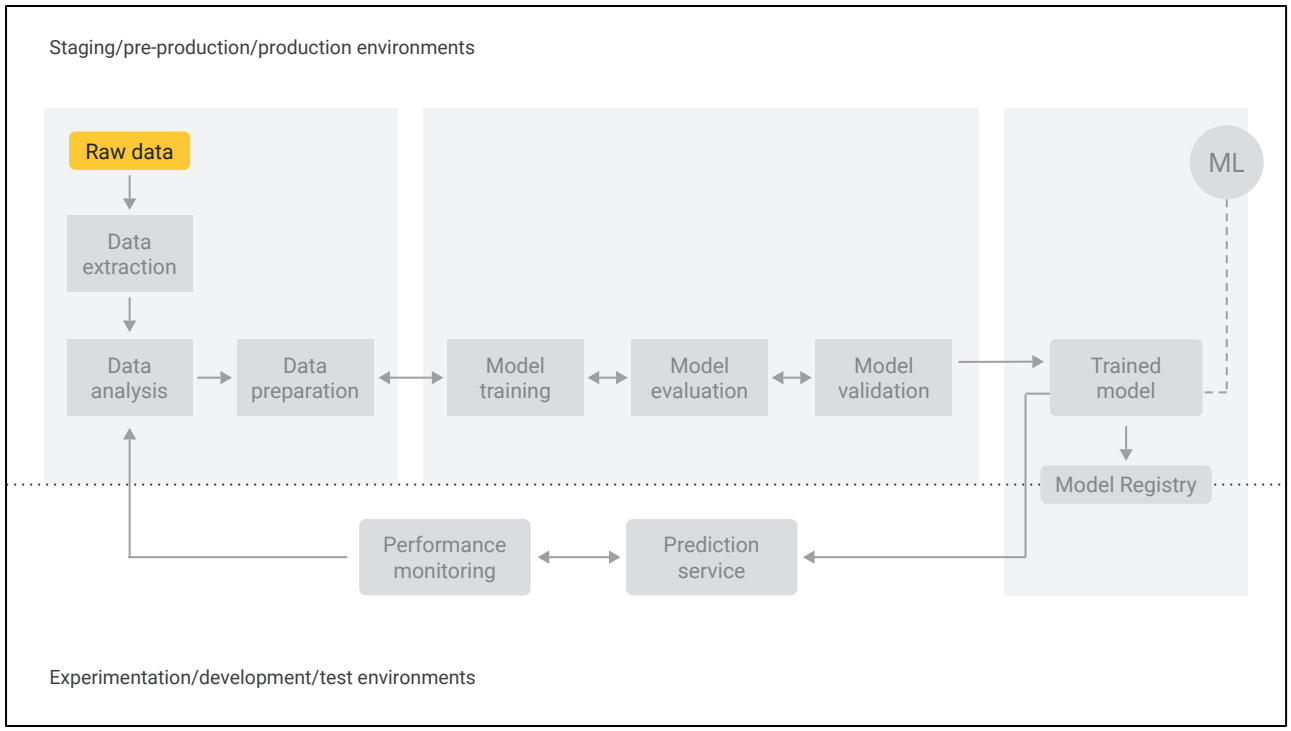
Module 01

Architecting production ML systems

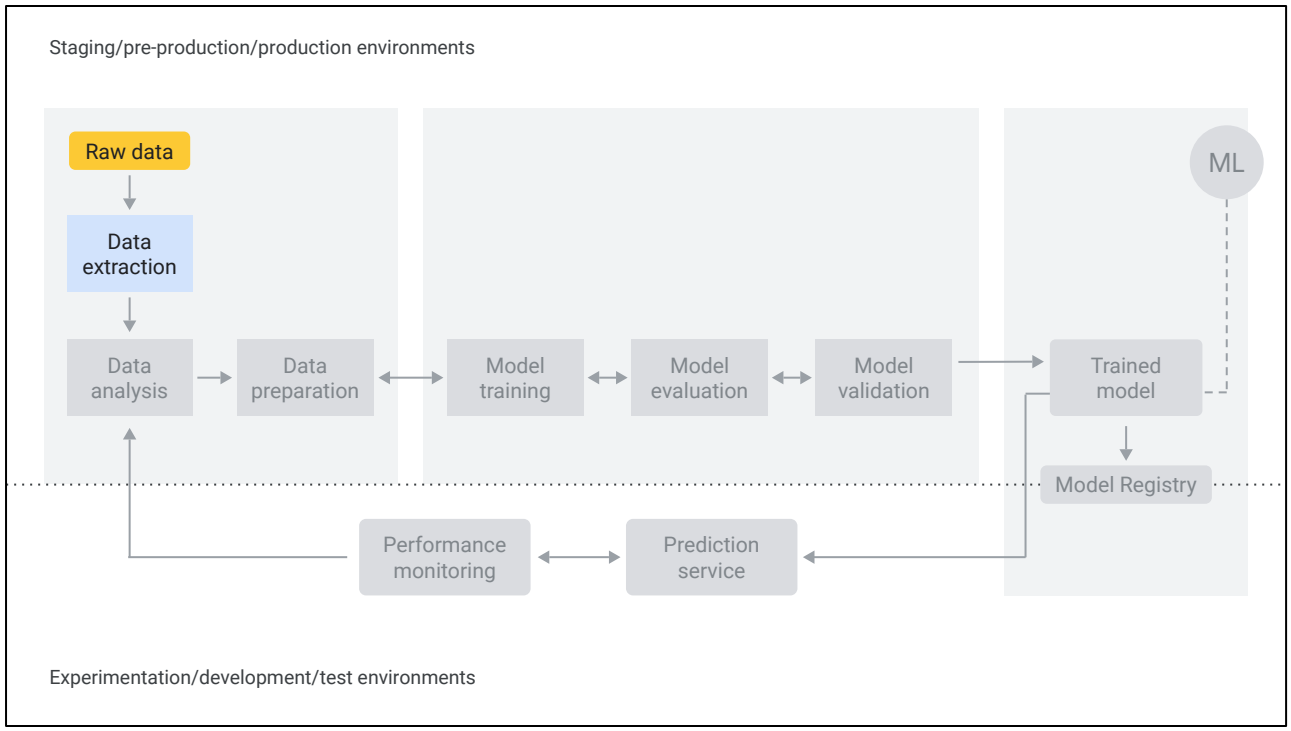
After you define the business use case and establish the success criteria, the process of delivering an ML model to production typically involves several steps, which can be completed manually or by an automated pipeline.



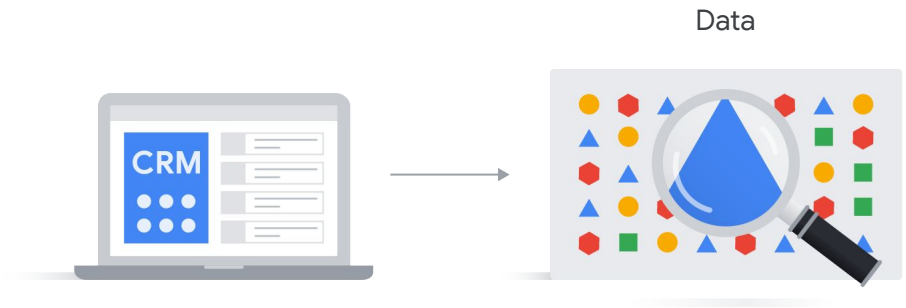
The first three steps deal with data.



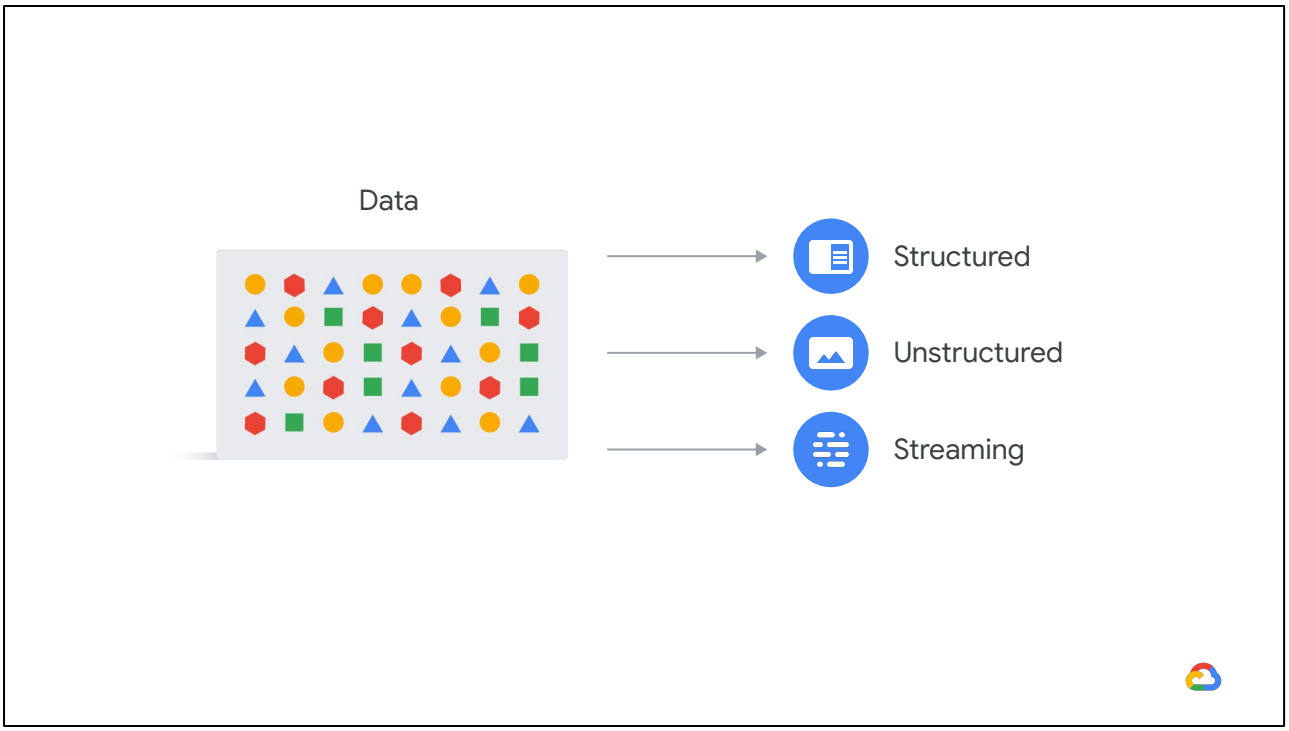
Data needs to be ingested, which means it's extracted from a **raw data** source.



With **data extraction**, you retrieve data from various sources. Those sources can be “streaming”, in “real-time”, or “batch”.

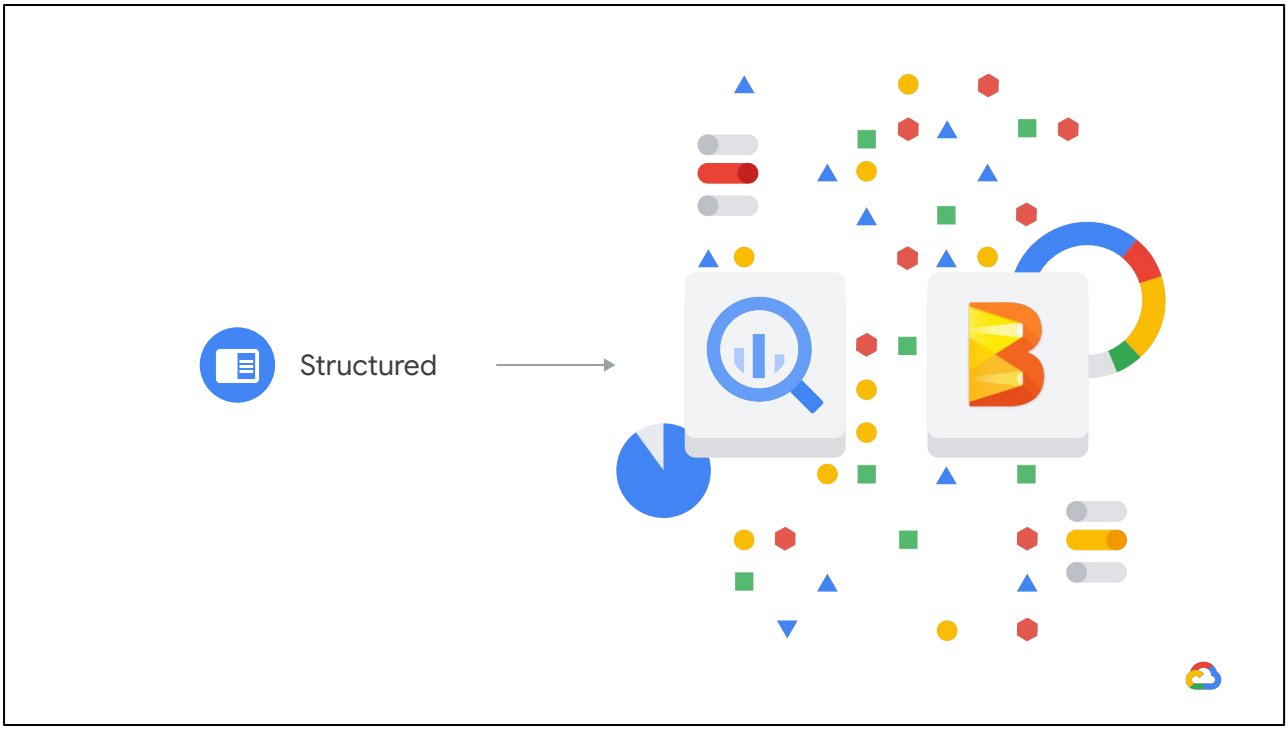


For example, you may extract data from a customer relationship management system, or CRM, to analyze customer behavior.



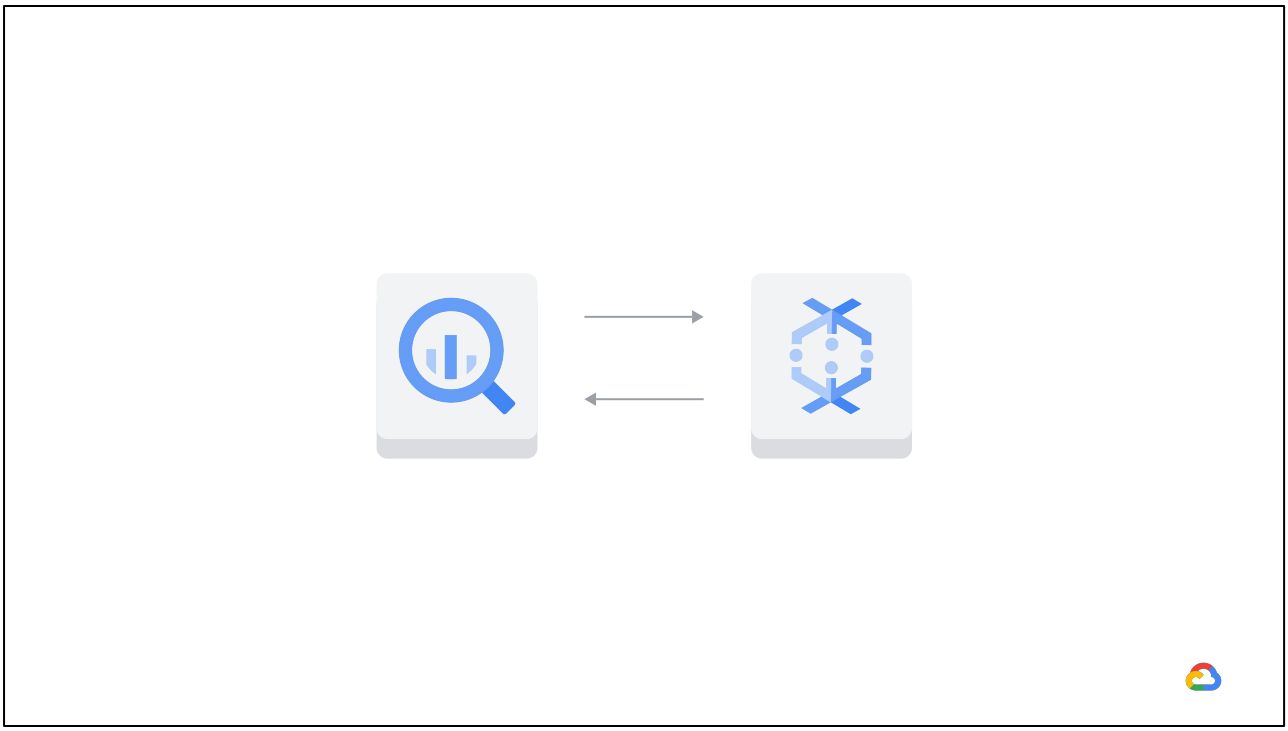
This data may be “structured”, where the file type is a .csv, .txt. JSON, or . XML format, or, you may have an “unstructured” data source where you have images of your customers, or text comments from chat sessions with your customers.

You may have to extract “streaming” data from your company’s transportation vehicles that are equipped with sensors that transmit data in real time.

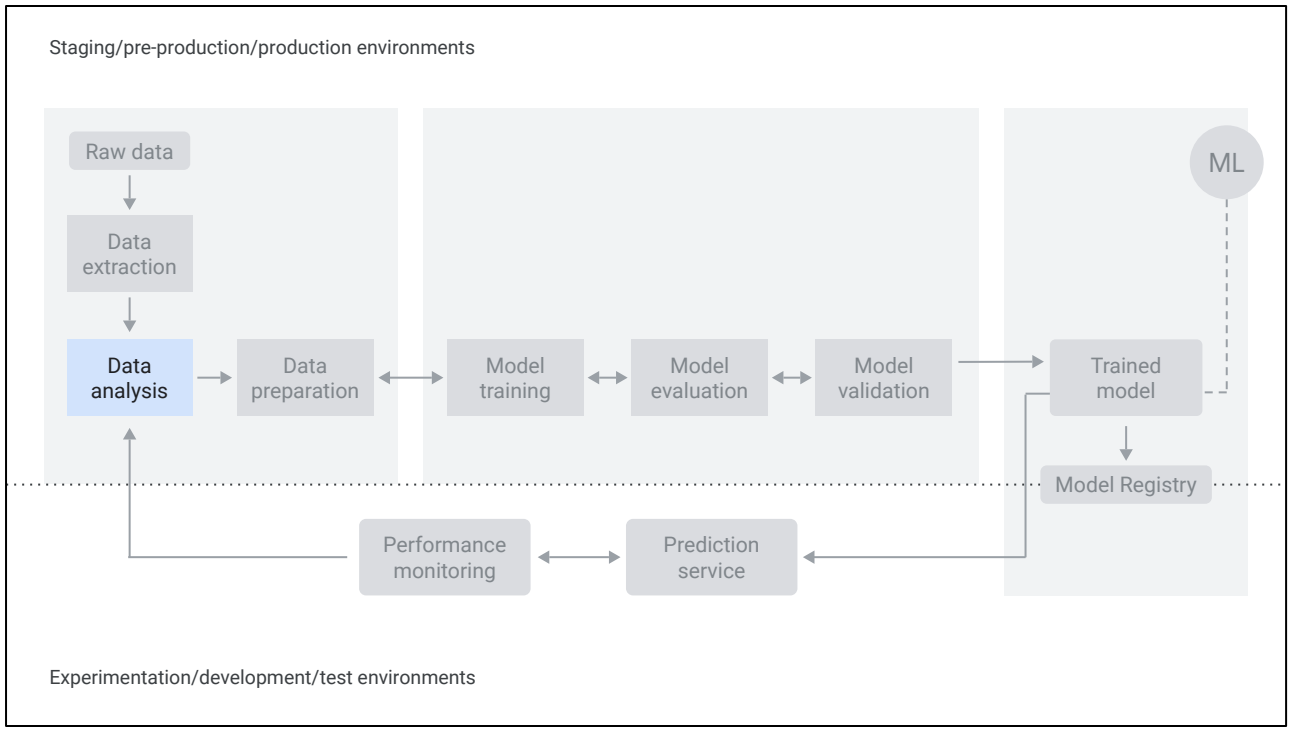


If the data you want to train your model on or get predictions for is structured, you might retrieve it from a data warehouse -- such as BigQuery.

Or, you can use Apache Beam's IO module.

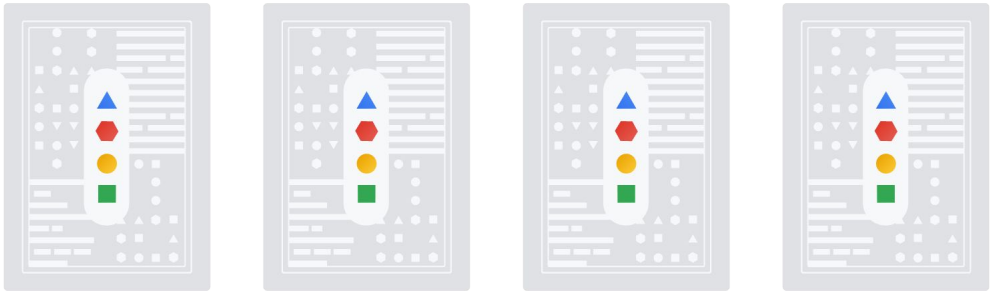


In this Dataflow example we're loading data from BigQuery, calling predict on every record, and then writing the results back into BigQuery.

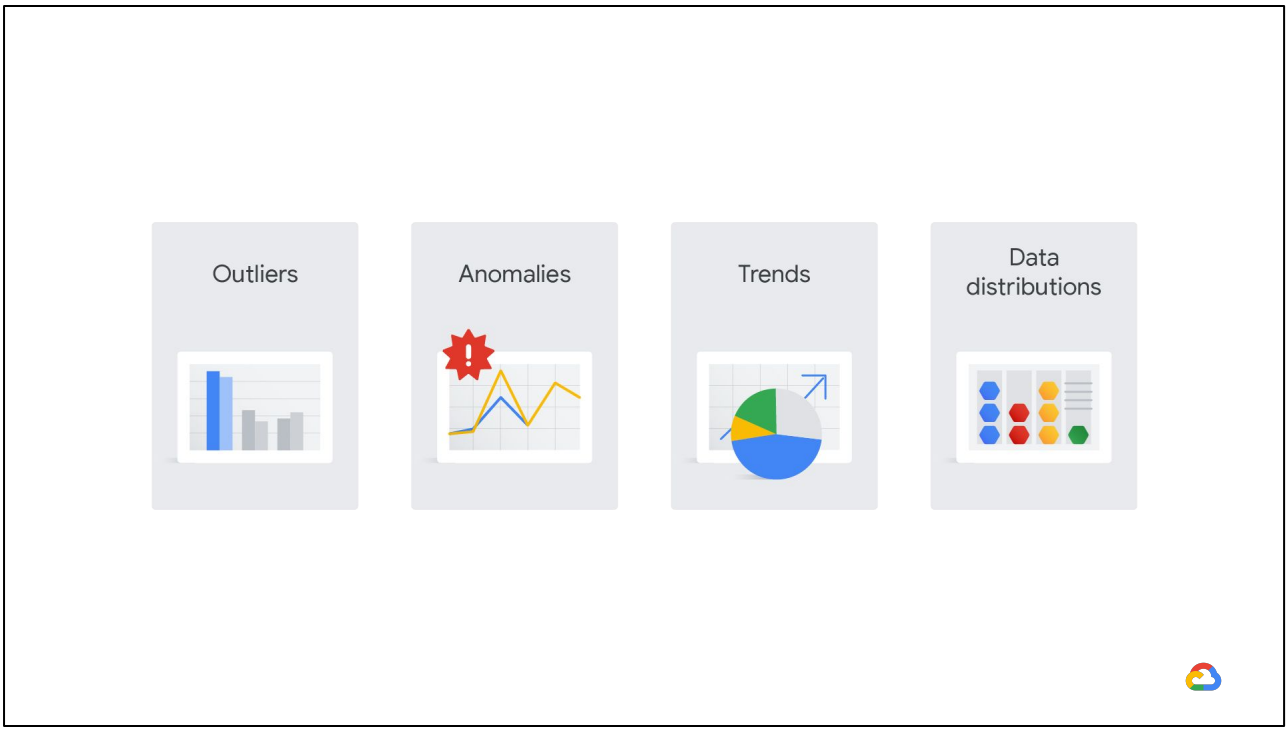


In **data analysis**, you analyze the data you've extracted.

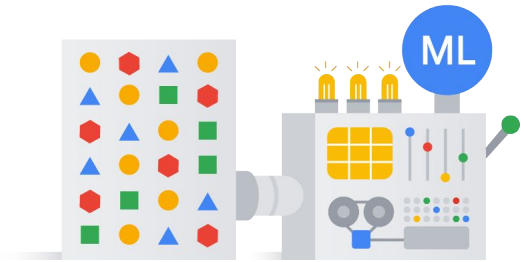
For example, you can use exploratory data analysis (or EDA).



This involves using graphics and basic sample statistics to explore your data,



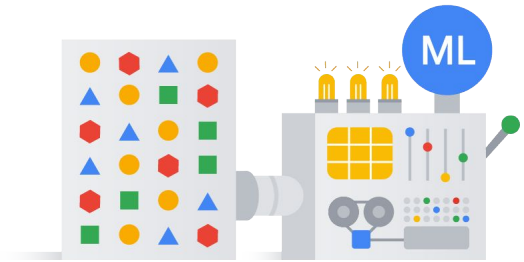
such as looking for outliers or anomalies, trends, and data distributions.



Number	Product Name
112	Blue T-Shirt
231	Dog Frisbee
1333	Mobile Phone Charger

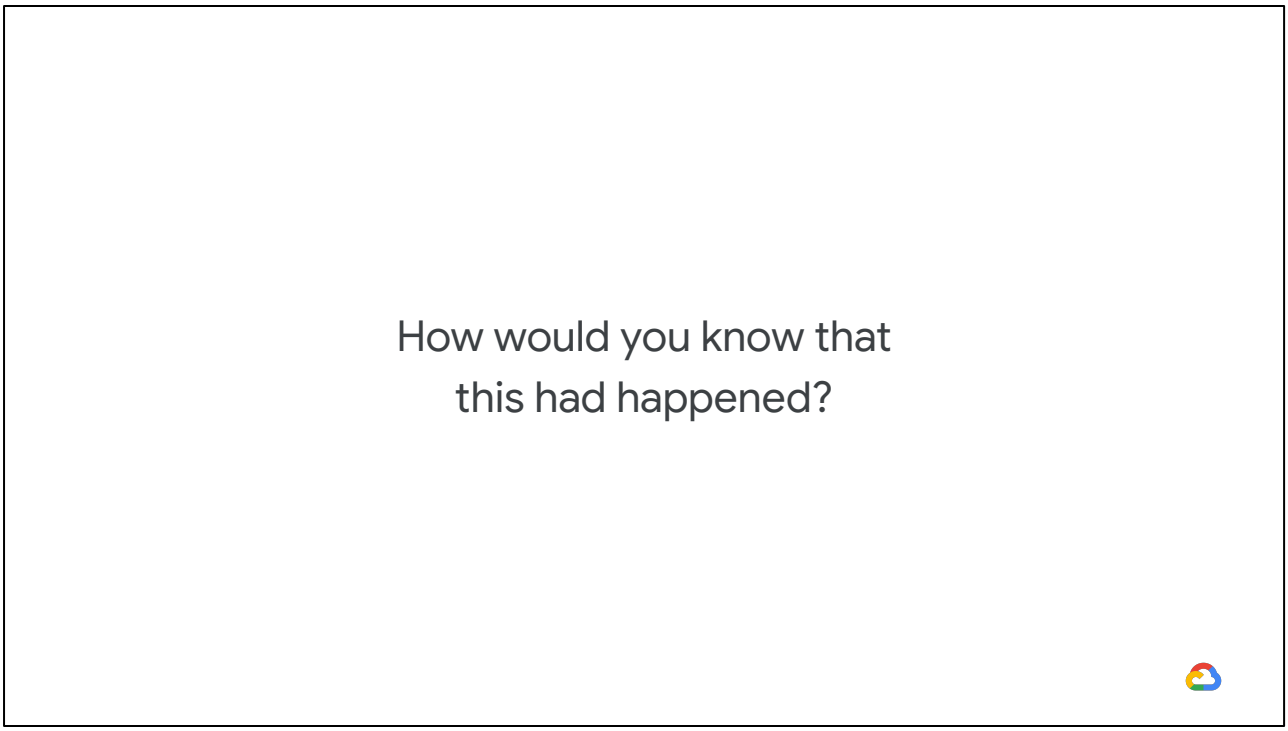
It may not be apparent how changes in the distribution of your data could affect your model, so let's consider a scenario.

In this scenario, an upstream data source encodes a categorical feature using a number, such as a product number.

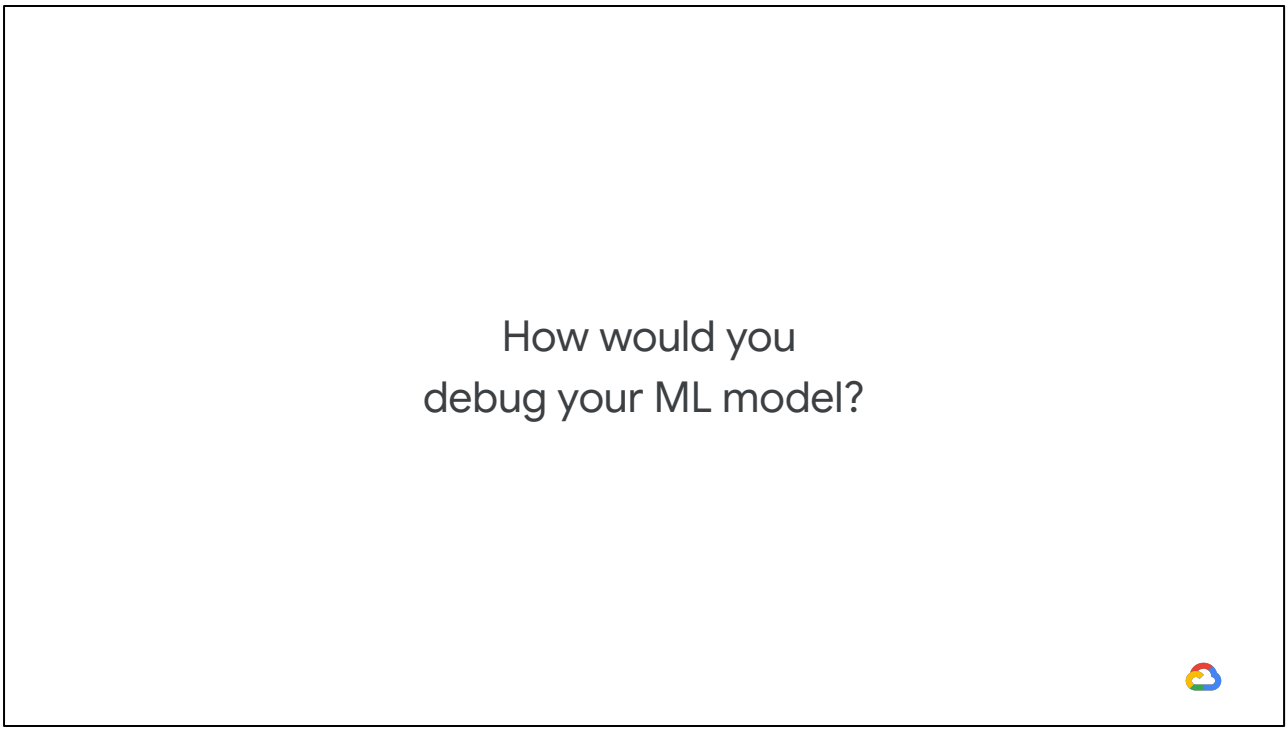


Number	Number	Product Name
112	231	Blue T-Shirt
231	231231	Dog Frisbee
1333	112	Mobile Phone Charger

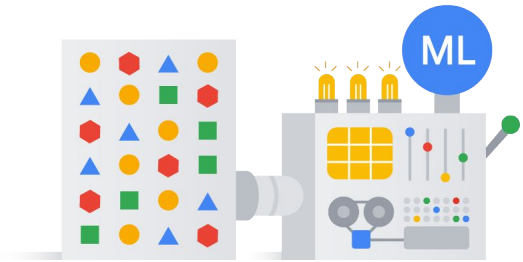
One day, the product numbering convention changes and now the customer uses a totally different mapping, using some old numbers and some new numbers.



How would you know that this had happened?



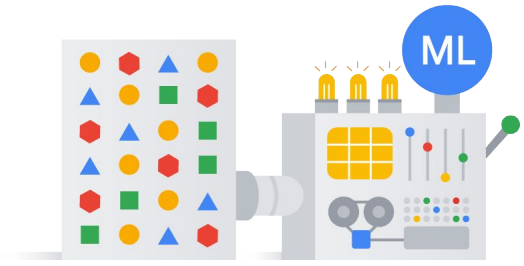
How would you debug your ML model?

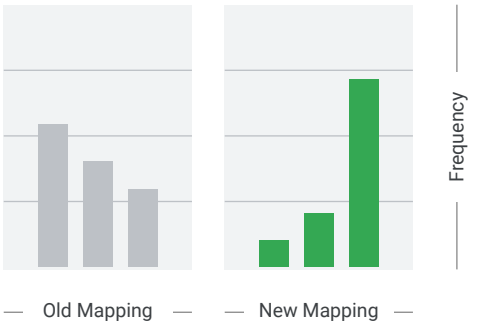


Number	Number	Product Name
112	231	Blue T-Shirt
231	231231	Dog Frisbee
1333	112	Mobile Phone Charger

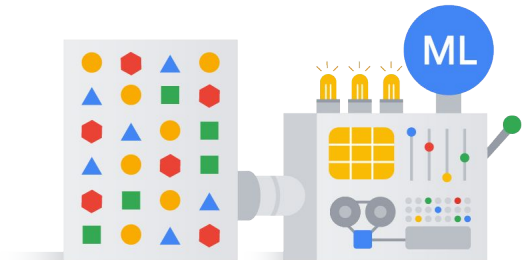
The output of your model would tell you if there's a drop in performance, but it won't tell you why. The raw inputs themselves would appear valid because we're still getting numbers.

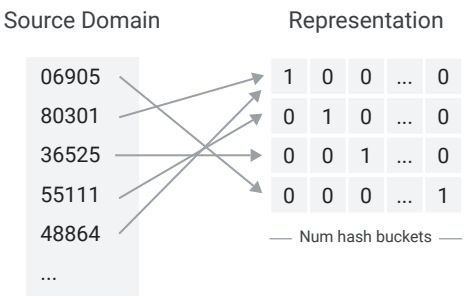
In order to recognize this change, you would need to look at changes in the distribution of your inputs.



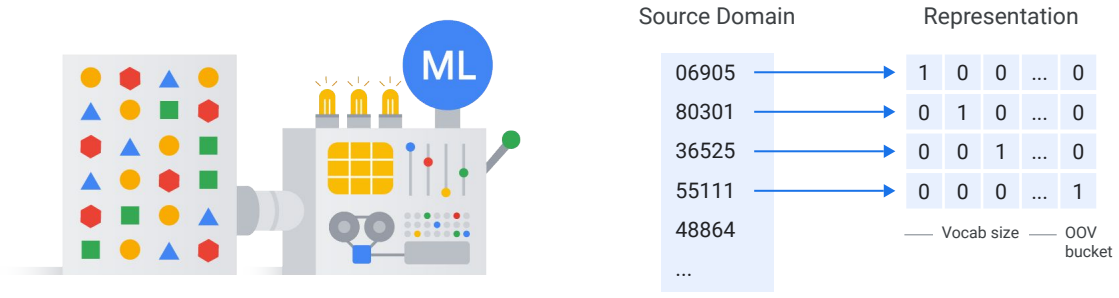


In doing so, you might find that while earlier, the most commonly occurring value might have been a 4, in the new distribution, 4 might never even occur and the most commonly occurring value might be a 10.



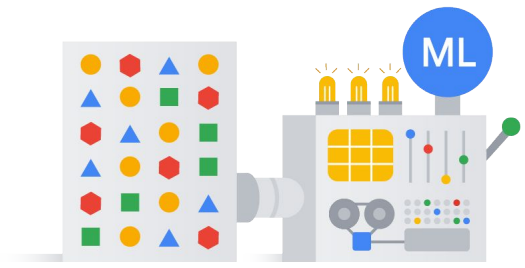


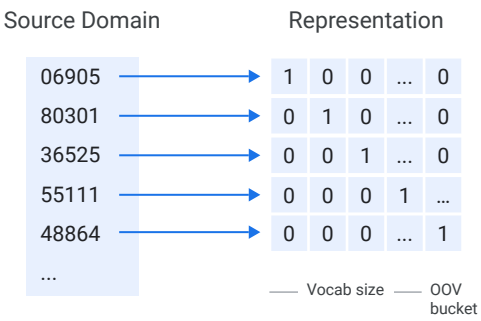
Depending on how you implemented your feature columns, these new values might be mapped to one component of a one-hot encoded vector or to many components.



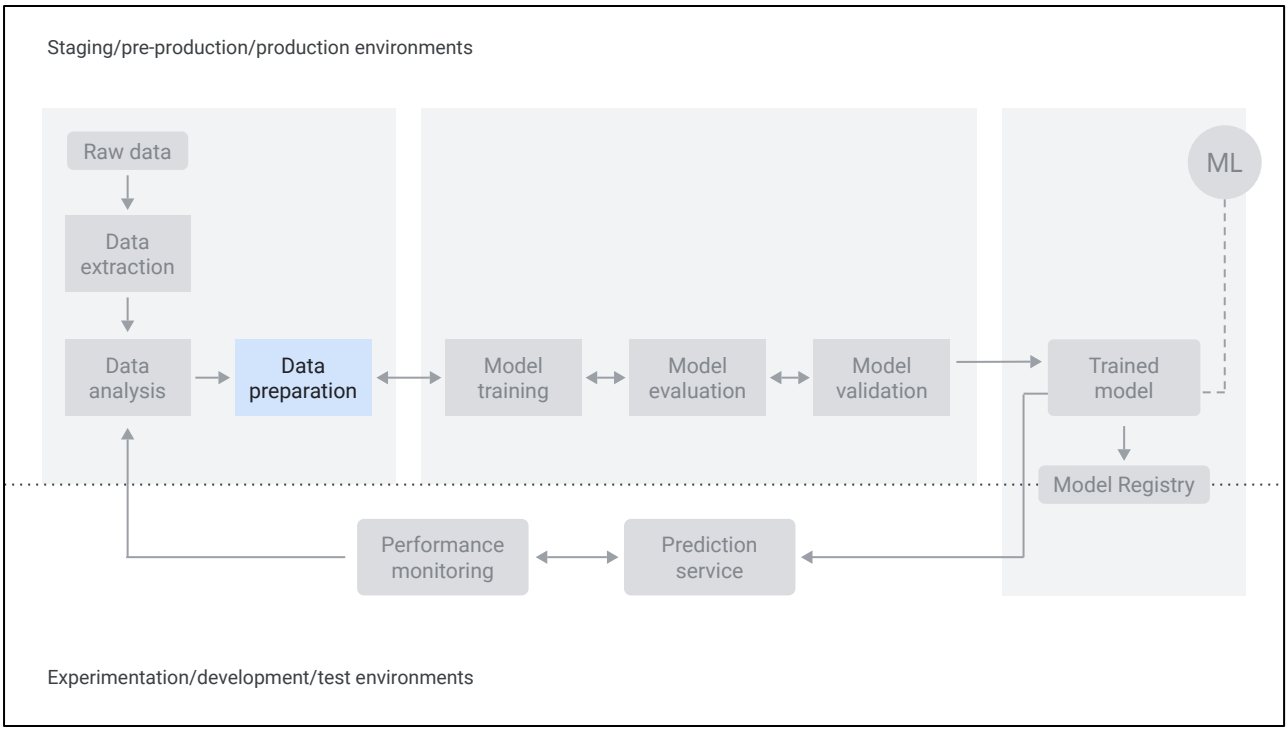
If, for example, you used a categorical column with a hash bucket, the new values would be distributed according to the hash function, and so one hash bucket might now get more and different values than before.

If you used a vocabulary, then the new values would map to OOV buckets.

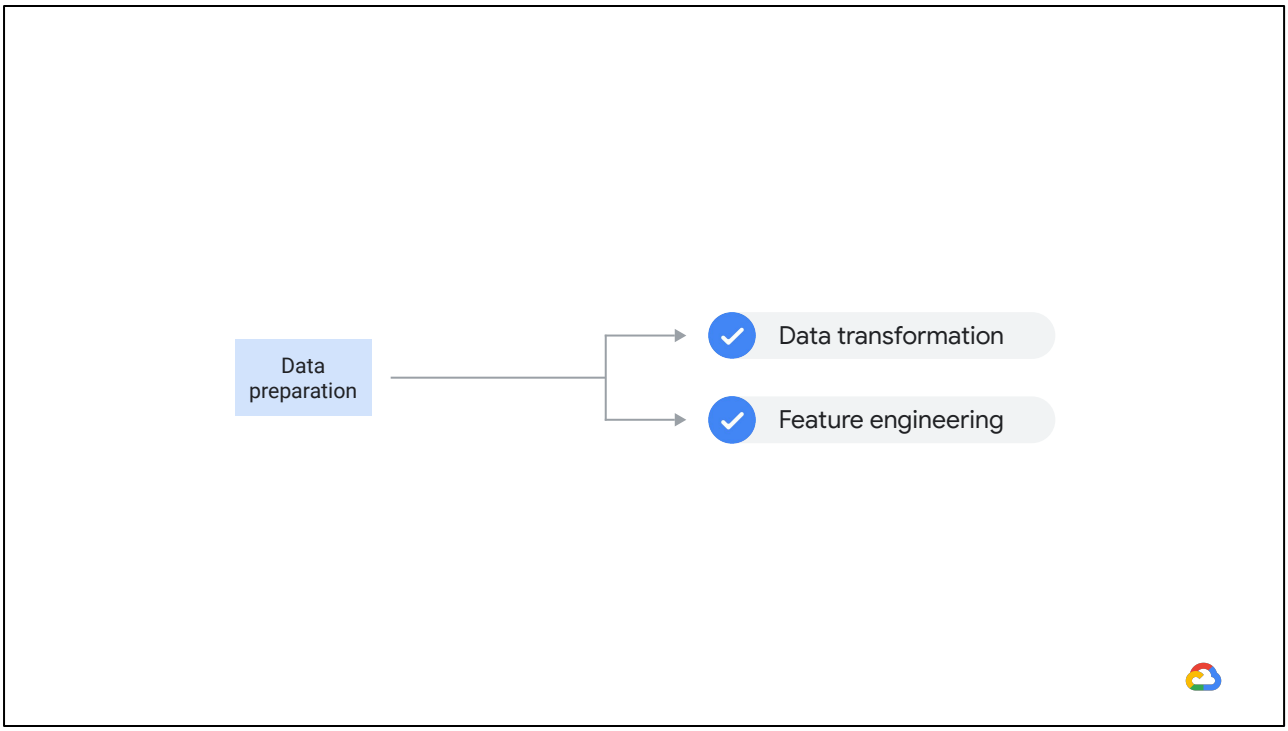




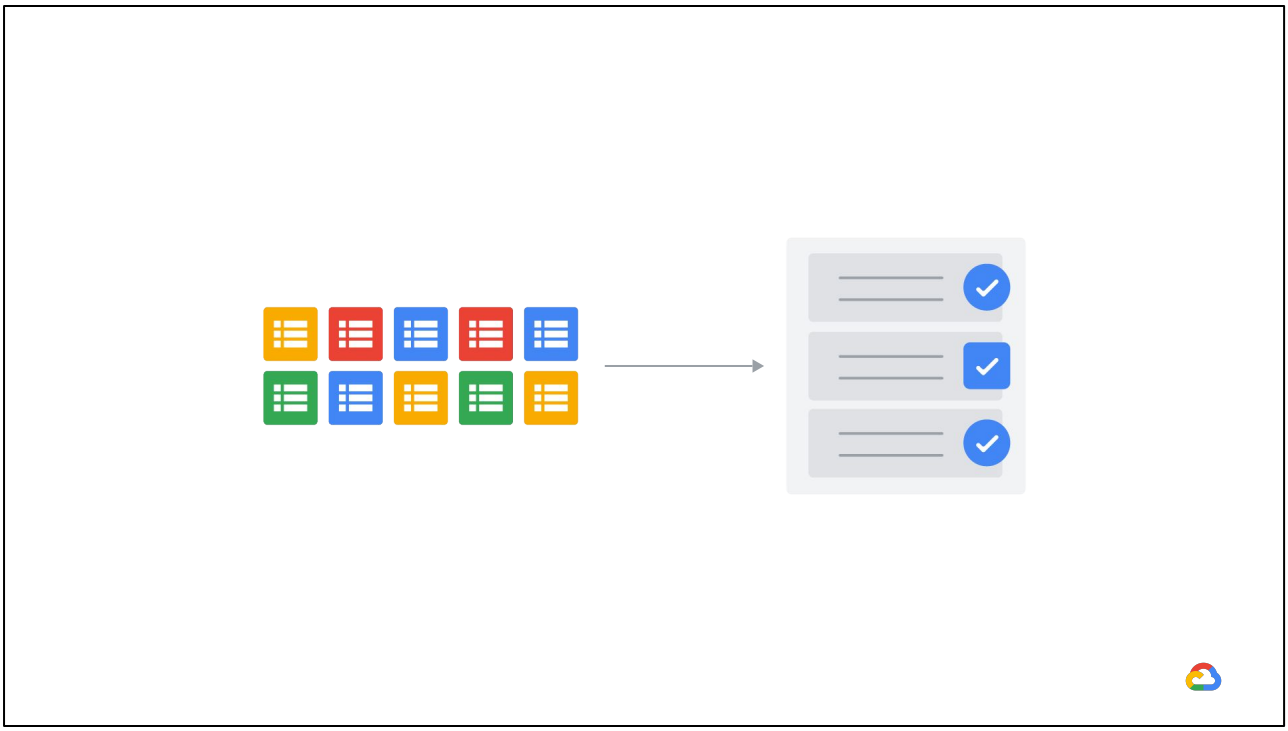
But what's important is that, for a given tensor, its relationship to the label before, and its relationship to the label now, are likely to be very different.



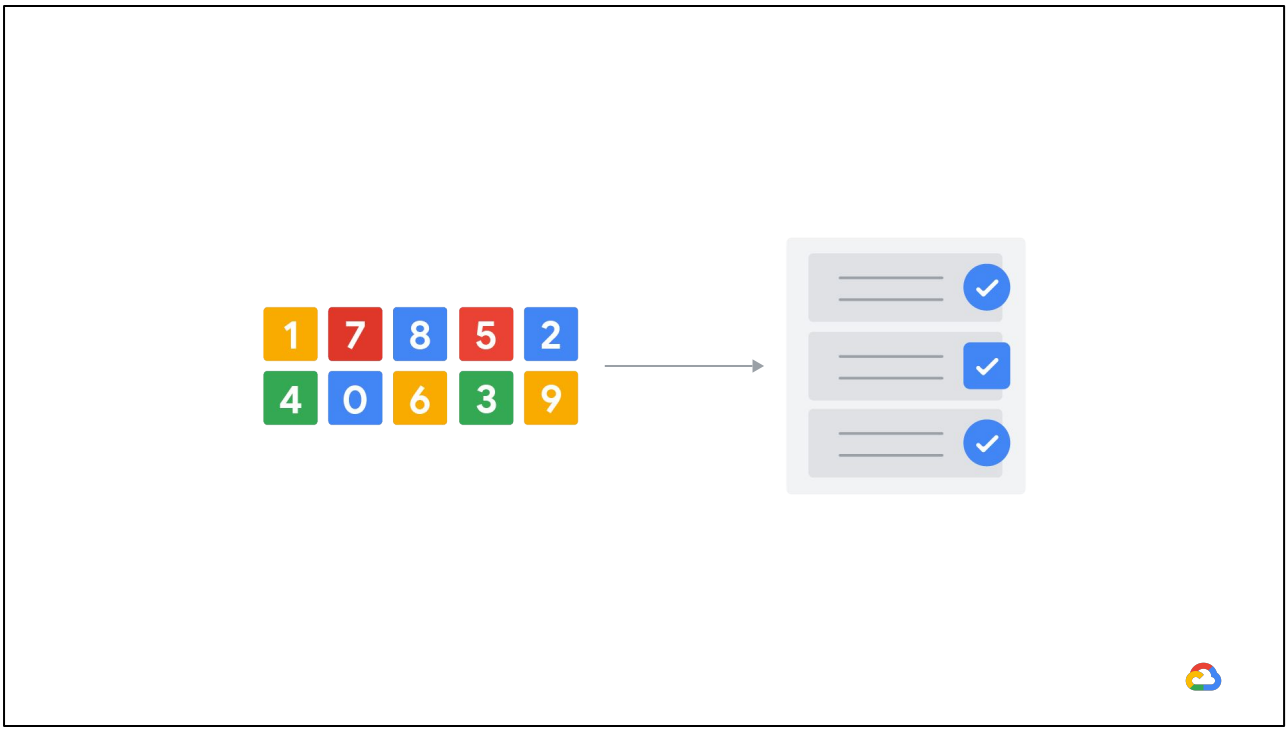
So, after you've extracted and analyzed your data, the next step in the process is **data preparation**.



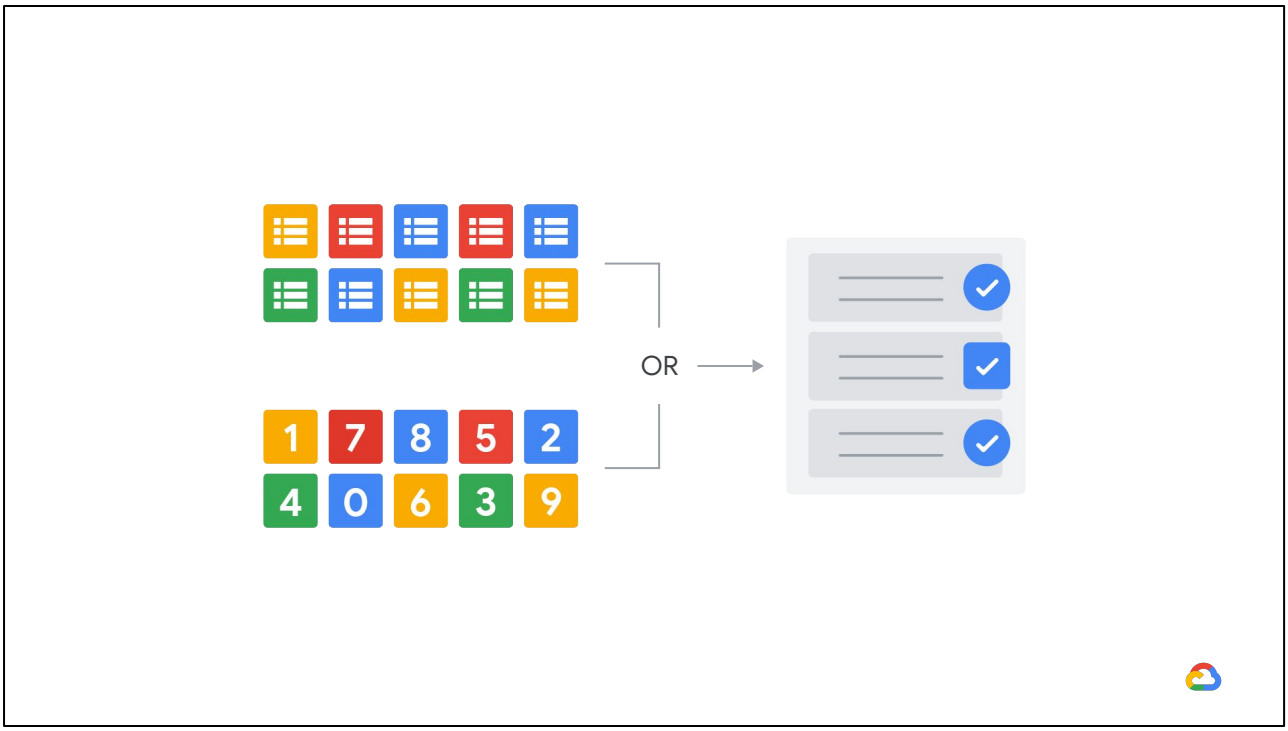
Data preparation includes data transformation and feature engineering, which is the process of changing, or converting, the format, structure, or values of data you've extracted, into another format or structure.



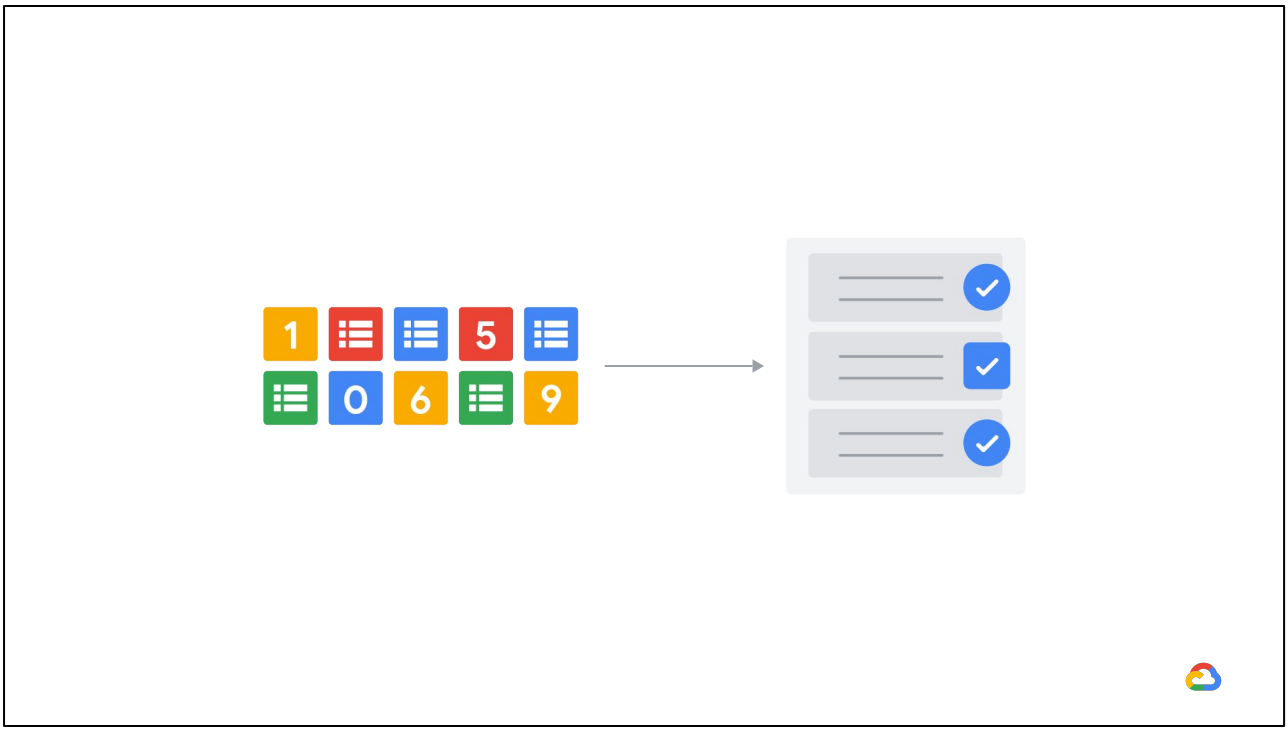
Most ML models require categorical data



to be in a numerical format,



but some models work either with numeric or categorical features,



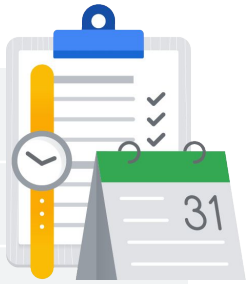
while others can handle mixed-type features.

Preprocessing for dates using SQL in BigQuery ML

```
EXTRACT(DAYOFWEEK FROM pickup_datetime) AS dayofweek,

EXTRACT(HOUR FROM pickup_datetime) AS hourofday,

CONCAT(CAST(EXTRACT(DAYOFWEEK FROM pickup_datetime) AS STRING),
CAST(EXTRACT(HOUR FROM pickup_datetime) AS STRING)) AS hourofday,
```



For example, here are three types of preprocessing for dates using SQL in BigQuery ML, where we are:

- Extracting the parts of the date into different columns: Year, month, day, etc.
- Extracting the time period between the current date and columns in terms of years, months, days, etc.
- And extracting some specific features from the date: Name of the weekday, weekend or not, holiday or not, etc.

Example

	dayofweek	hourofday
1	Week 6	4 AM
2	Week 5	3 AM
3	Week 4	2 AM
4	Week 7	1 AM
5	Week 3	12 AM

Data Studio output: from EXTRACT dates/time queries

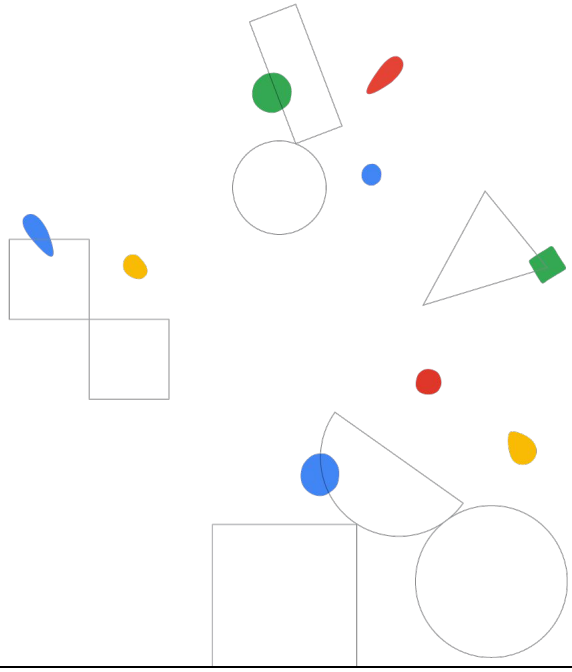
Now here is an example of the dayofweek and hourofday queries extracted using SQL and visualized as a table in Data Studio.

For all non-numeric columns, other than TIMESTAMP, BigQuery ML performs a one-hot encoding transformation

Please note that for all non-numeric columns, other than TIMESTAMP, BigQuery ML performs a one-hot encoding transformation. This transformation generates a separate feature for each unique value in the column.

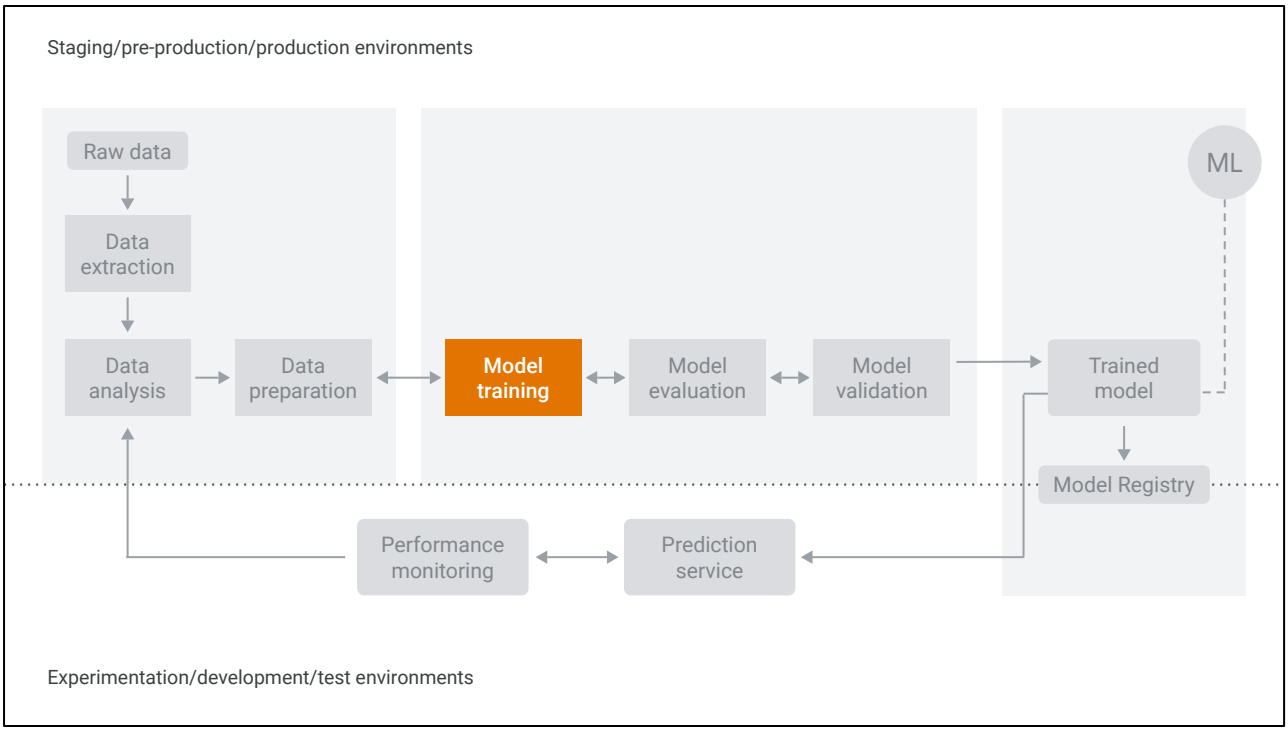
Google Cloud

Model training, evaluation, and validation

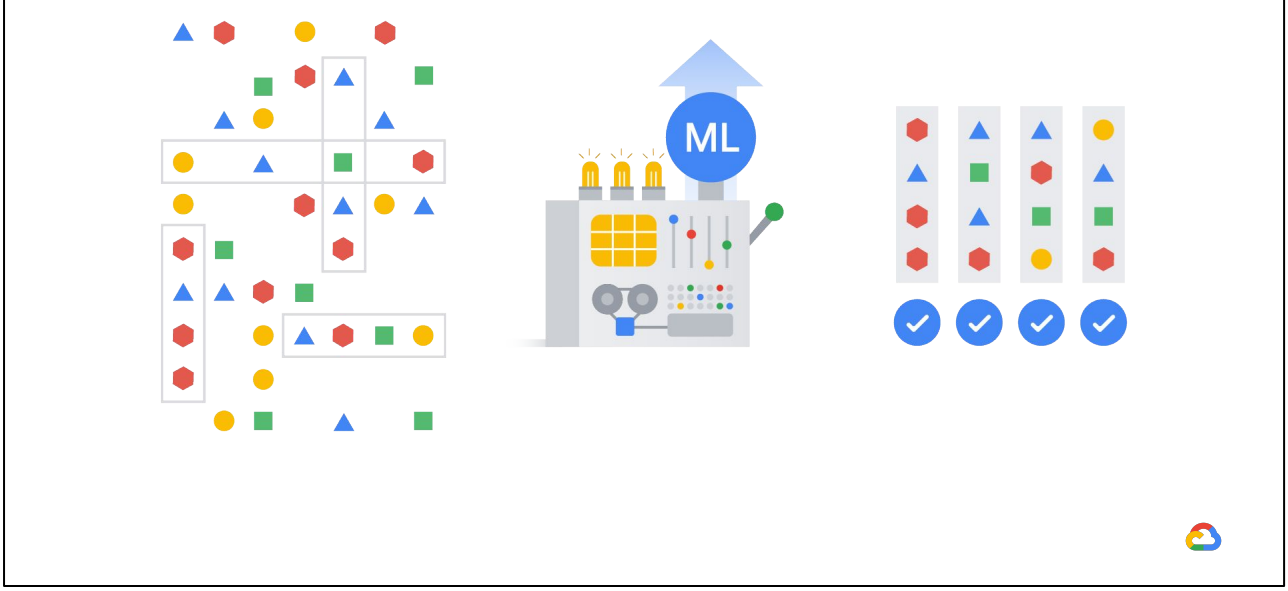


Module 01

Architecting production ML systems

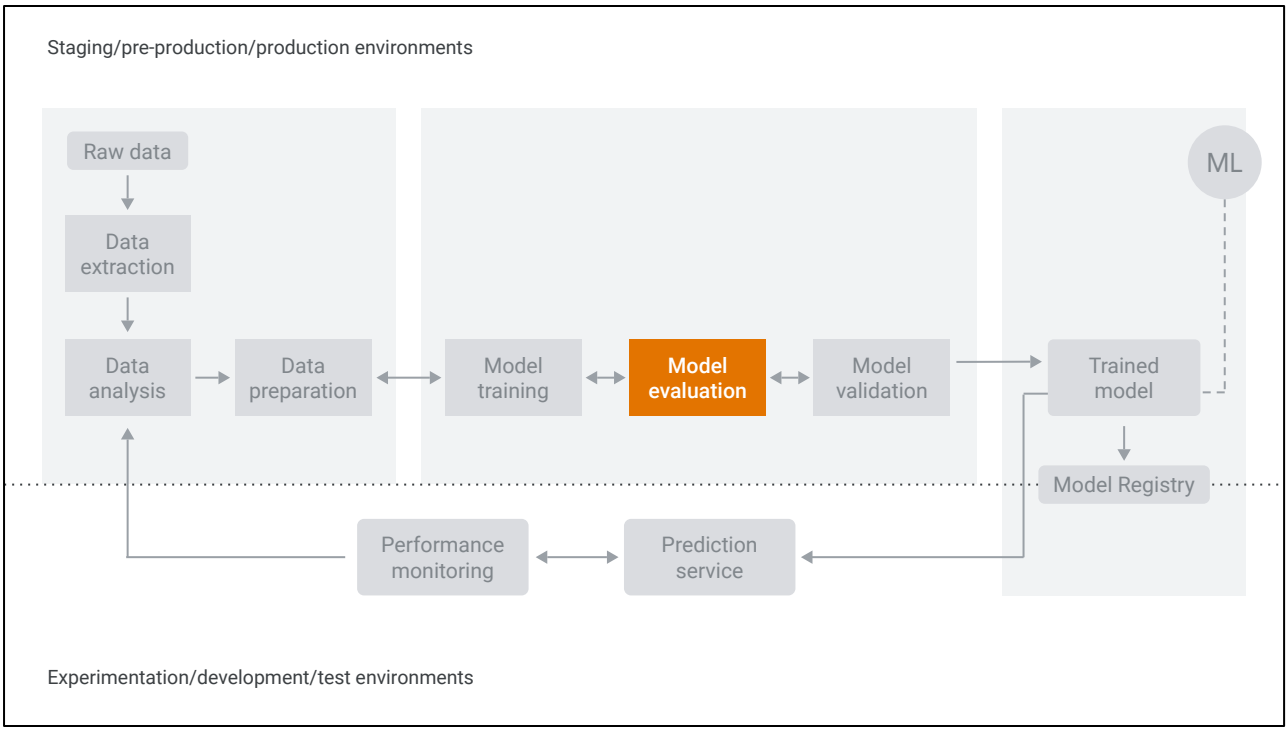


The next step in our workflow is choosing a model that you'll train. In **model training**, you implement different machine learning algorithms with the prepared data to train various ML models.

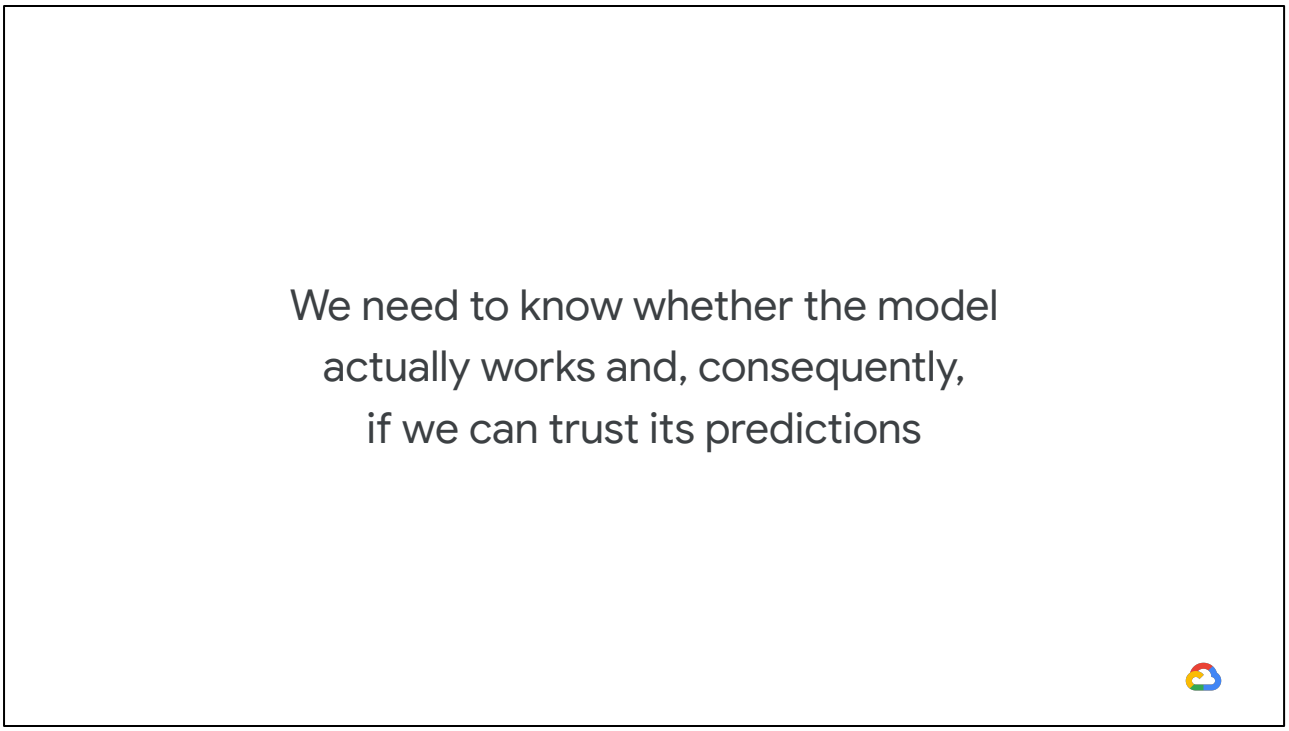


Essentially, model training is the process of feeding an ML algorithm with data to help identify and learn good values for the feature set.

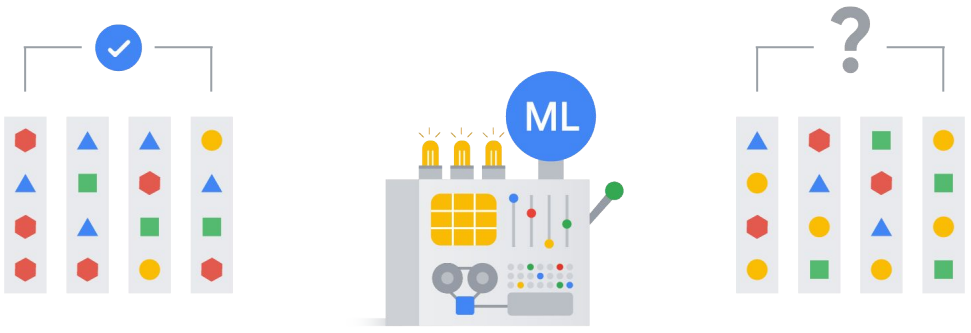
So, you use your data to incrementally improve your model's predictive ability.



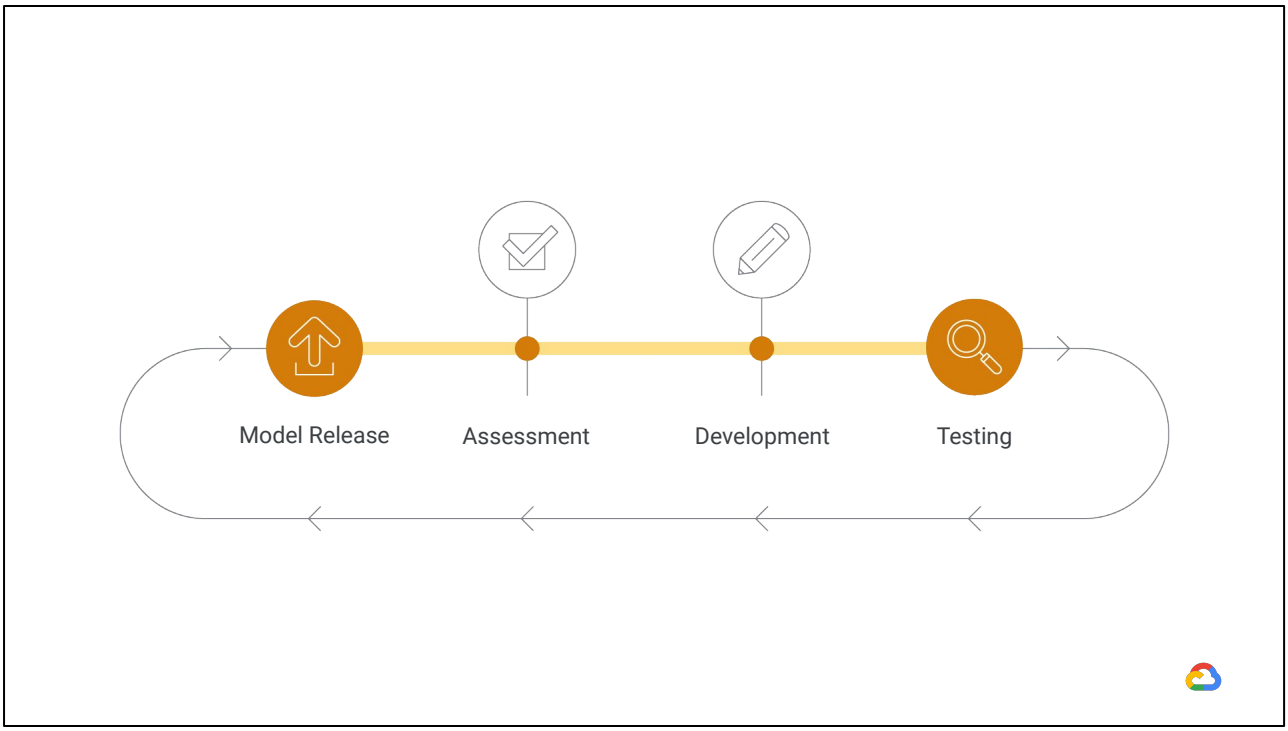
Model evaluation aims to estimate the generalization accuracy of a model on future, unseen, or out-of-sample data. While training a model is a key step in the pipeline, how the model generalizes on unseen data is an equally important aspect that should be considered in every machine learning pipeline.



This means that we need to know whether the model actually works and, consequently, if we can trust its predictions.



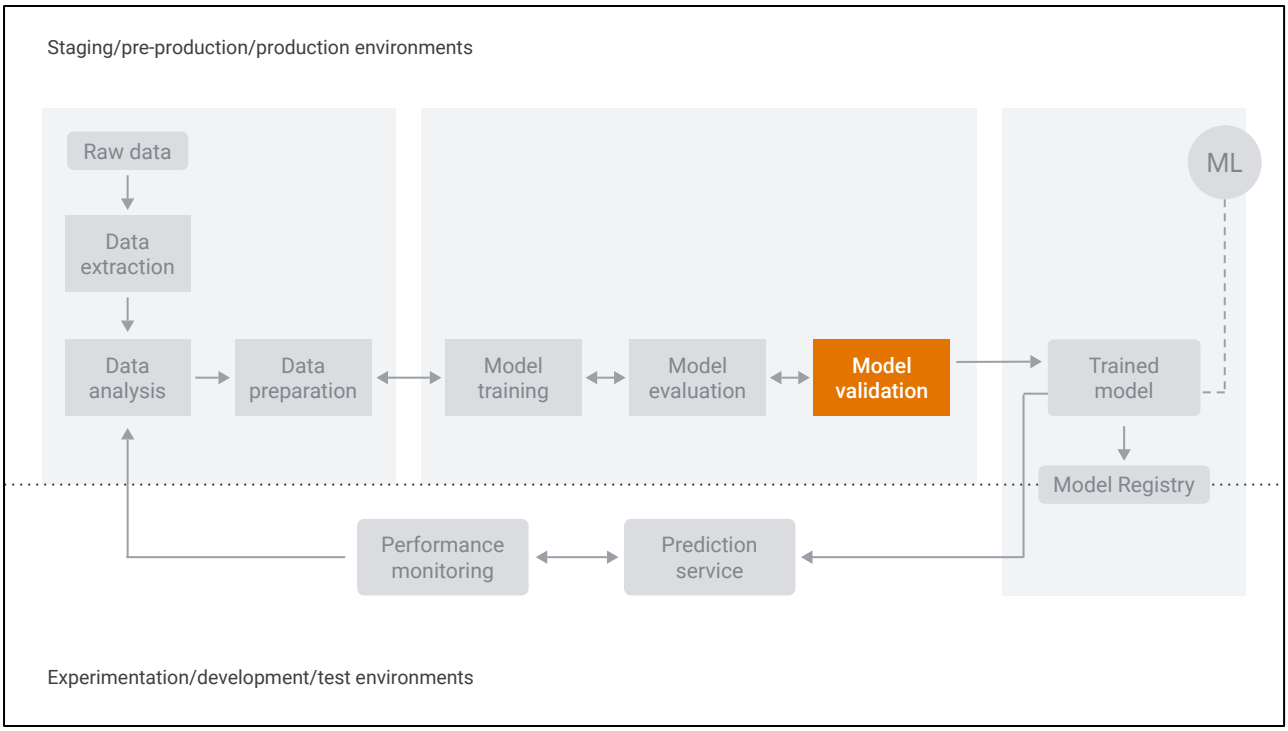
Could the model be merely memorizing the data it's fed with, and therefore unable to make good predictions on future samples, or samples that it hasn't seen before, such as the test set?



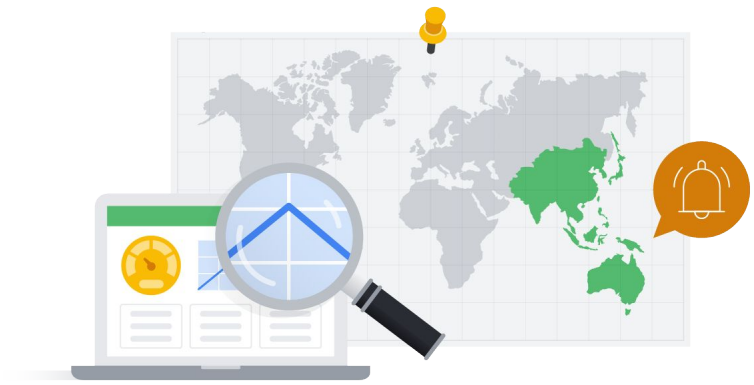
Model evaluation consists of a person or group of people evaluating or assessing the model with respect to some business-relevant metric, like AUC (area under the curve) or cost-weighted error. If the model meets their criteria, then the model is moved from the assessment phase to development.

For example, in the development phase, you may want to make modifications to hyperparameter values to increase the model's performance, which correlates to improvements in evaluation results.

After development, the model is then ready for a live experiment or real-world test of the model.

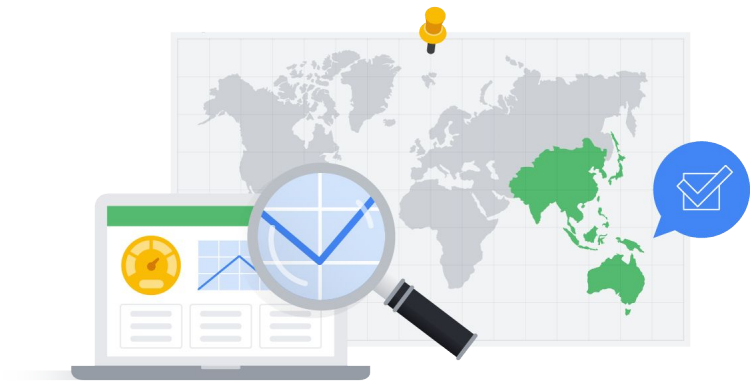


In contrast to the Model Evaluation component, which is performed by humans, the **Model Validation** component evaluates the model against fixed thresholds and alerts engineers when things go awry. One common test is to look at performance by slice.



Let’s say, for example, business stakeholders care strongly about a particular geographic market region.

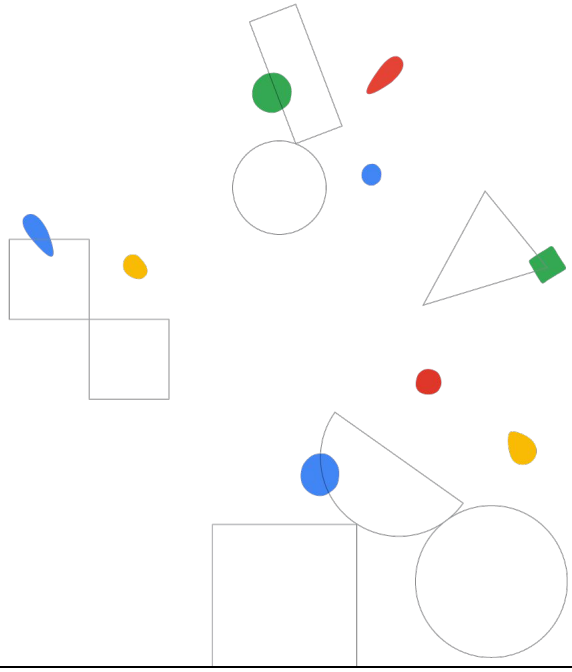
An alert can be set to notify engineers when the accuracy by country begins to skew downward.



The model evaluation and validation components have one responsibility: to ensure that the models are ‘good’ before moving them into a production environment.

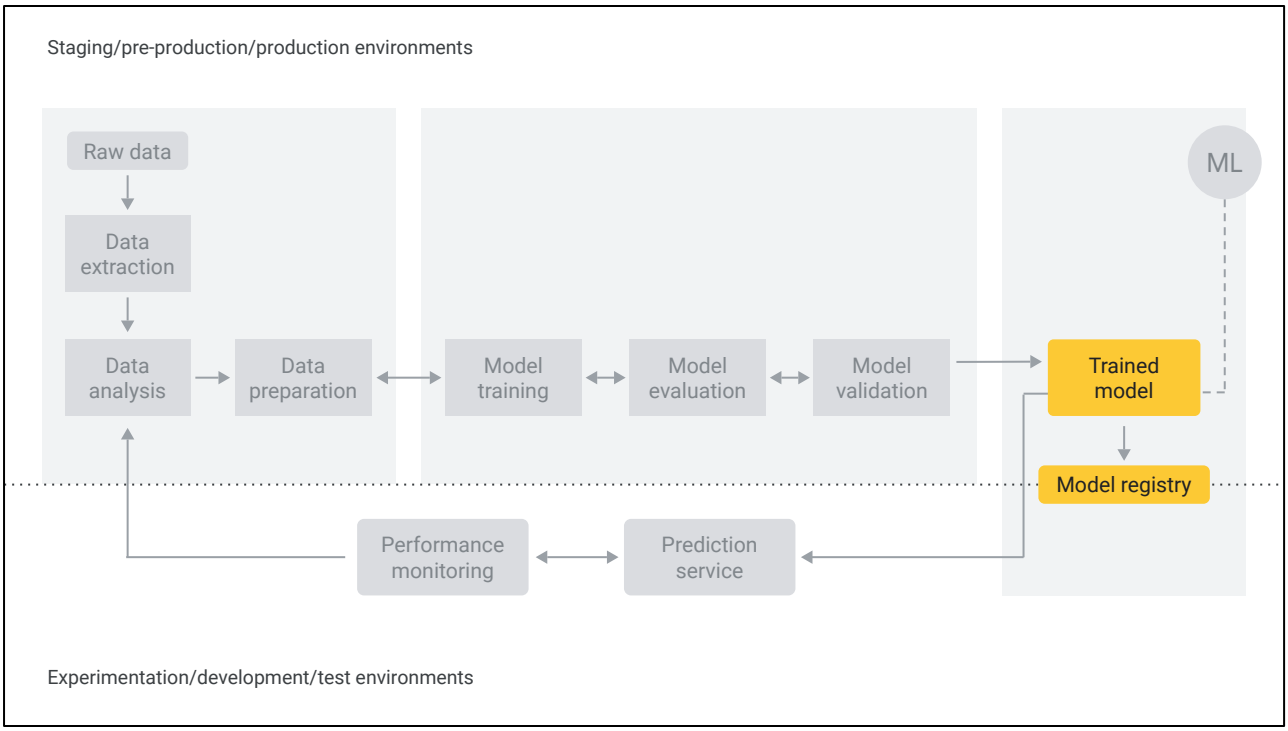
Google Cloud

Trained model, prediction service, and performance monitoring

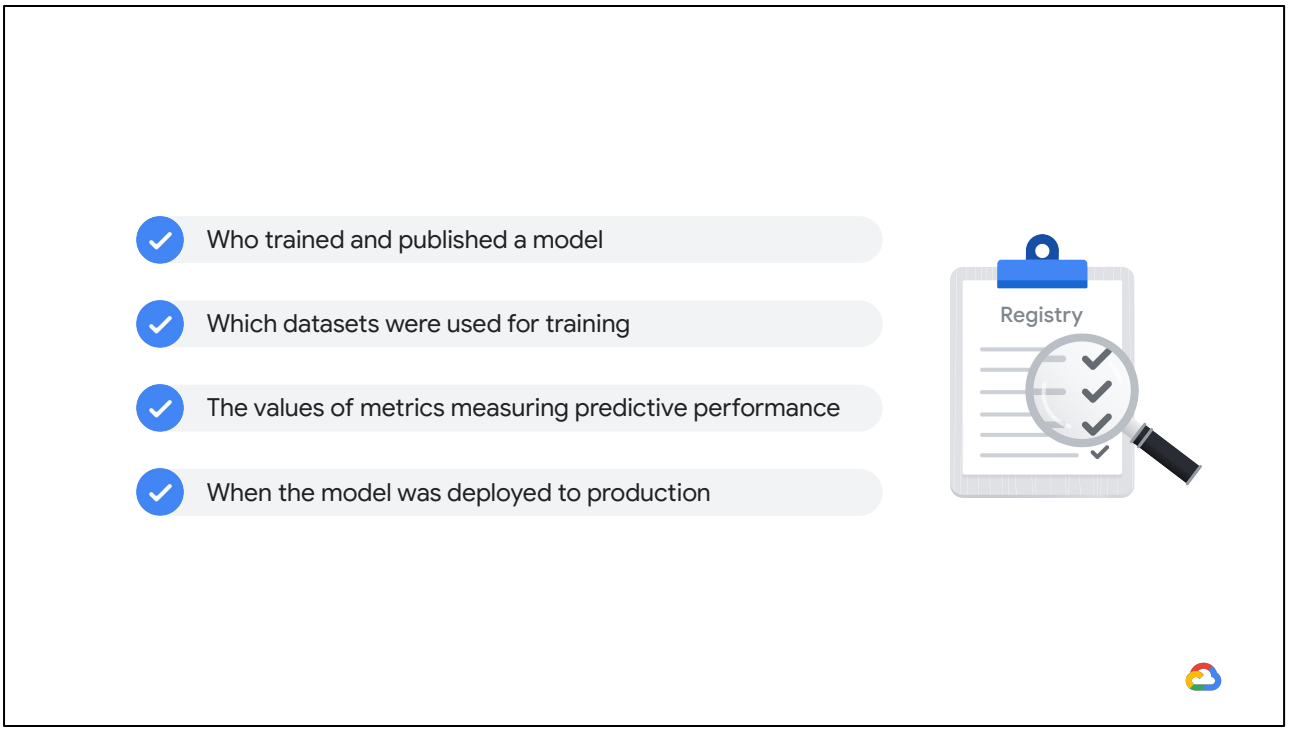


Module 01

Architecting production ML systems



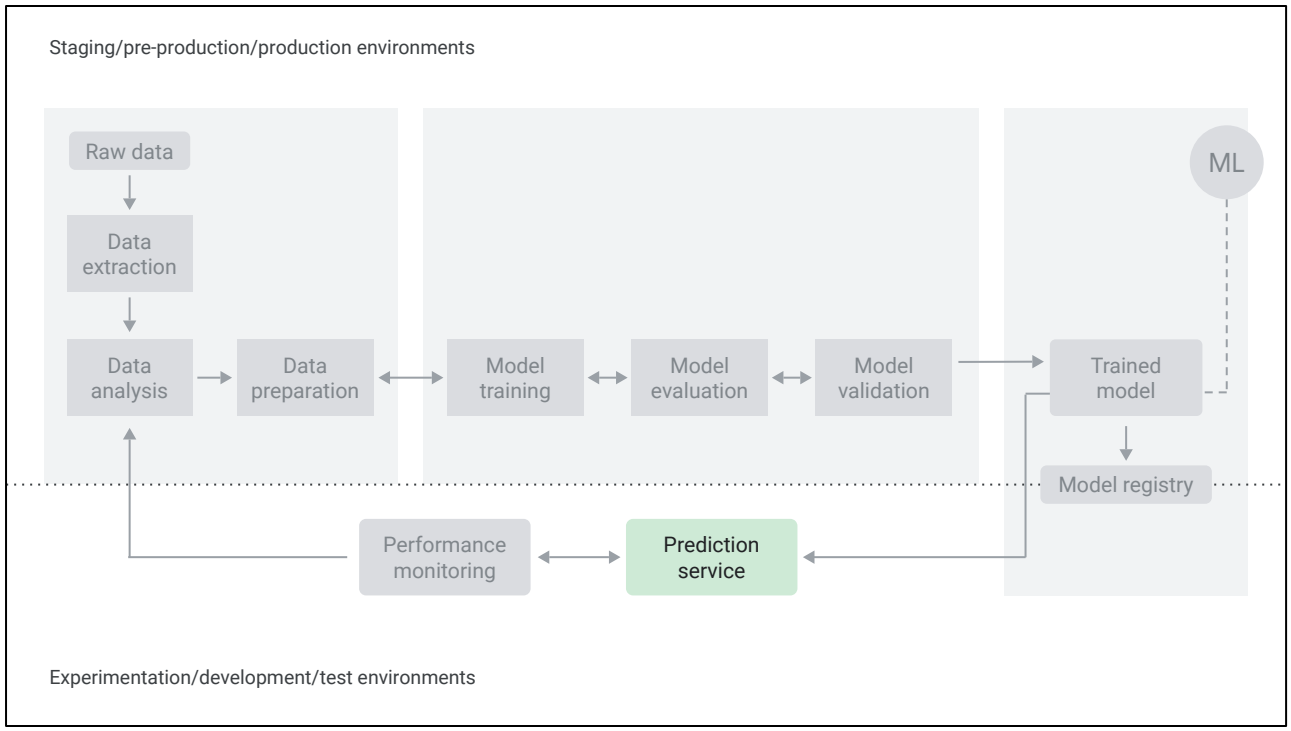
The output of model validation is a **trained model** that can be pushed to the **model registry**. A machine learning model registry is a centralized tracking system that stores lineage, versioning, and related metadata for published machine learning models.



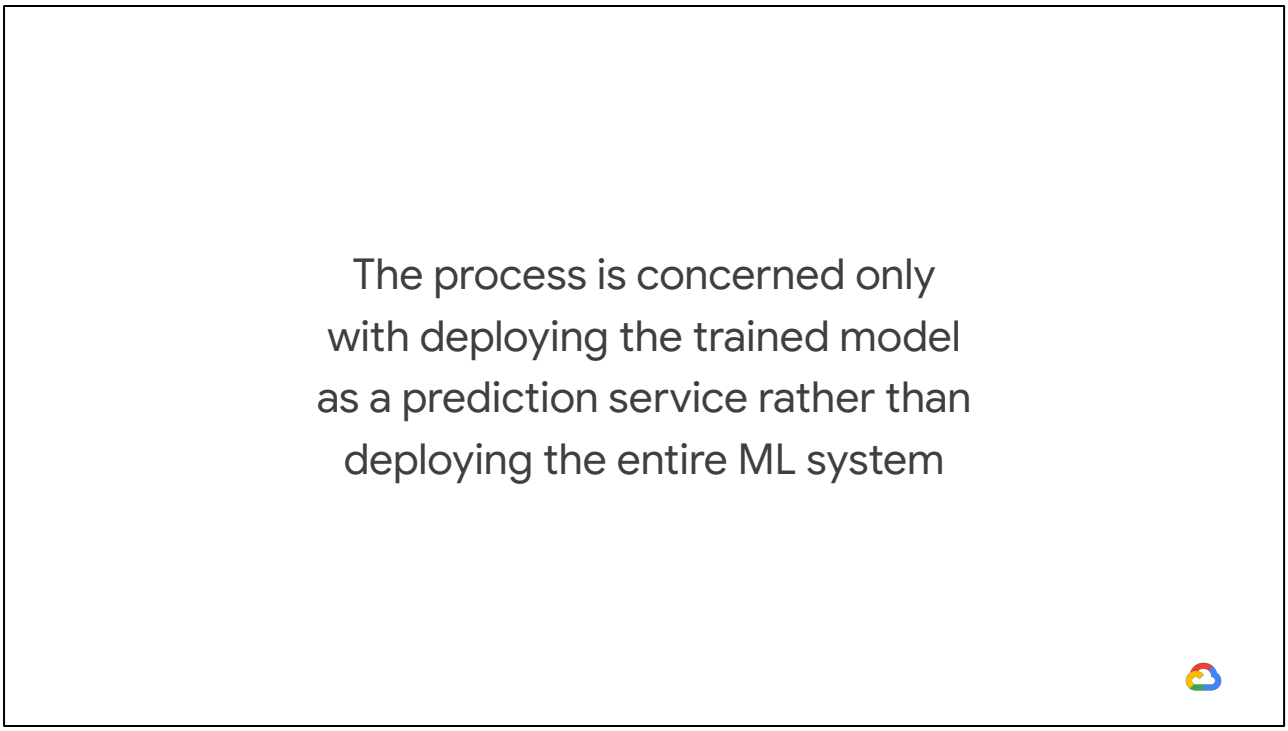
A registry may capture governance data required for auditing purposes, such as:

- Who trained and published a model
- Which datasets were used for training
- The values of metrics measuring predictive performance
- When the model was deployed to production

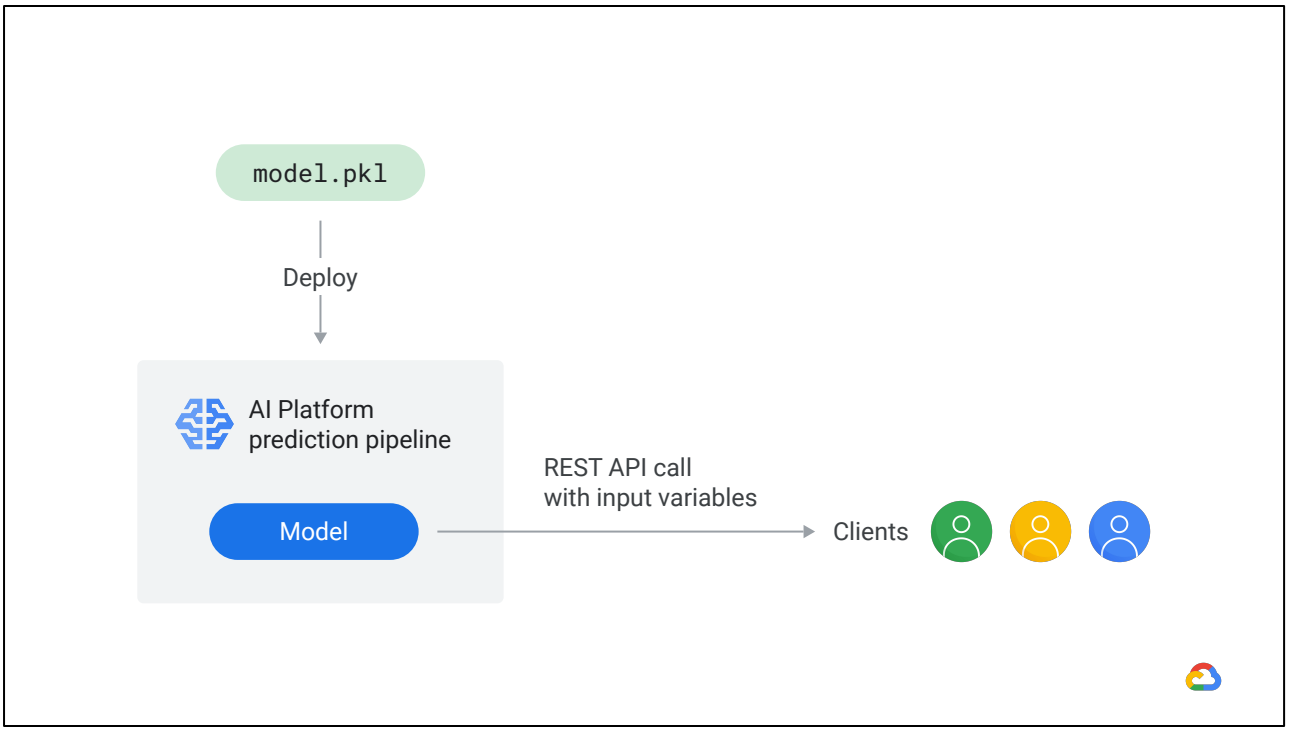
It's a place to find, publish, and use ML models or model pipeline components.



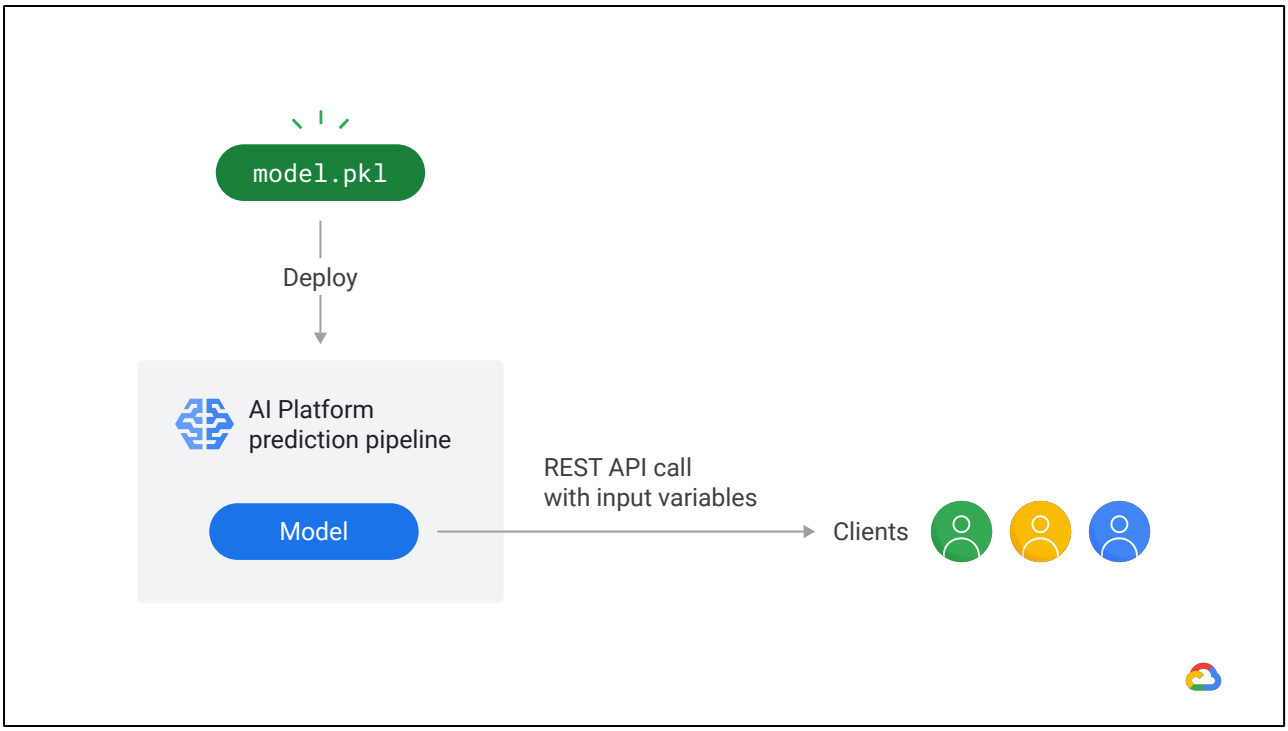
Machine learning uses data to answer questions. So prediction, or inference, is the step where we get to answer the questions we posed–whether it be a business problem or an academic research problem. The trained model is served as a **prediction service** to production.



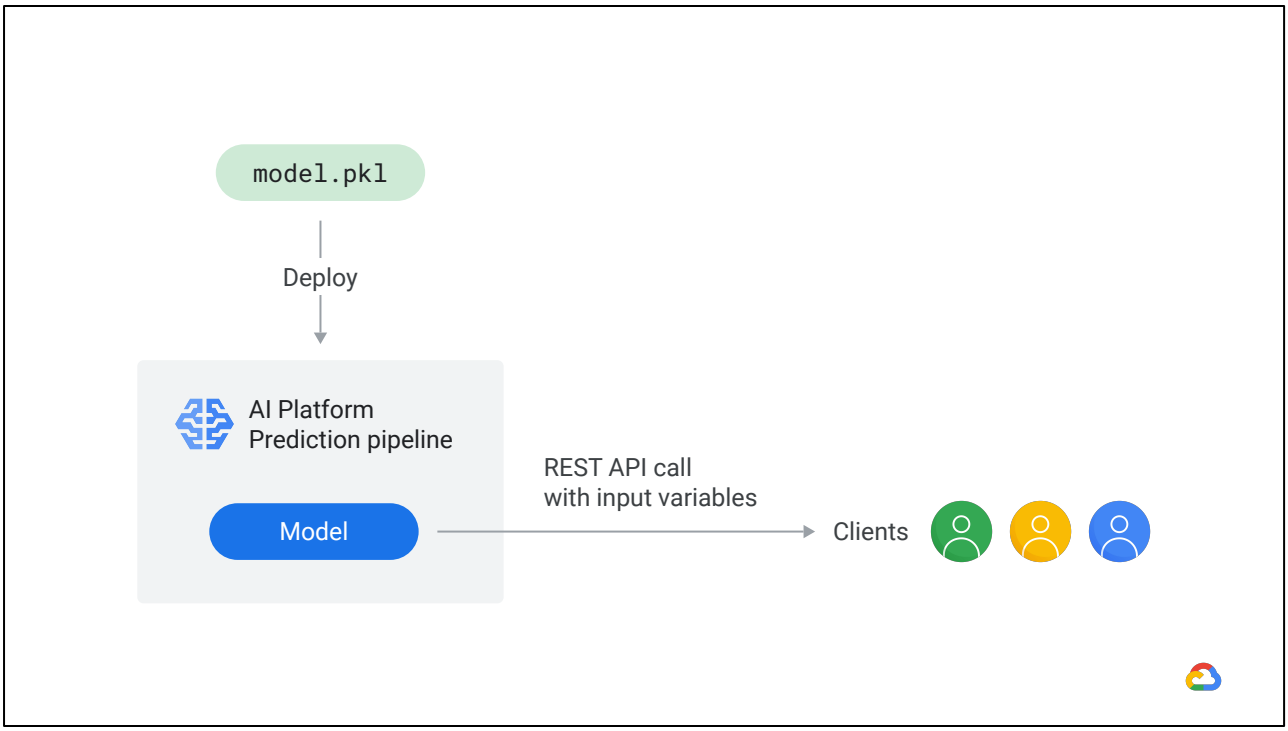
It’s important to note that the process is concerned only with deploying the trained model as a prediction service, for example, a microservice with a REST API, rather than deploying the entire ML system.



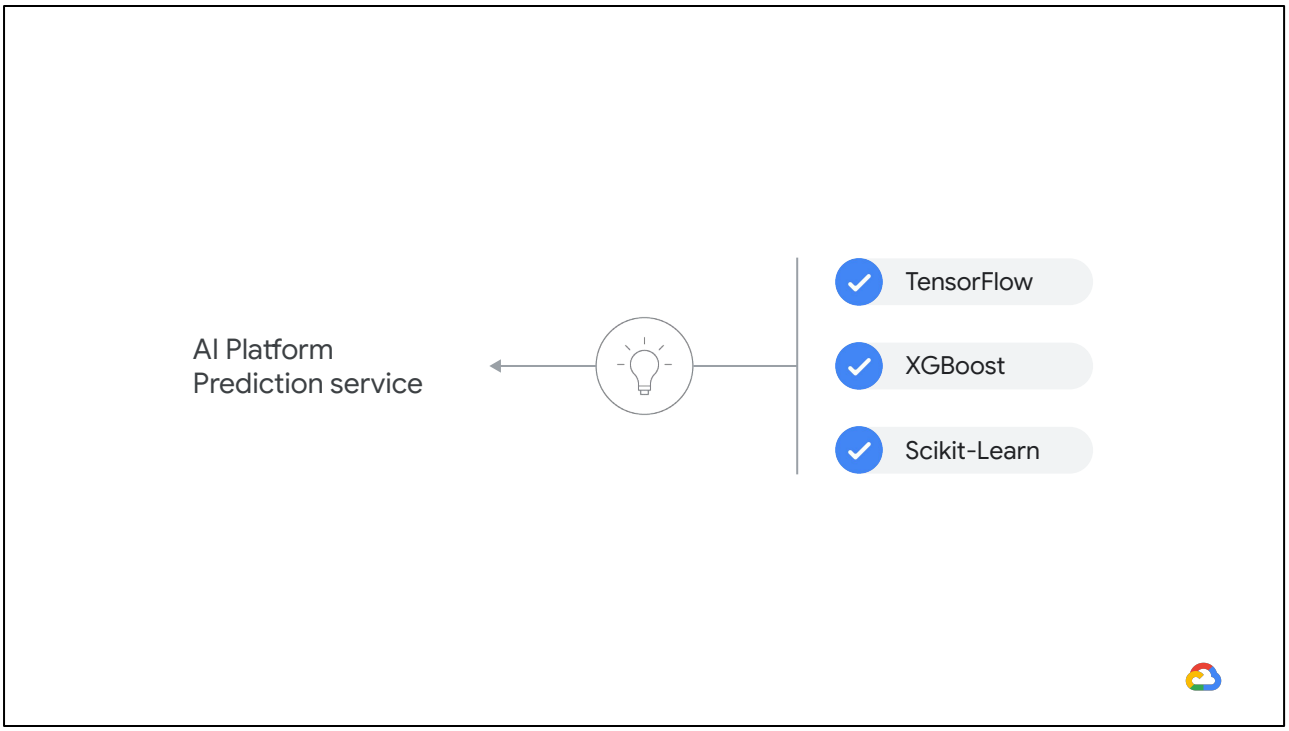
For example, Google's AI Platform Prediction service has an API for serving predictions from machine learning models.



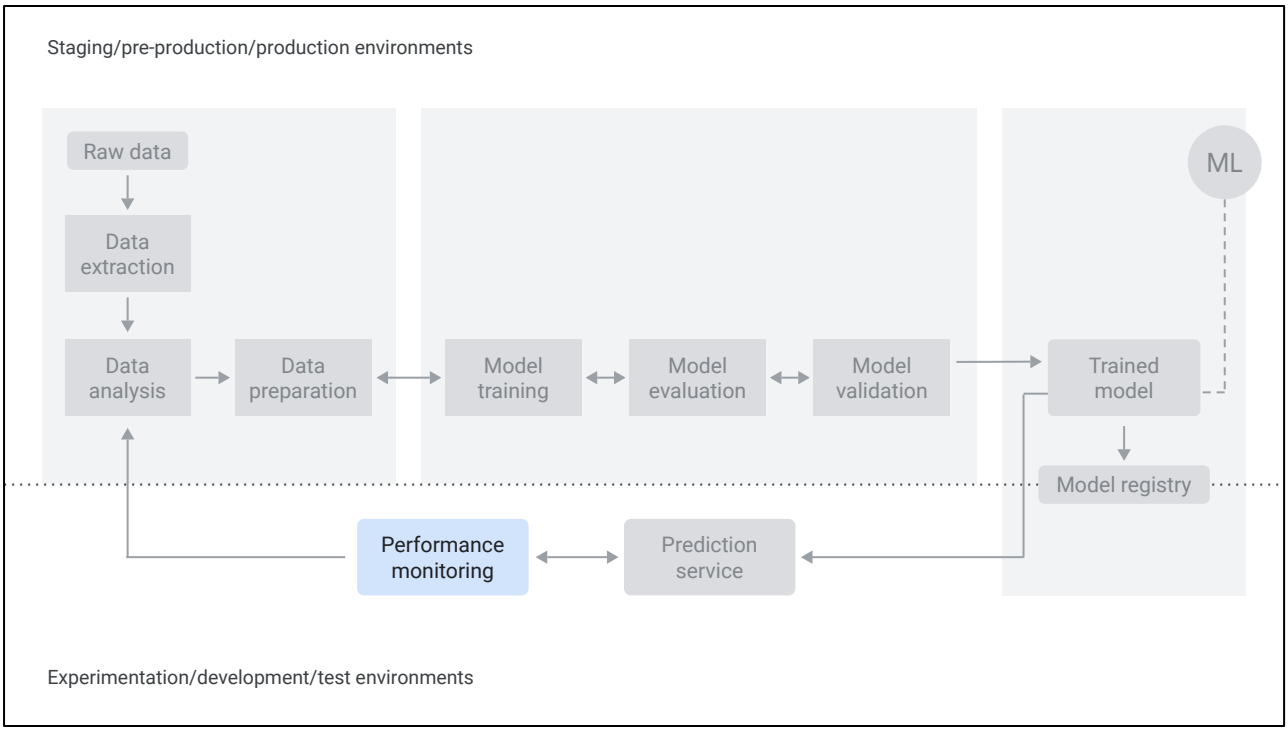
In this particular example, AI Platform Prediction retrieves the trained model and saves it as a pickle in Cloud Storage. Pickle is the standard way of serializing objects in Python.



Trained models deployed in AI Platform Prediction service are exposed as REST endpoints that can be invoked from any standard client that supports HTTP, such as a JupyterLab notebook.

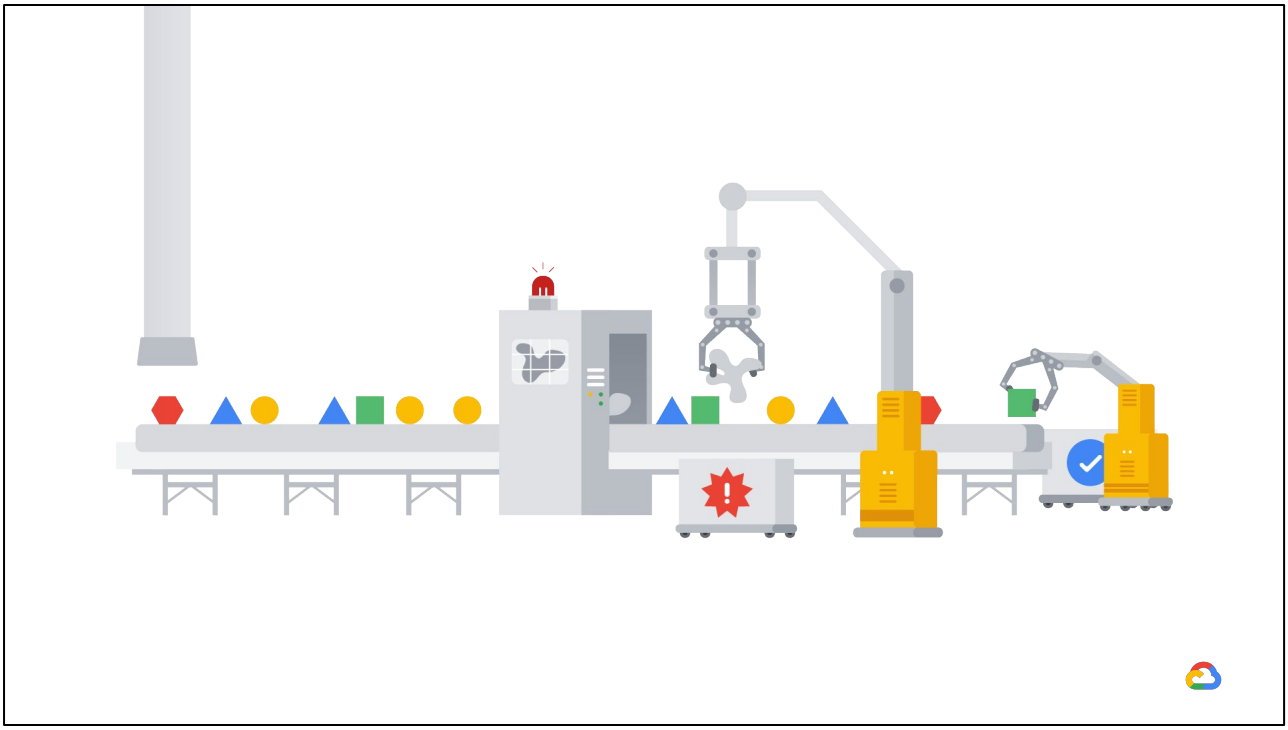


The AI Platform Prediction service can host models trained in popular machine learning frameworks including TensorFlow, XGBoost, and Scikit-Learn.

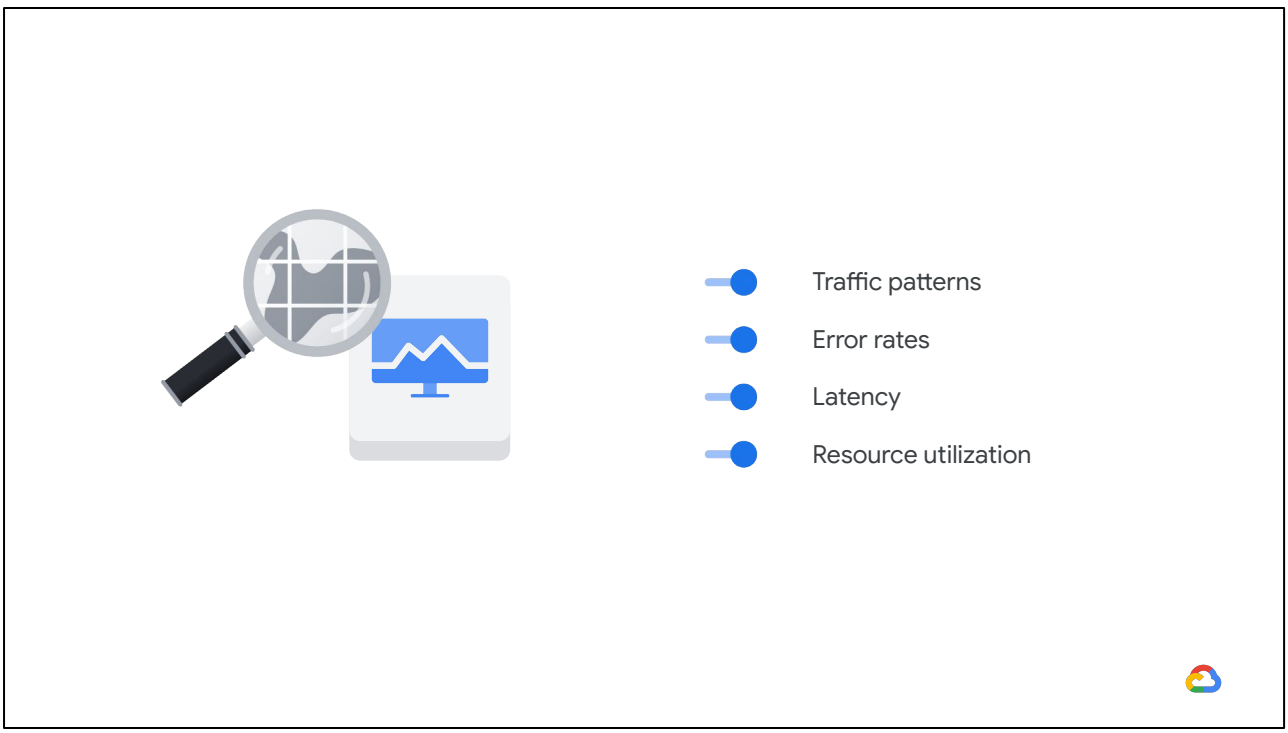


As a best practice, you need a way to actively monitor the quality of your model in production. **Monitoring** lets you detect model performance degradation or model staleness.

The output of monitoring for these changes then feeds into the data analysis component, which could serve as a trigger to execute the pipeline or to execute a new experimental cycle.

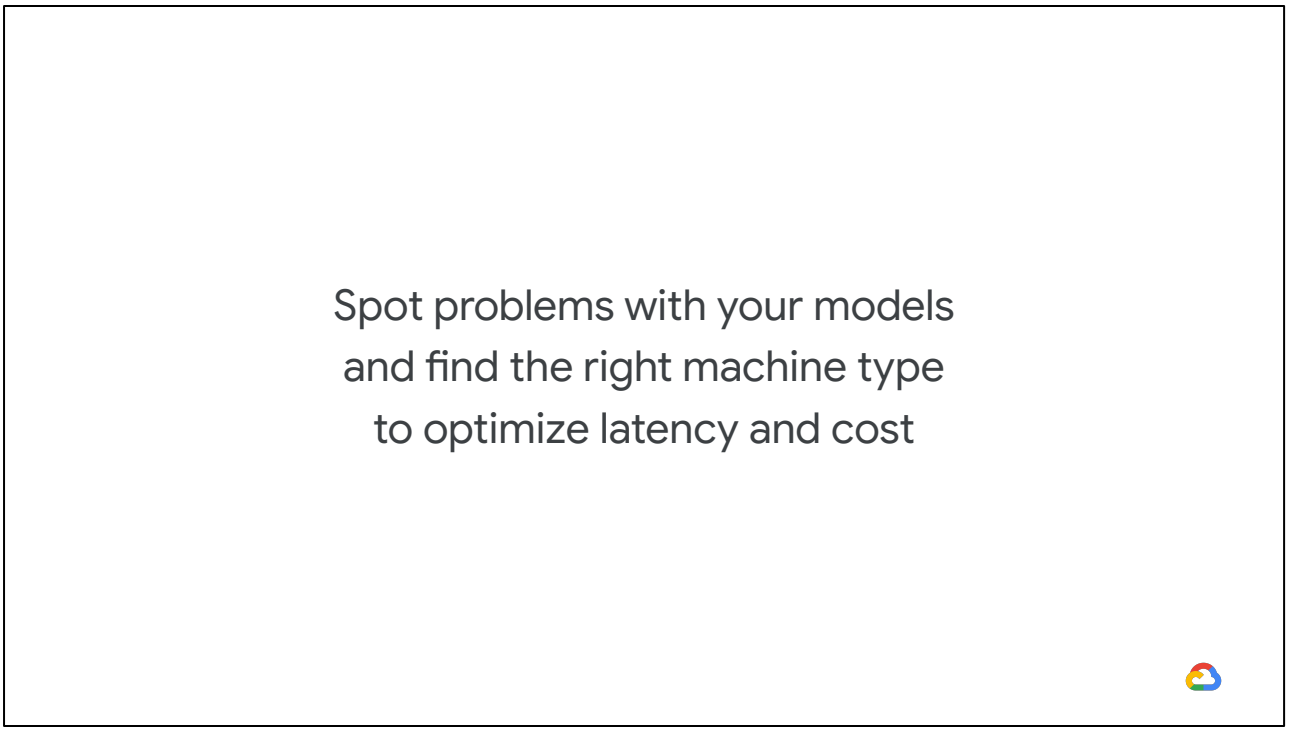


For example, monitoring should be designed to detect data skews, which occur when your model training data is not representative of the live data. That is to say, the data that we used to train the model in the research or production environment does not represent the data that we actually get in our live system, and this leads to model staleness.



To understand other performance metrics, you can configure Google's Cloud Monitoring to monitor your model's:

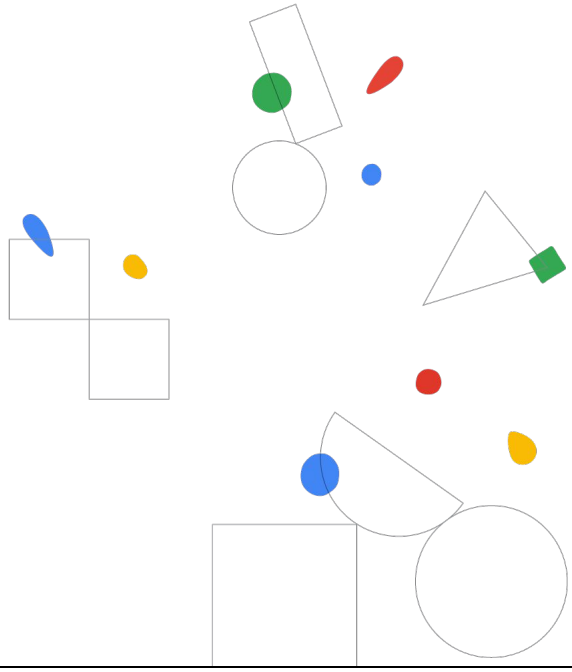
- Traffic patterns
- Error rates
- Latency
- Resource utilization



This can help spot problems with your models and find the right machine type to optimize latency and cost.

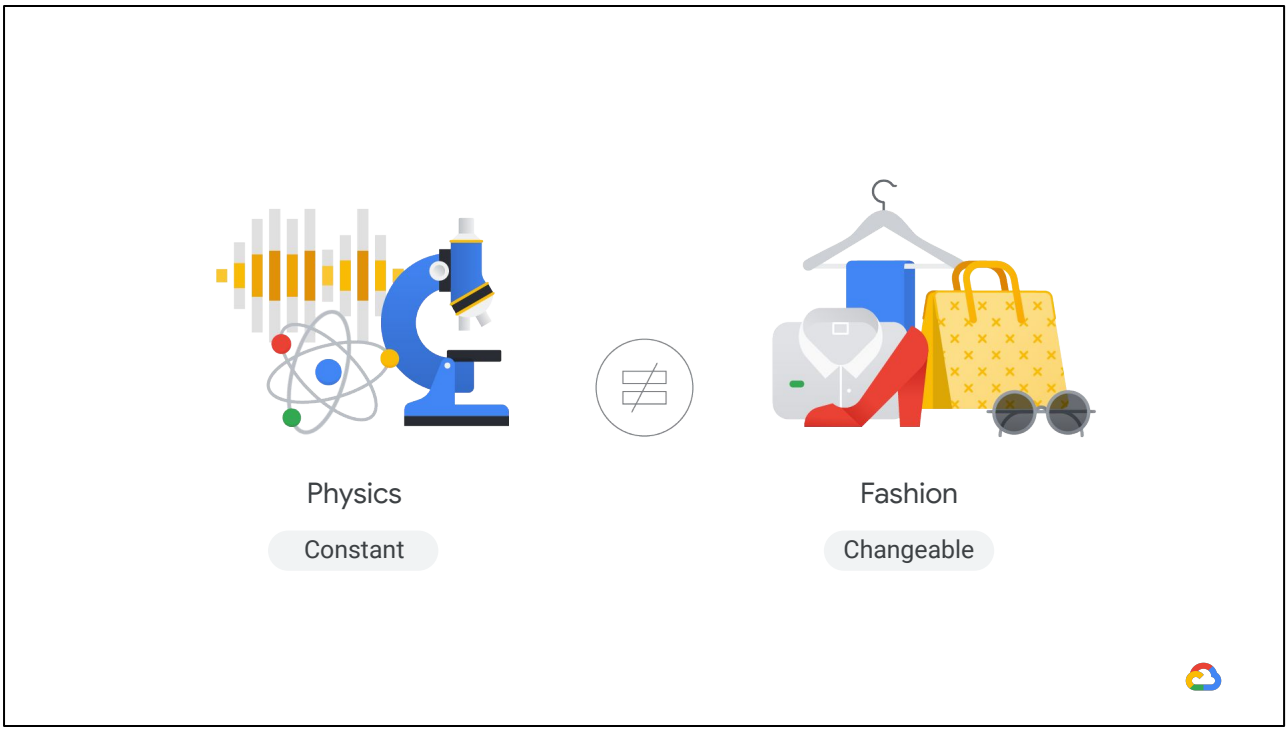
Google Cloud

Training design decisions



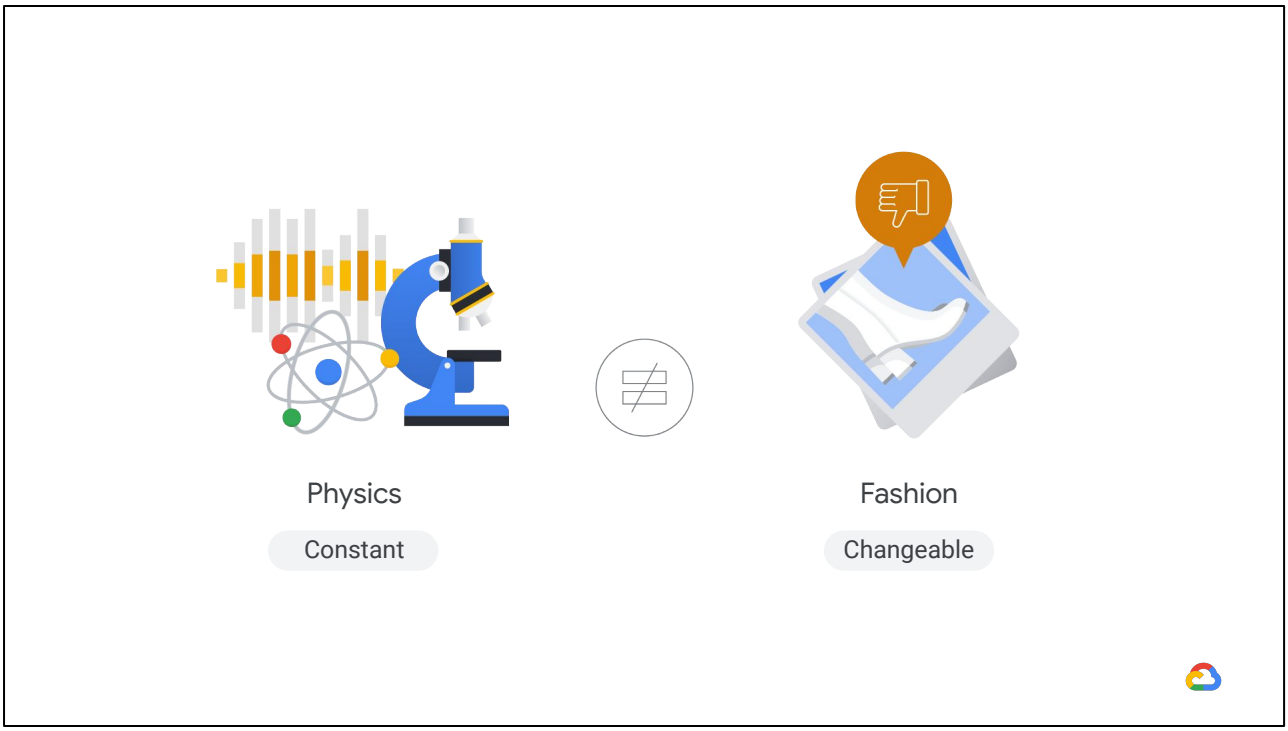
Module 01
Architecting production ML systems

One of the key decisions you'll need to make about your production ML system concerns training.



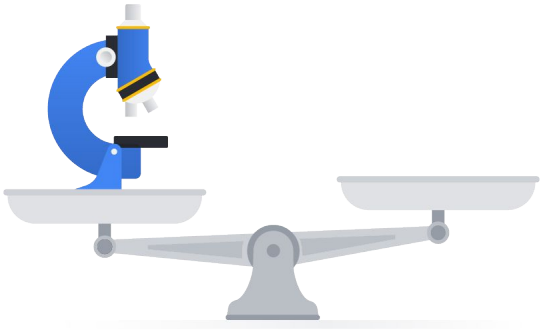
Here's a question.

How is physics unlike fashion? If we assume that science is about discovering relationships that already exist in the world, then the answer is that physics is constant whereas fashion isn't.

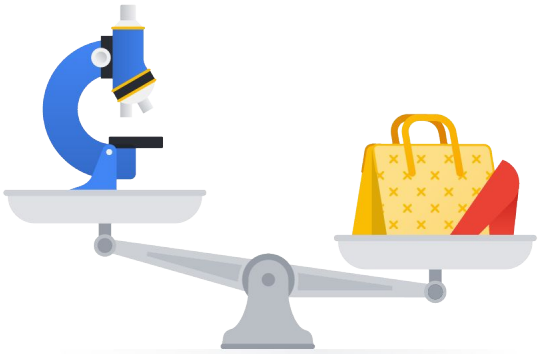


To see some proof, just look at some old pictures of yourself.

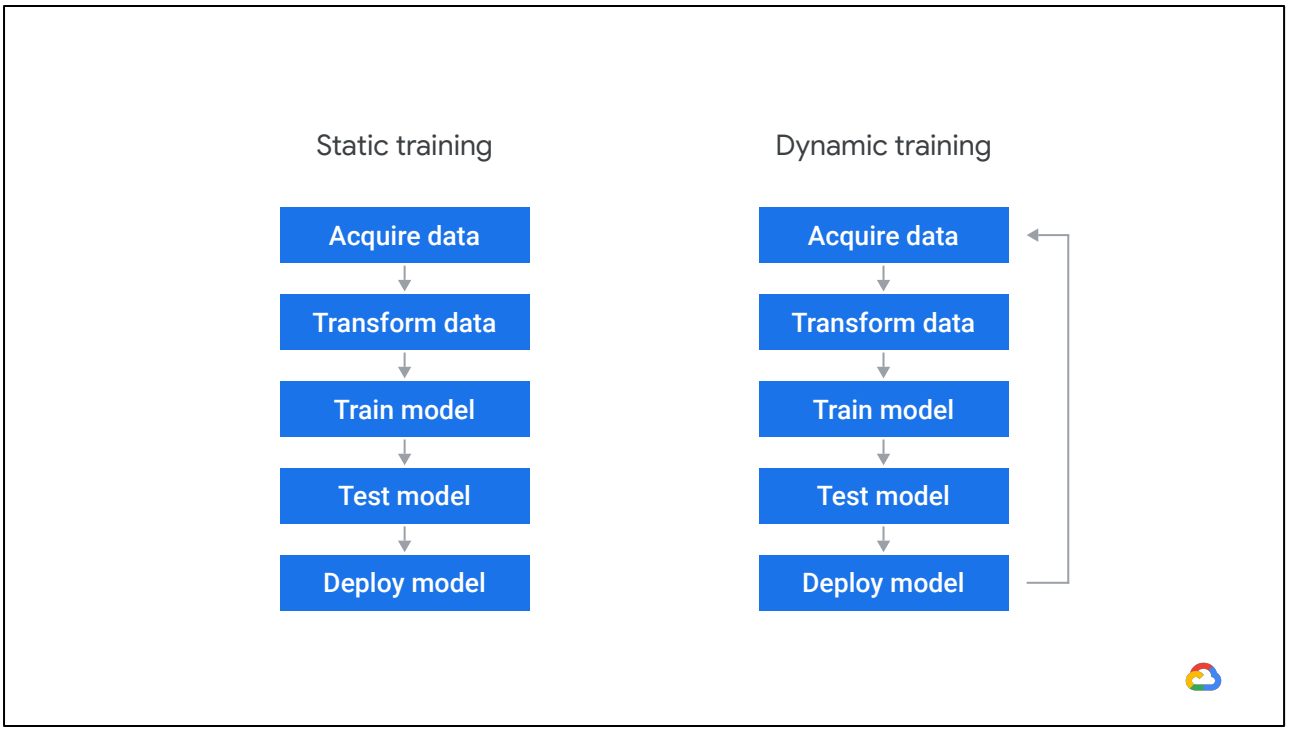
Now, you might be asking, why is this relevant?



Well, when making decisions about training, you have to decide whether the phenomenon you're modelling is more like physics,

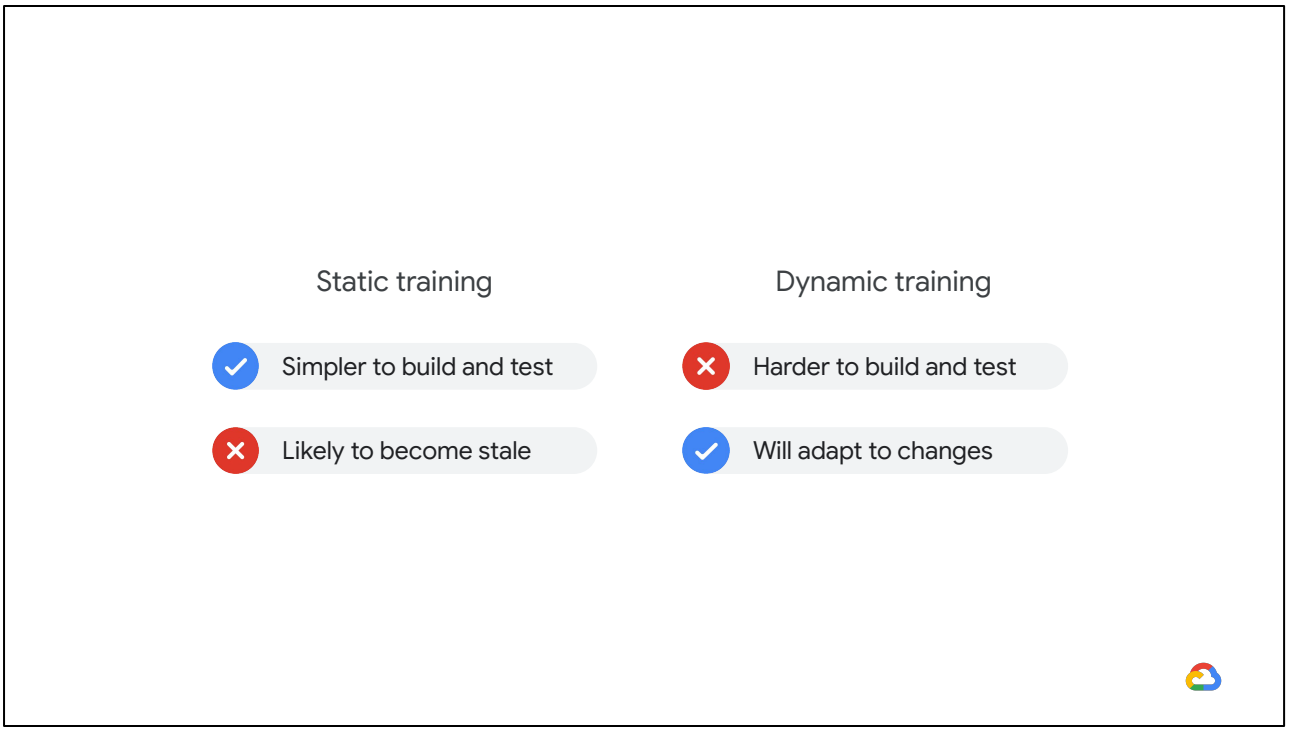


or like fashion.



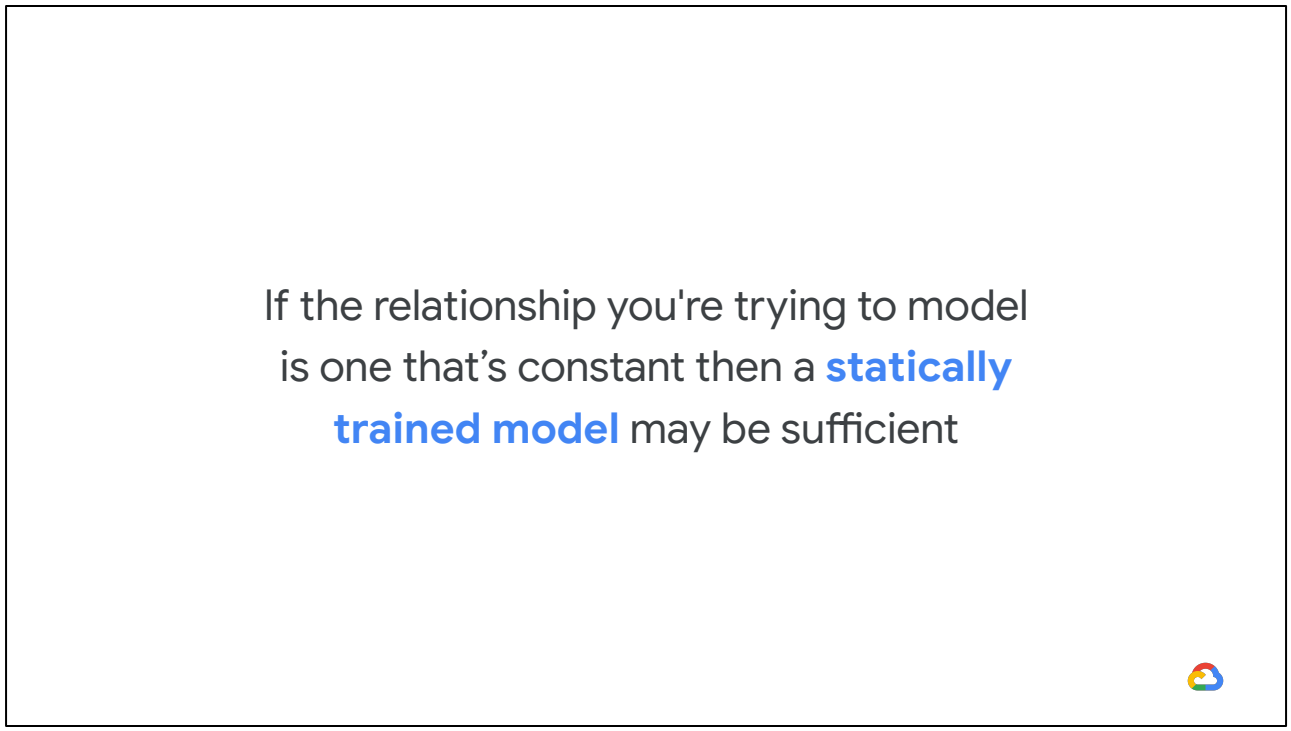
When training your model, there are two paradigms; static training and dynamic training.

- In **static training**, we do what we did in the last specialization. We gather our data, we partition it, we train our model, and then we deploy it.
- In **dynamic training**, we do this repeatedly as more data arrives. This leads to the fundamental trade-off between static and dynamic.

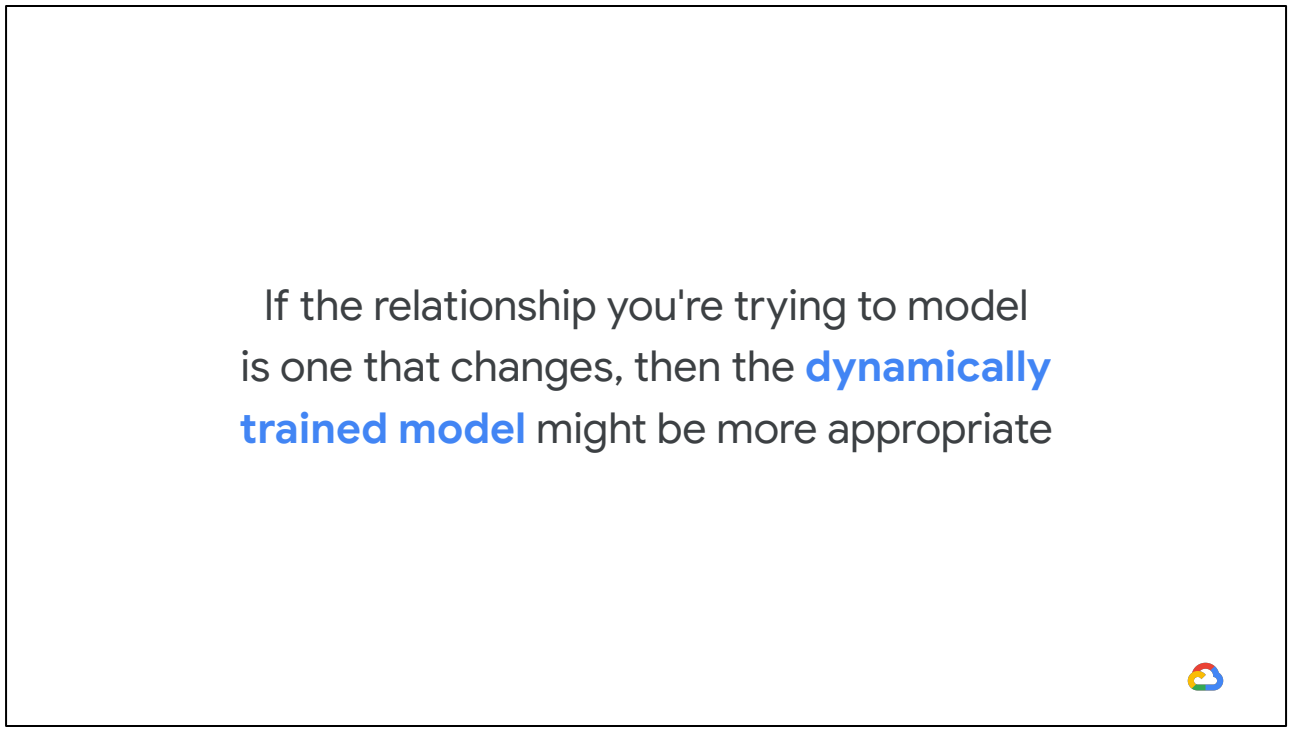


Static is simpler to build and test, but likely to become stale.

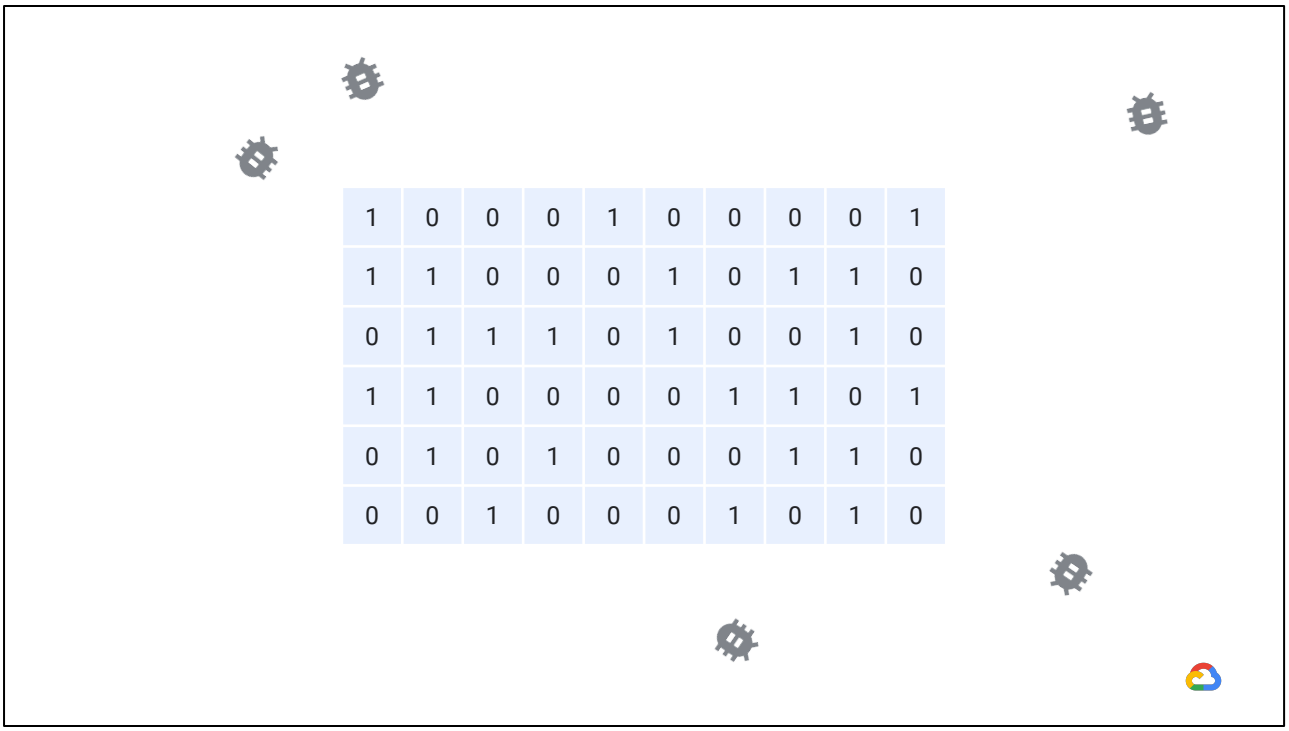
Whereas dynamic is harder to build and test, but will adapt to changes. And the tendency to become or not become stale is what was being alluded to earlier when we contrasted physics and fashion.



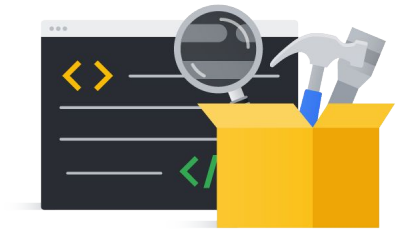
If the relationship you're trying to model is one that's constant, like physics, then a statically trained model may be sufficient.



If the relationship you're trying to model is one that changes, then the dynamically trained model might be more appropriate.

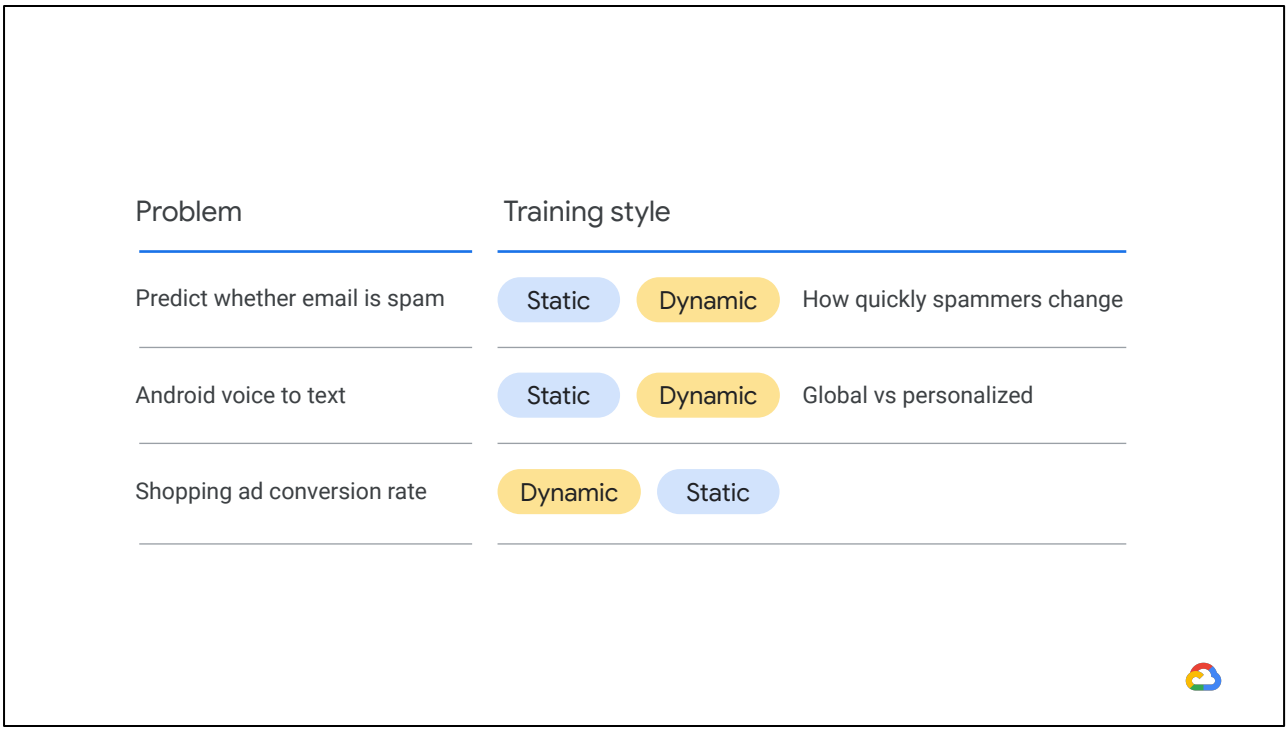


Part of the reason the dynamic is harder to build and test is that new data may have all sorts of bugs in it. And that's something we'll talk about more deeply in a later module on designing adaptable ML systems.



- Monitoring
- Model rollback
- Data quarantine capabilities

Engineering might also be harder because we need more monitoring, model rollback, and data quarantine capabilities.

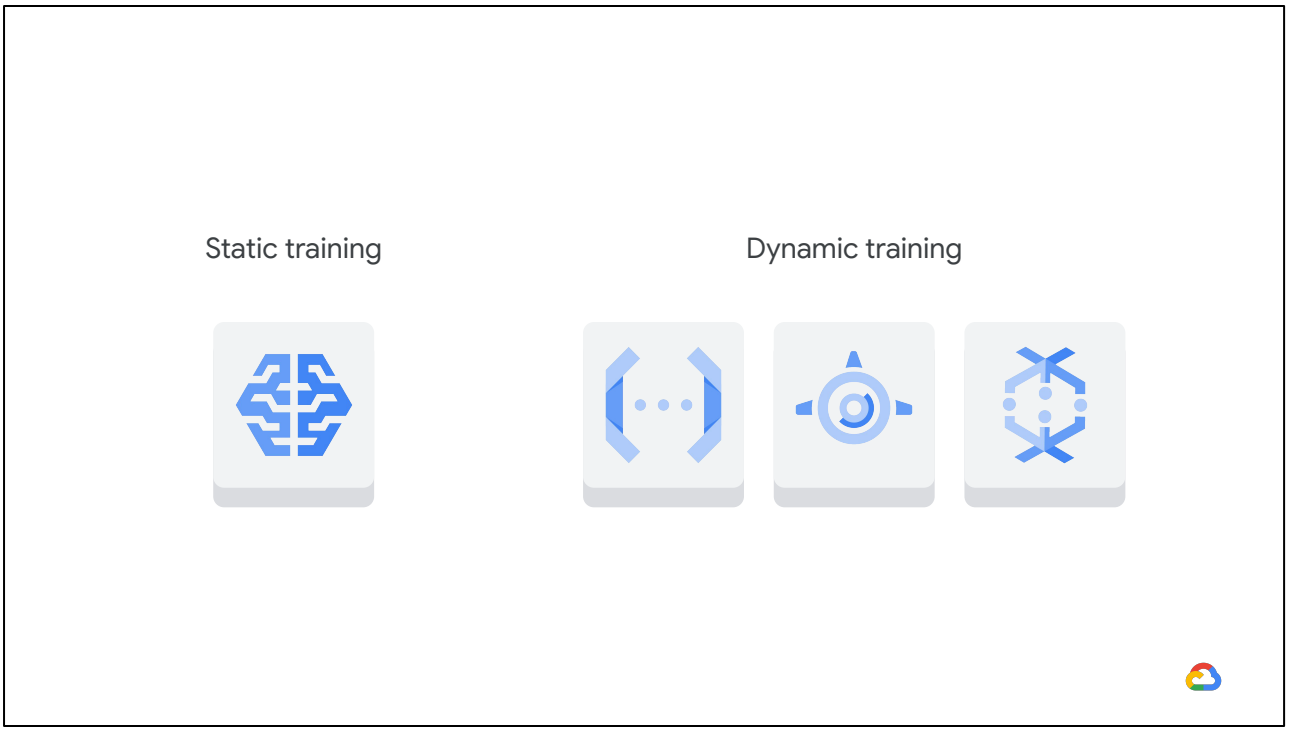


Let’s explore some use cases and think about which sort of training style would be most appropriate.

- The first use case concerns spam detection, and the question you should ask yourself is, “how fresh does spam detection need to be?” You could do this as static, but spammers are a crafty and determined bunch. They will probably discover ways of passing whatever filter you impose within a short time.
  So, dynamic is likely to be more effective over time.

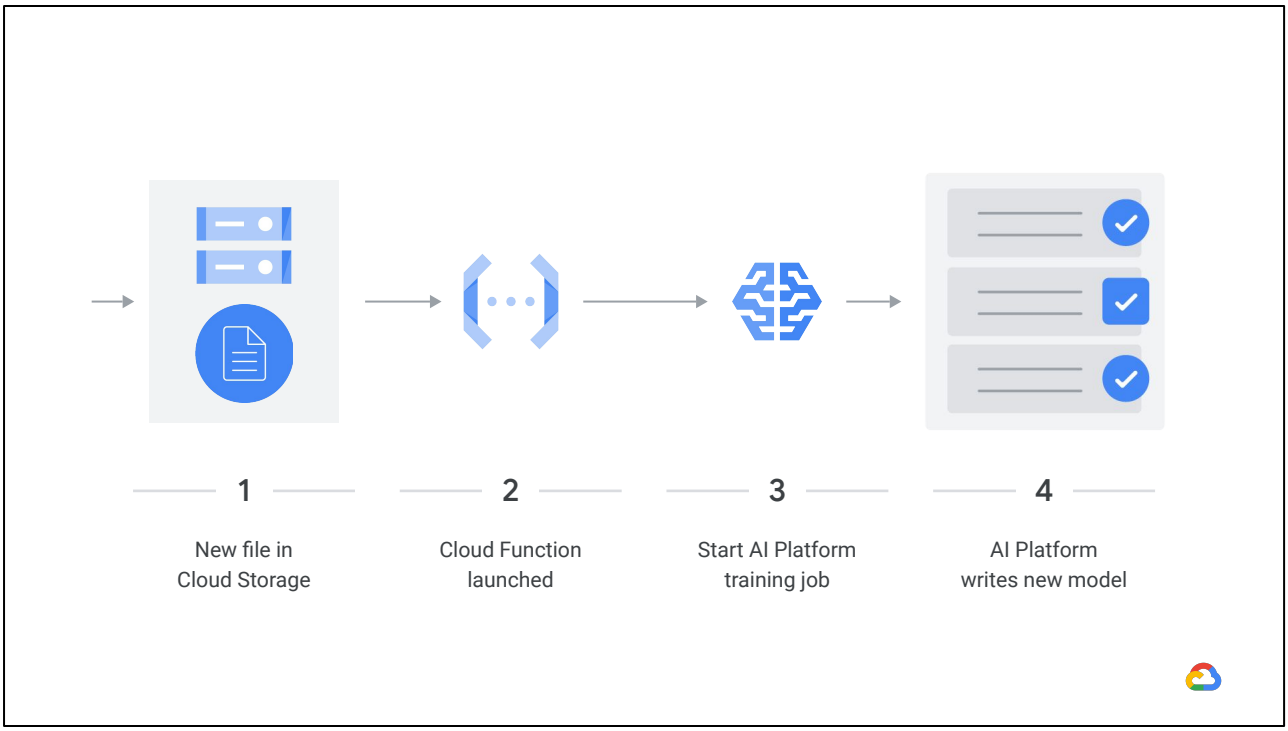
- What about Android Voice-to-Text? Note that this question has some subtlety. For a global model, training offline is probably fine. But, if you want to personalize the voice recognition, you may need to do something online, or at least different, on the phone.
  So this could be static or dynamic, depending on whether you want global or personalized transcription.

- What about ad conversion rate? The interesting subtlety here is that conversions may come in very late. For example, if I'm shopping for a car online, I'm unlikely to buy for a very long time. This system could use dynamic training, then regularly going back at different intervals to catch up on new conversion data that has arrived for the past.
  So in practice, most of the time, you'll need to use dynamic, but you might start with static because it's simpler.



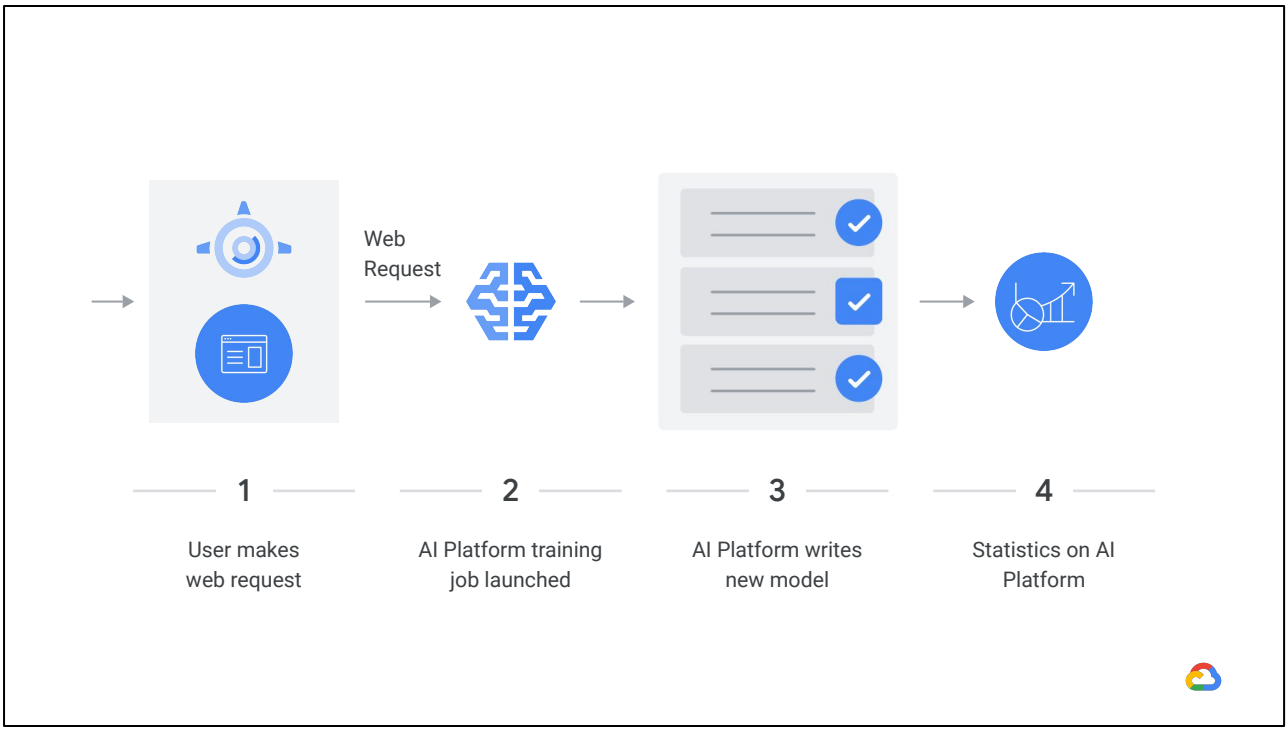
In a reference architecture for static training, models are trained once and then pushed to AI Platform.

Now, for dynamic training, there are three potential architectures to explore, Cloud Functions, App Engine, or Cloud Dataflow.



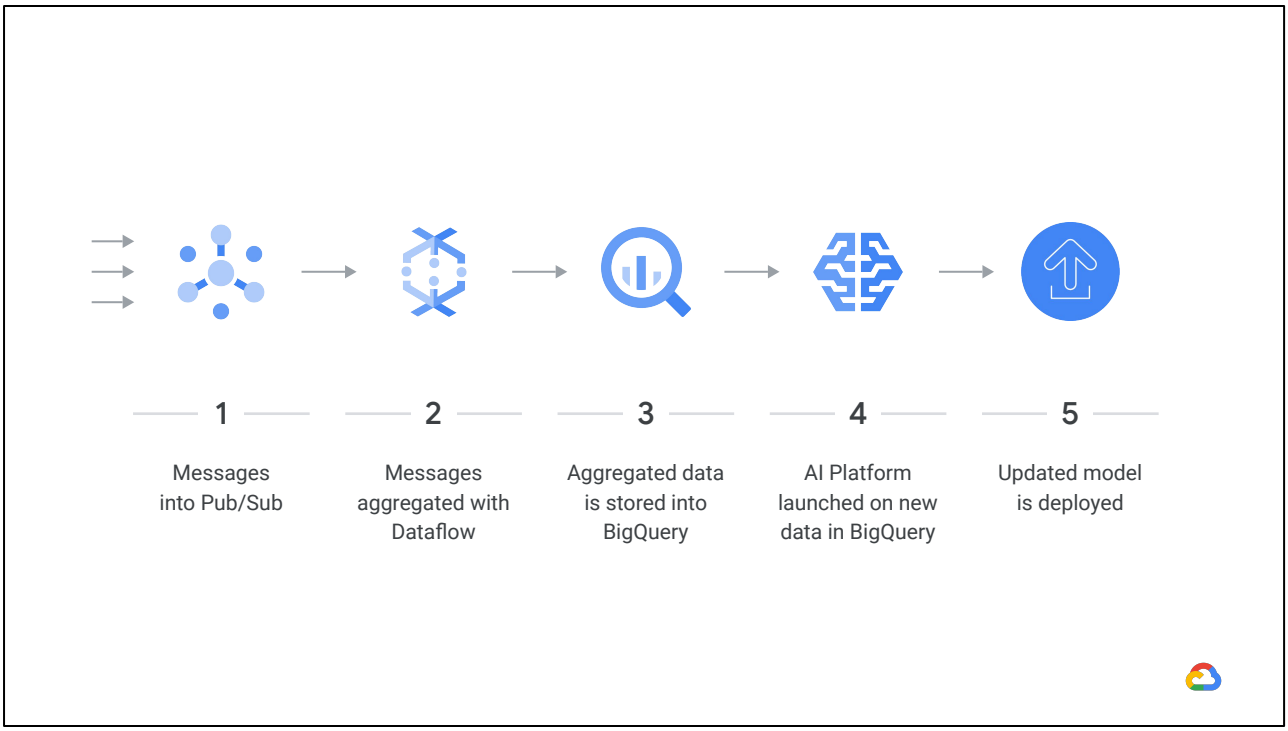
In a general architecture for dynamic training using Cloud functions, a new data file appears in Cloud storage and then the Cloud function is launched.

After that, the Cloud function starts the AI Platform training job, and then the AI Platform writes out a new model.



In a general architecture for dynamic training using App Engine, when a user makes a web request, perhaps from a dashboard to AppEngine, an AI Platform training job is launched, and the AI Platform job writes a new model to Cloud storage.

From there, the statistics of the training job are displayed to the user when the job is complete.



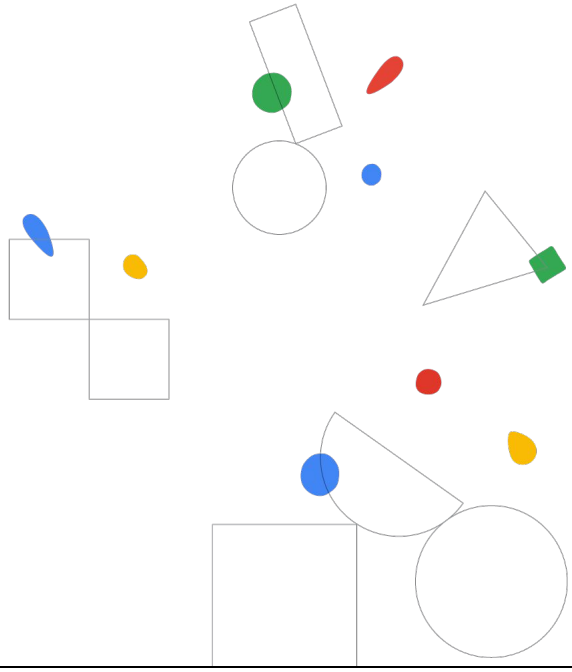
It's possible that the Dataflow pipeline is also invoking the model for predictions.

Here, a streaming topic is ingested into Pub/Sub from subscribers. Messages are then aggregated with Dataflow and aggregated data is stored into BigQuery.

AI Platform is launched on the arrival of new data in BigQuery and then an updated model is deployed.

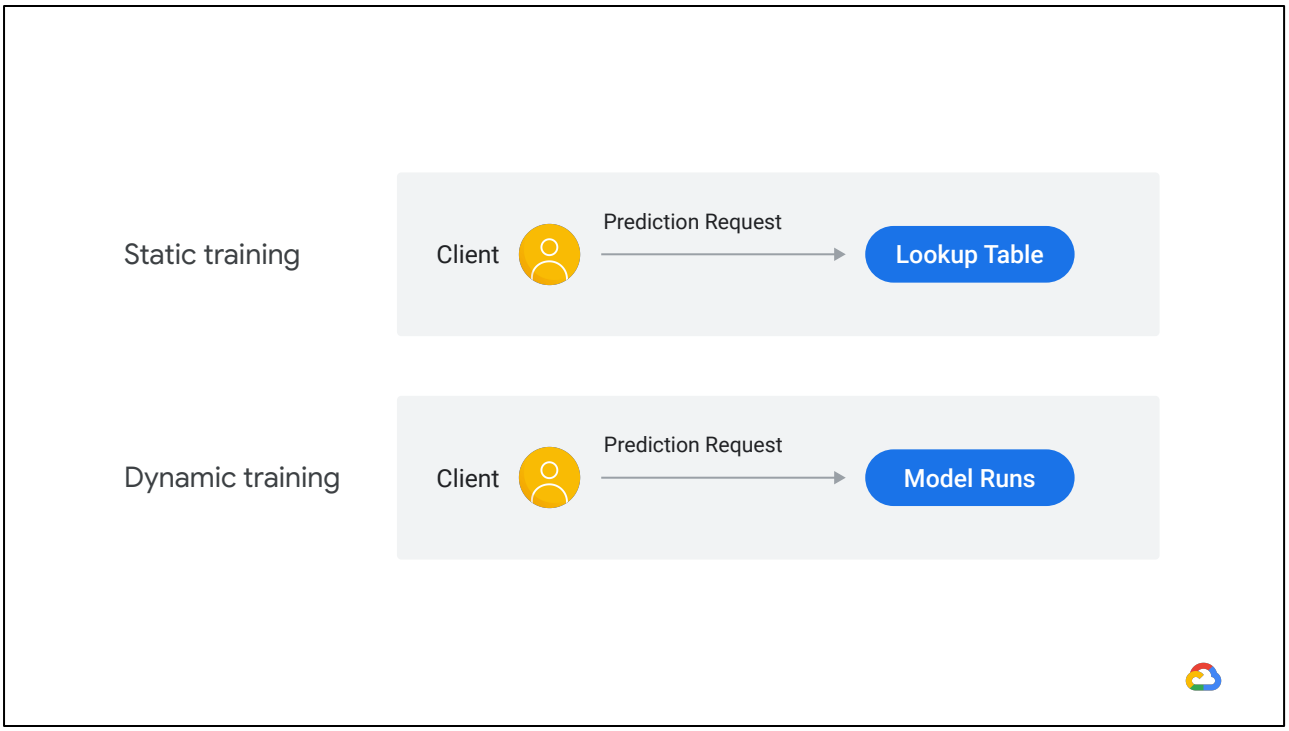
Google Cloud

Serving design decisions



Module 01
Architecting production ML systems

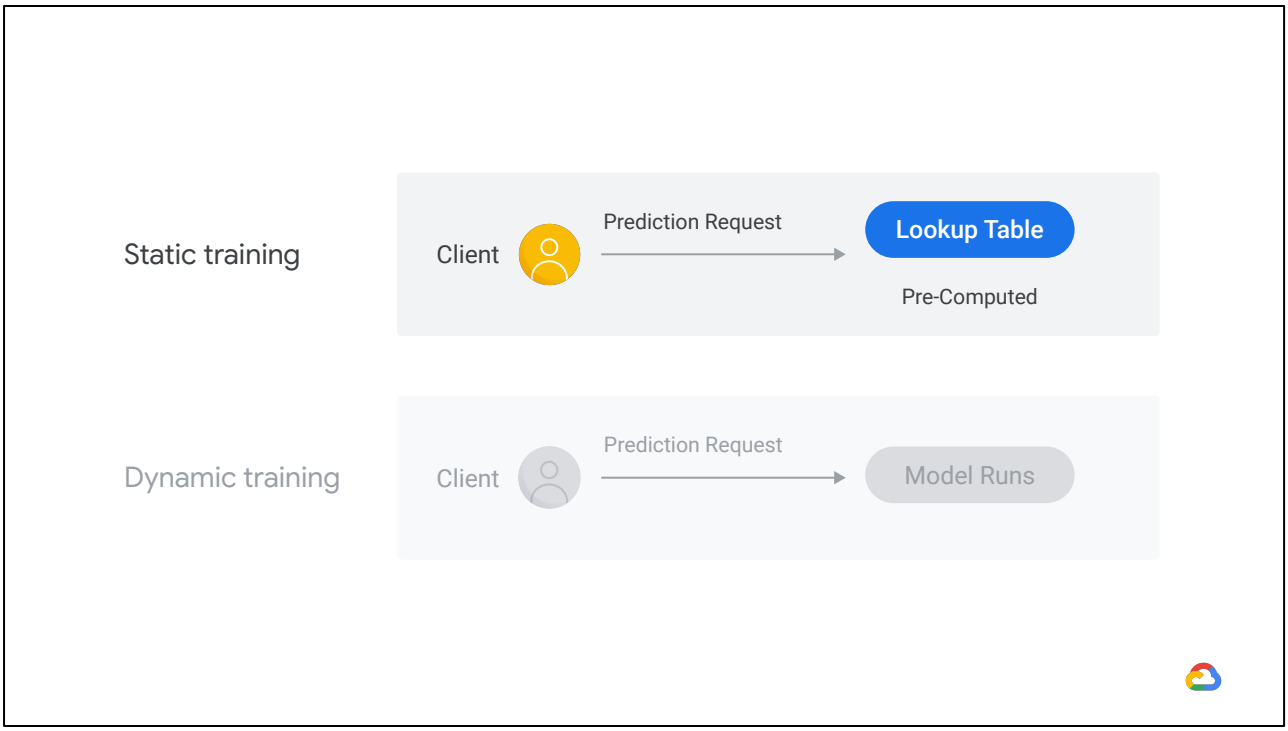
Just as they use case determines appropriate training architecture, it's also determines the appropriate serving architecture.



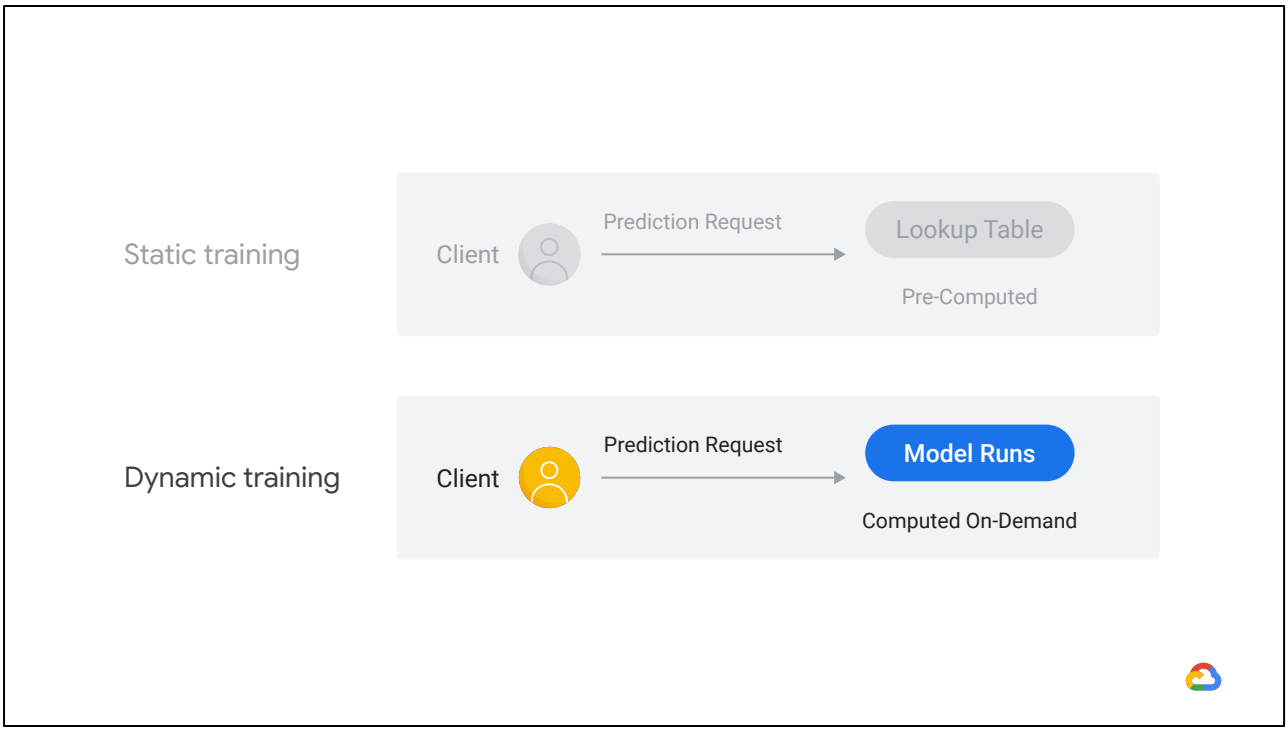
In designing our serving architecture, one of our goals is to minimize average latency.

Just like in operating systems, where we don't want to be bottlenecked by slow disk I/O, when serving models, we don't want to be bottlenecked by slow-to-decide models.

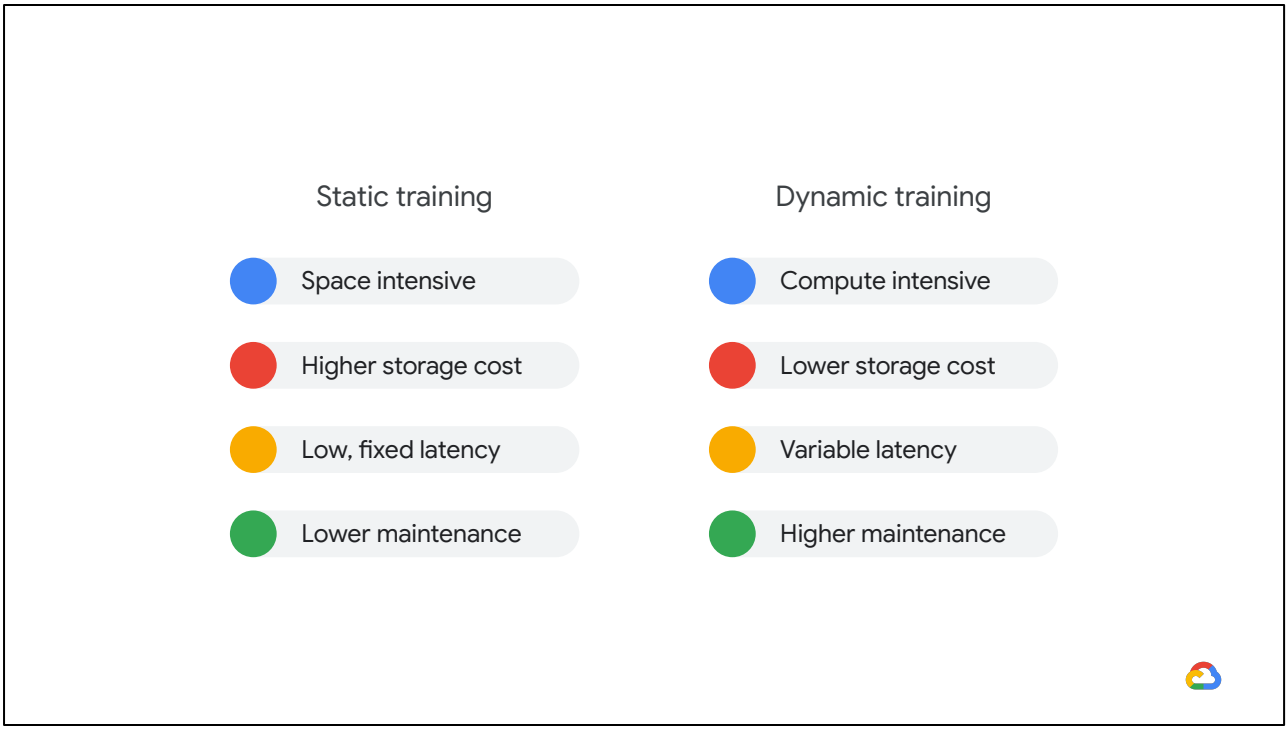
Remarkably, the solution for serving models is very similar to what we do to optimize I/O performance: we use a cache. In this case, rather than faster memory, we'll use a table.



Static serving then computes the label ahead of time and serves by looking it up in the table.



Dynamic serving, in contrast, computes the label on-demand.

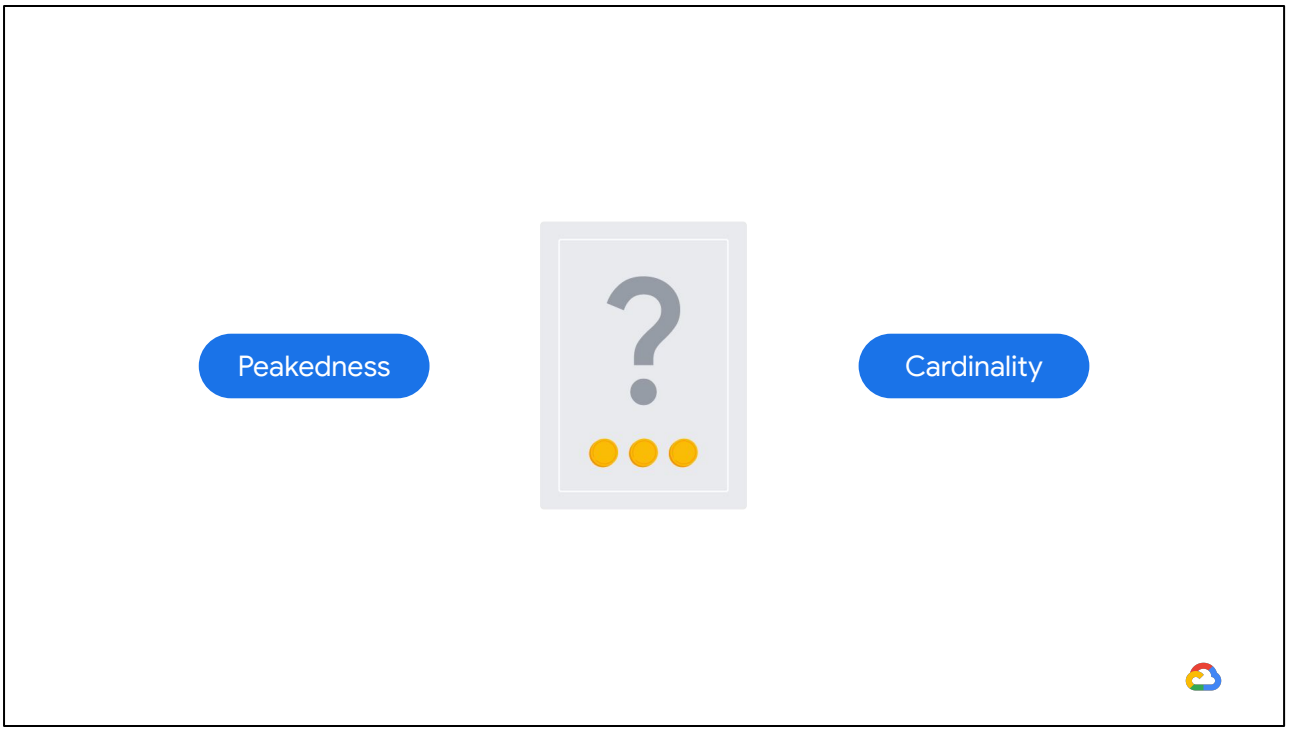


There's a space-time tradeoff.

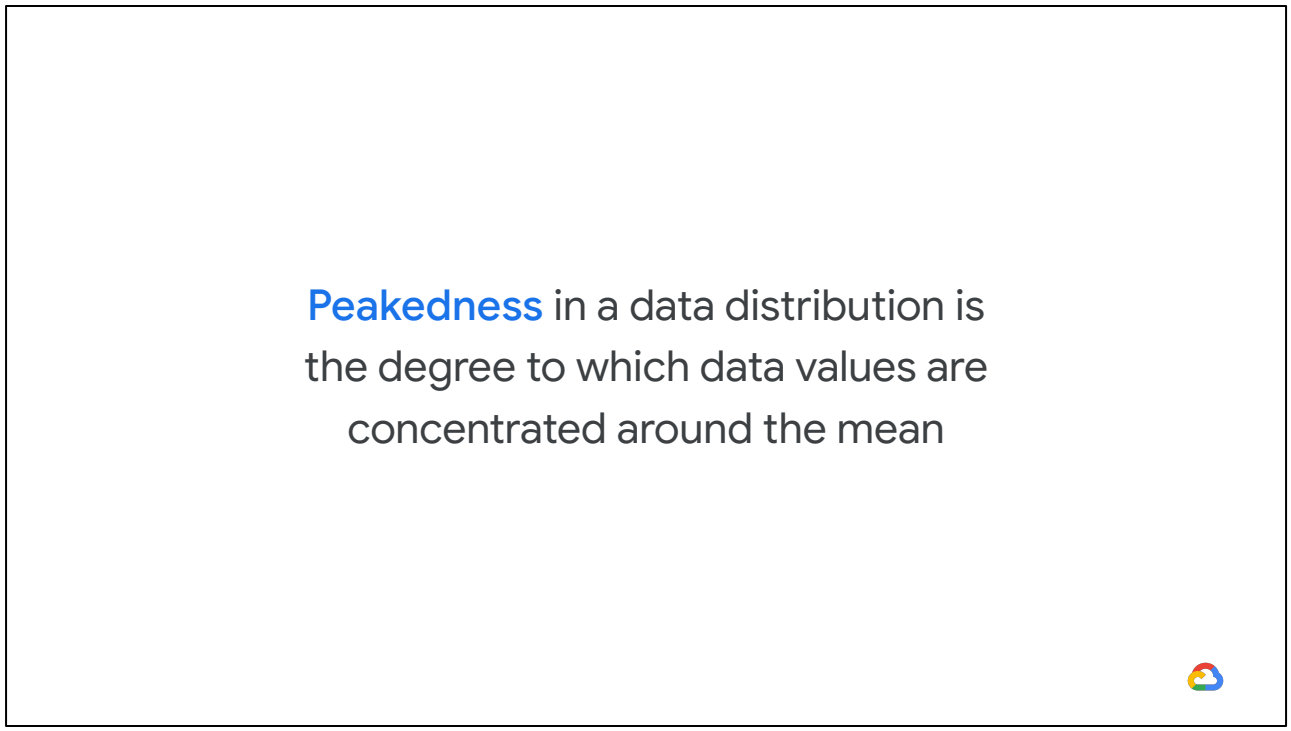
Static serving is space-intensive, resulting in higher storage costs, because we store pre-computed predictions with a low, fixed latency and lower maintenance costs.

Dynamic serving, however, is compute-intensive. It has lower storage costs, higher maintenance, and variable latency.

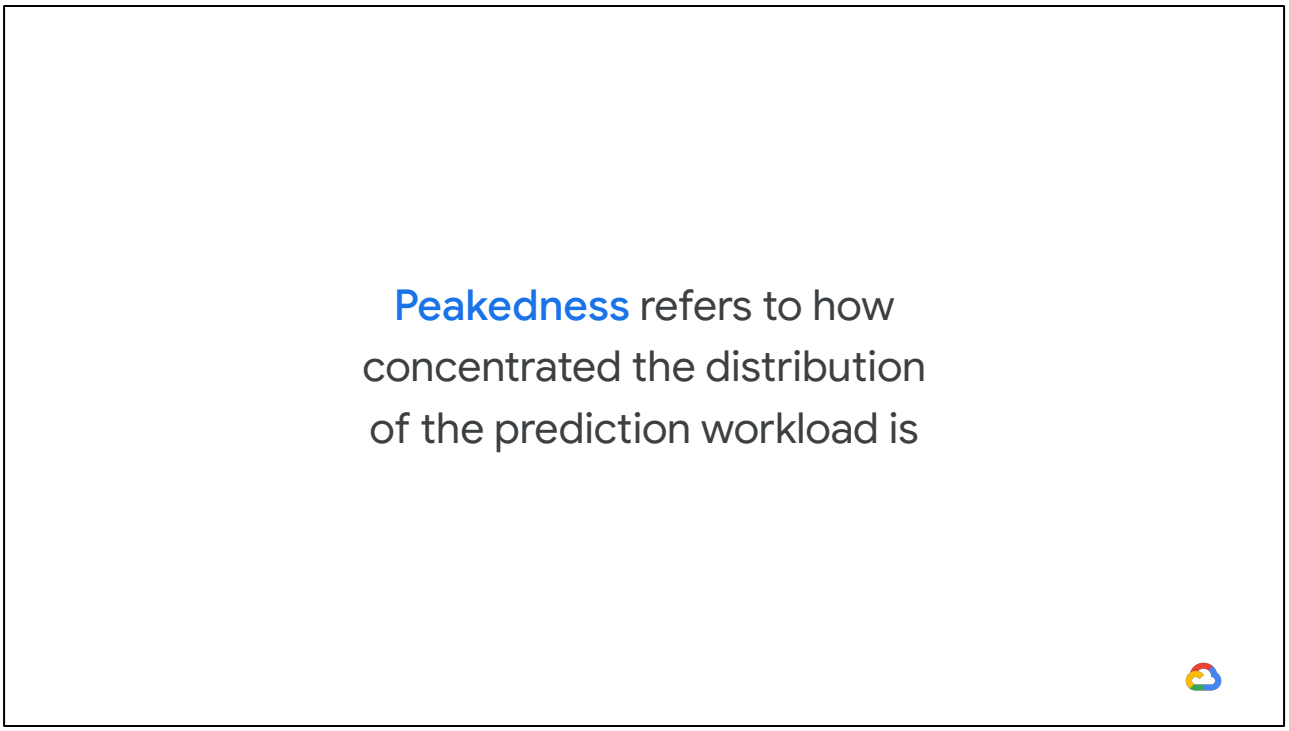
The choice whether to use static or dynamic serving is determined by considering how important latency, storage, and CPU costs are.



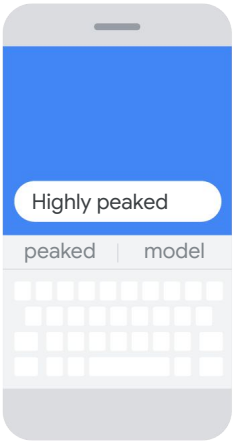
Sometimes, it can be hard to express the relative importance of these three areas. As a result, it might be helpful to consider static and dynamic serving through another lens: peakedness and cardinality.

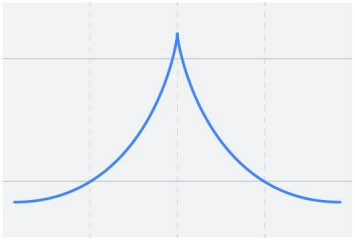


Peakedness in a data distribution is the degree to which data values are concentrated around the mean,



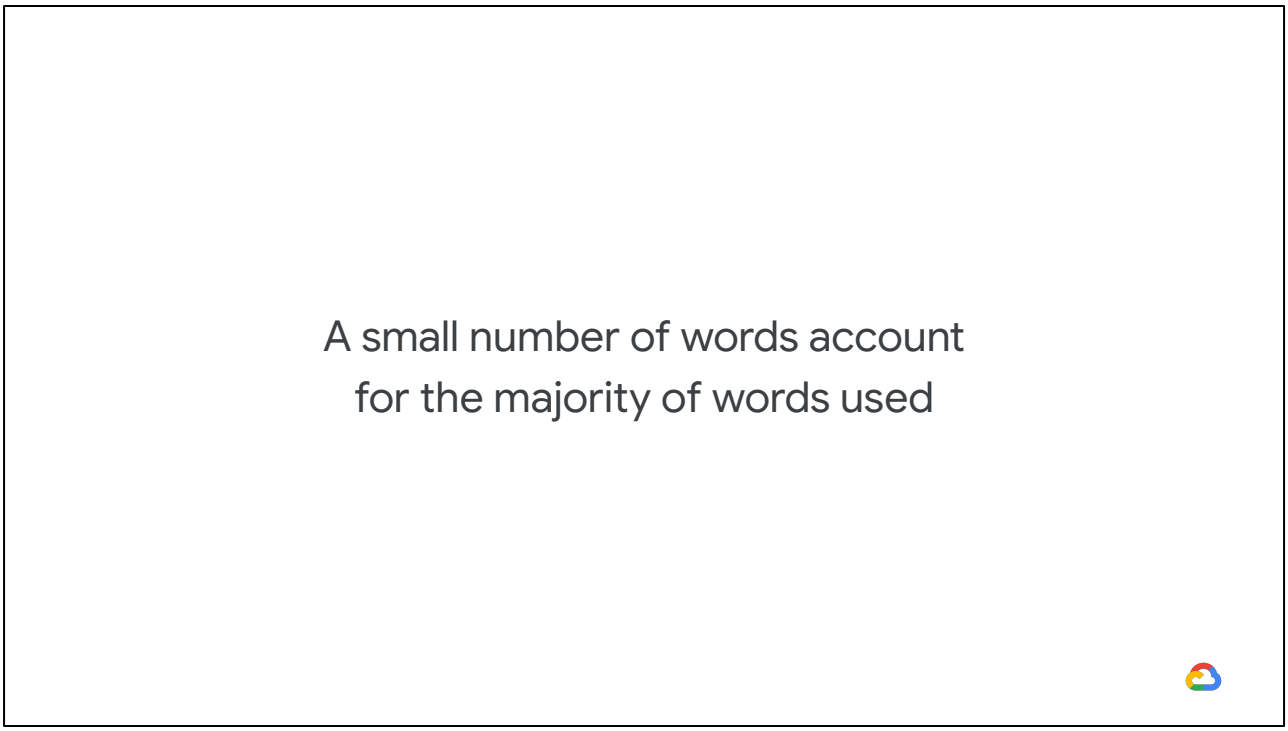
or in this case, how concentrated the distribution of the prediction workload is.



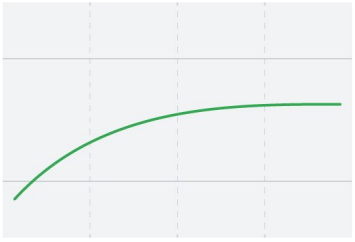


Highly peaked model

For example, a model that predicts the next word given the current word, which you might find in your mobile phone keyboard app, would be highly peaked because a small number of words account for the majority of words used.



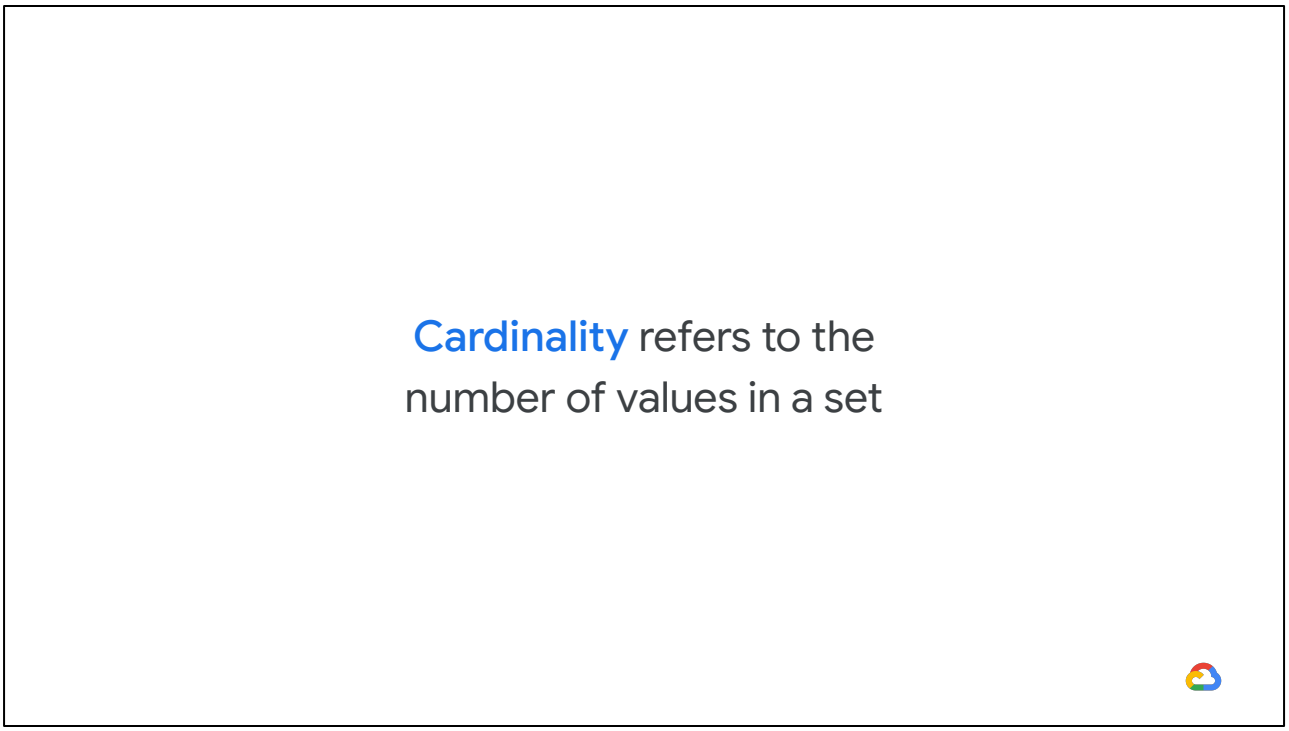
because a small number of words account for the majority of words used.



Flat model

In contrast, a model that predicted quarterly revenue for all sales verticals in order to populate a report would be run on the same verticals, and with the same frequency for each, and so it would be very flat.

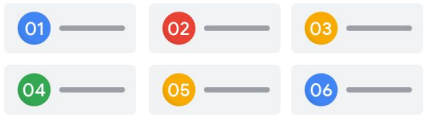
A report would be run on the same verticals, and with the same frequency



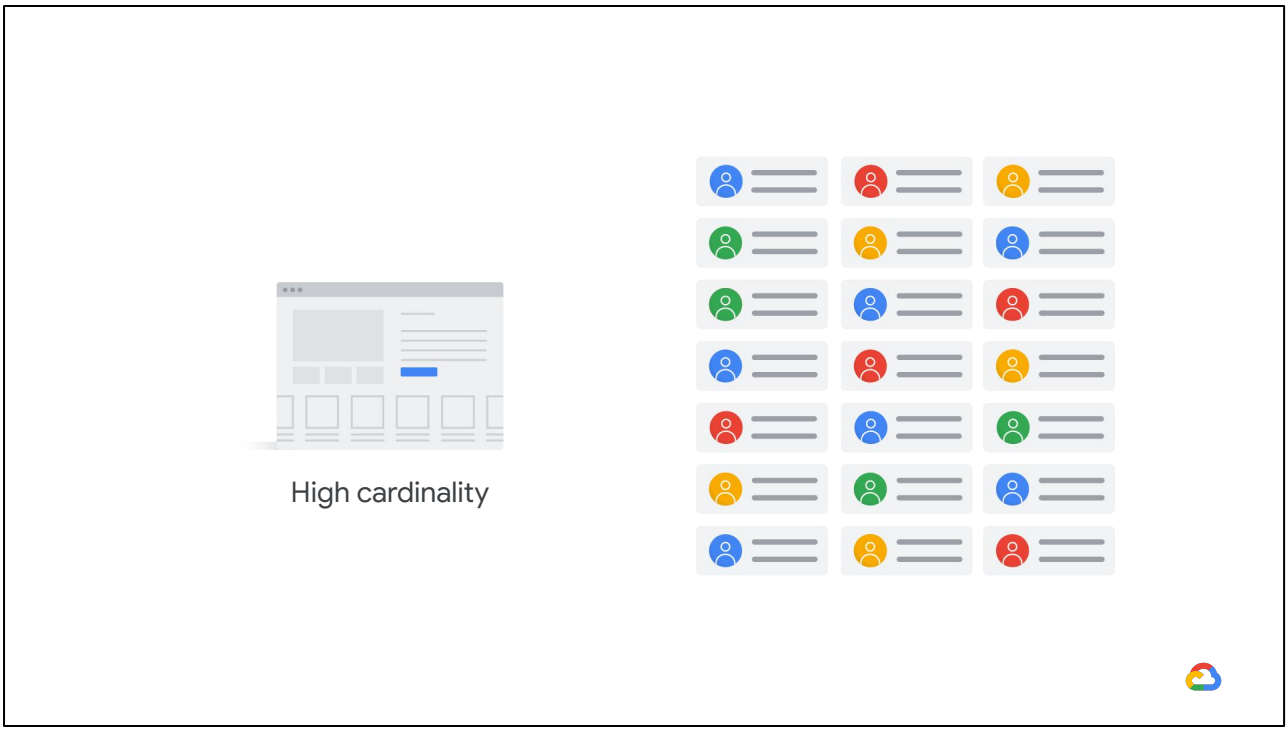
Cardinality refers to the number of values in a set.

In this case, the set is composed of all the possible things we might have to make predictions for.

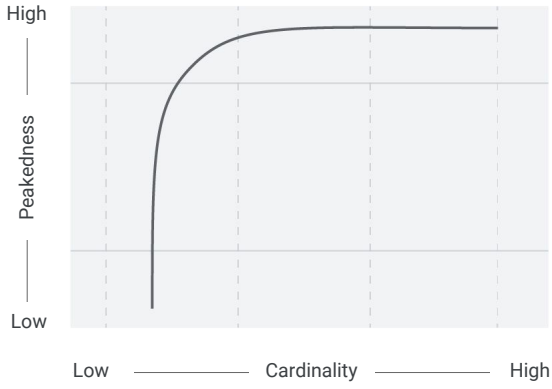
Low cardinality



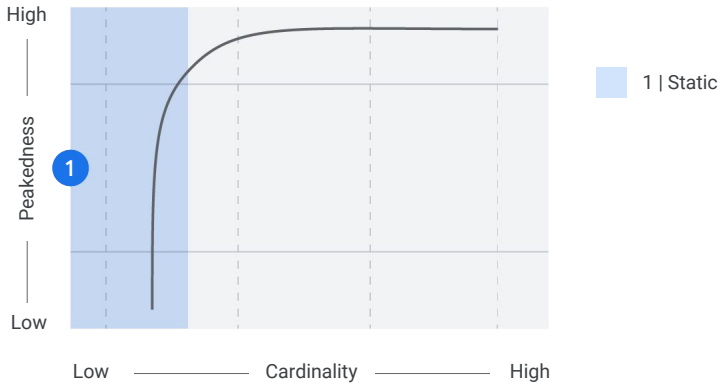
So, a model predicting sales revenue given organization division number would have fairly low cardinality.



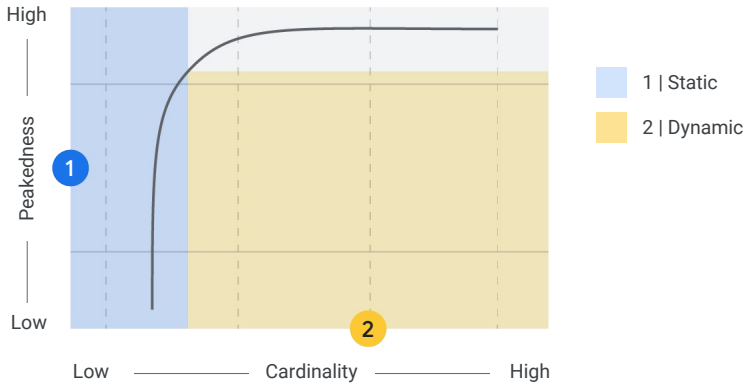
A model predicting lifetime value given a user for an ecommerce platform would be high cardinality because the number of users, and the number of characteristics of each user, are likely to be quite large.



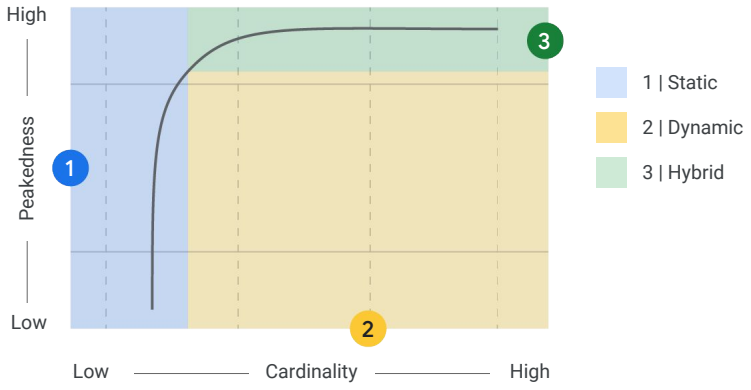
Taken together, peakedness and cardinality create a space.



When the cardinality is sufficiently low, we can store the entire expected prediction workload, for example, the predicted sales revenue for all divisions, in a table and use static serving.

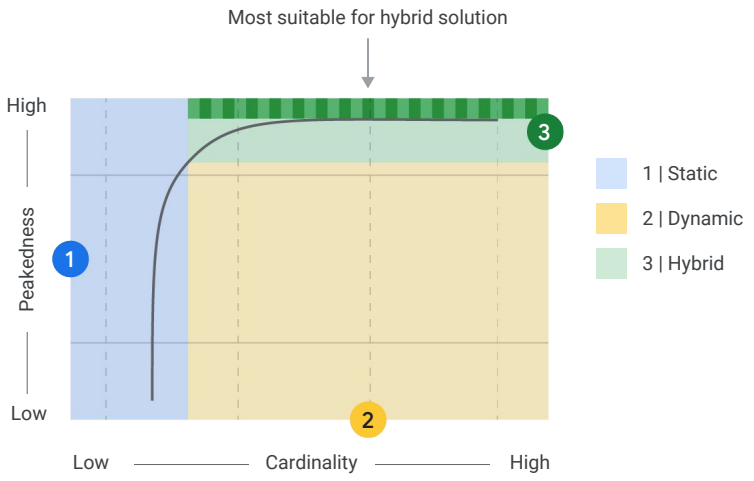


When the cardinality is high, because the size of the input space is large, and the workload is not very peaked, you probably want to use dynamic training.

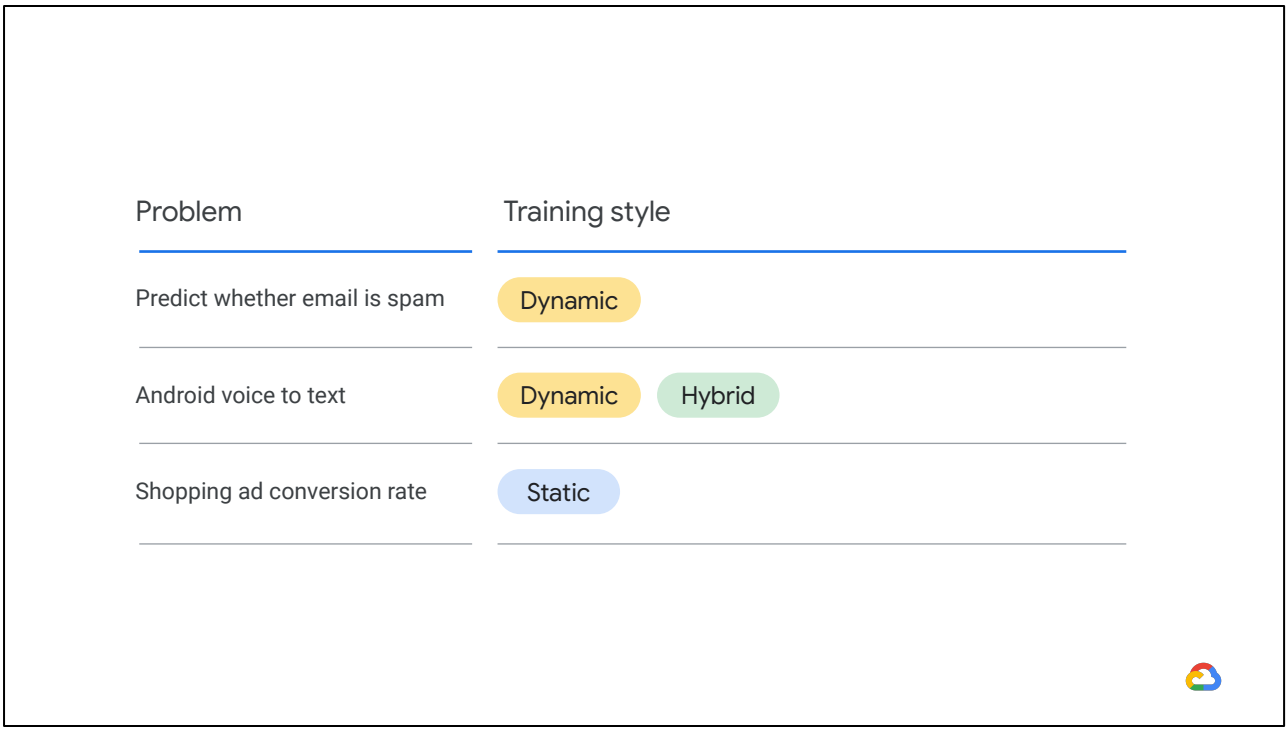


In practice, though, you often choose a hybrid of static and dynamic, where you statically cache some of the predictions while responding on-demand for the long tail.

This works best when the distribution is sufficiently peaked.



The striped area above the curve and not inside the blue rectangle is suitable for a hybrid solution, with the most frequently requested predictions cached and the tail computed on demand.

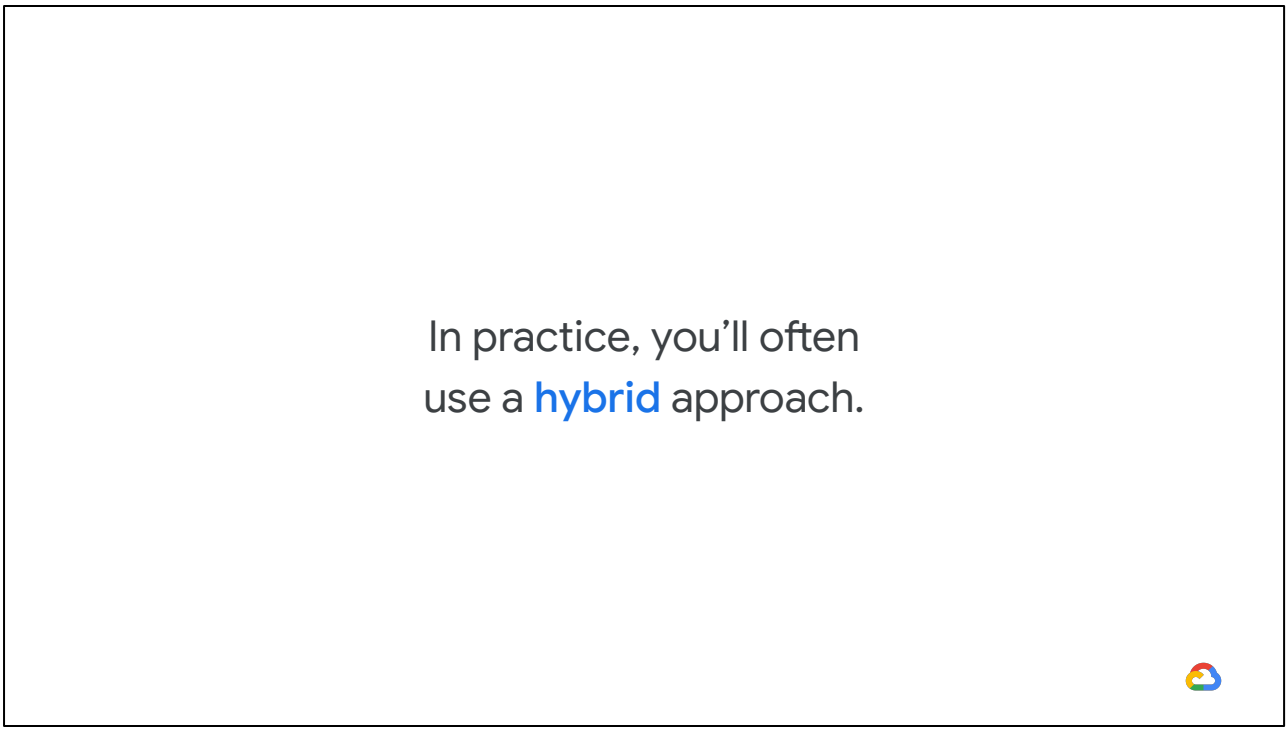


Let's try to estimate training and inference needs for the same use cases that we saw in the previous lesson.

- The first use case is predicting whether an email is spam. What inference style is needed? Well, first we need to consider how peaked the distribution is. The answer is *not at all*; most emails are likely to be different, although they may be very similar if generated programmatically. Depending on the choice of representation, the cardinality might be enormous.
  So, this would be **dynamic**.

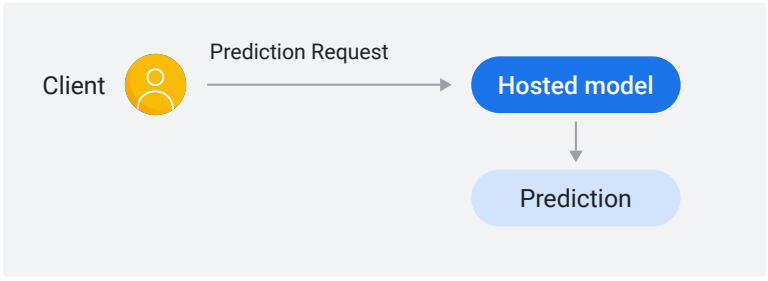
- The second use case is Android voice-to-text. This is again subtle. Inference is almost certainly online, since there's such a long tail of possible voice clips. But maybe with sufficient signal processing, some key phrases like "okay google" may have precomputed answers.
  So, this would be **dynamic or hybrid**.

- And the third use case is shopping ad conversion rate. The set of all ads doesn't change much from day to day. Assuming users are comfortable waiting for a short while after uploading their ads, this could be done statically, and then a batch script could be run at regular intervals throughout the day.
  This would be **static**.



In practice, you'll often use a hybrid approach.

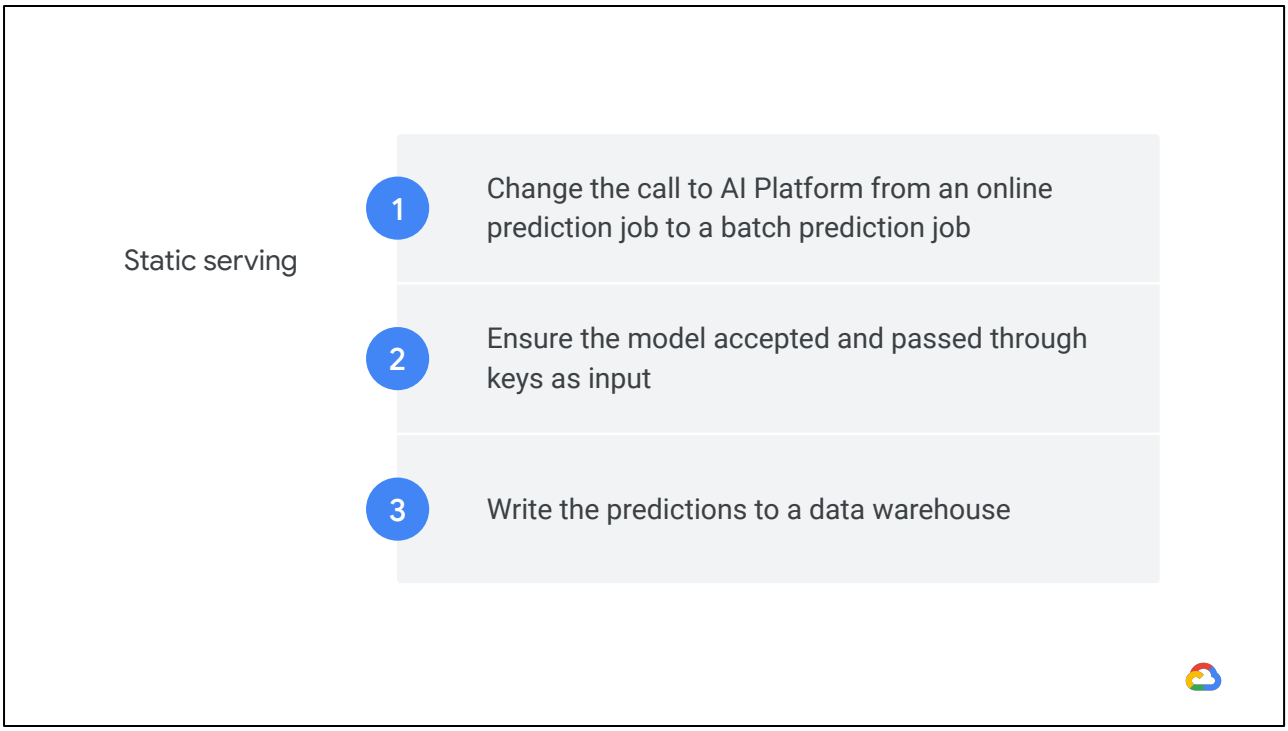
Dynamic serving




```
gcloud ai-platform predict --model $MODEL_NAME  \
                   --version $VERSION_NAME \
                   --json-instances $INPUT_DATA_FILE
```

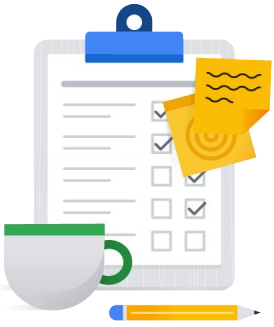
You might not have realized it, but dynamic serving is what we have learned so far.

Think back to the architecture of the systems we've used to make predictions: a model that lived in AI Platform was sent one or more instances and returned predictions for each.



If you wanted to build a static serving system, you would need to make three design changes.

- First, you would need to change your call to AI Platform from an online prediction job to a batch prediction job.
- Second, you'd need to make sure that your model accepted and passed through keys as input. These keys are what will allow you to join your requests to predictions at serving time.
- And third, you would write the predictions to a data warehouse, like BigQuery and create an API to read from it.



- Submitting a batch prediction job
- Enabling pass-through features
- Loading data into BigQuery

Although the details for each of these instructions are beyond the scope of this lesson, we've provided links in the course resources on:

1. Submitting a batch prediction job: https://cloud.google.com/ai-platform/prediction/docs/batch-predict

2. Enabling pass-through features in your model: https://towardsdatascience.com/how-to-extend-a-canned-tensorflow-estimator-to-add-more-evaluation-metrics-and-to-pass-through-ddf66cd3047d

3. And loading data into BigQuery: https://cloud.google.com/bigquery/docs/loading-data

Agenda

What's in a Production ML System

Training Design Decisions

Serving Design Decisions

AI Platform Notebooks

Designing an Architecture from Scratch

In this section, we'll apply what we've learned to a new use case.

Lab: Build a system that predicts the traffic levels on roads

Let's pretend the head of a municipal transit system has contacted you to build a system that predicts the traffic levels on roads.

As part of your preparation for this task, you're trying to thoroughly understand the business constraints in order to make the appropriate system design trade-offs.

Lab: Build a system that predicts the traffic levels on roads

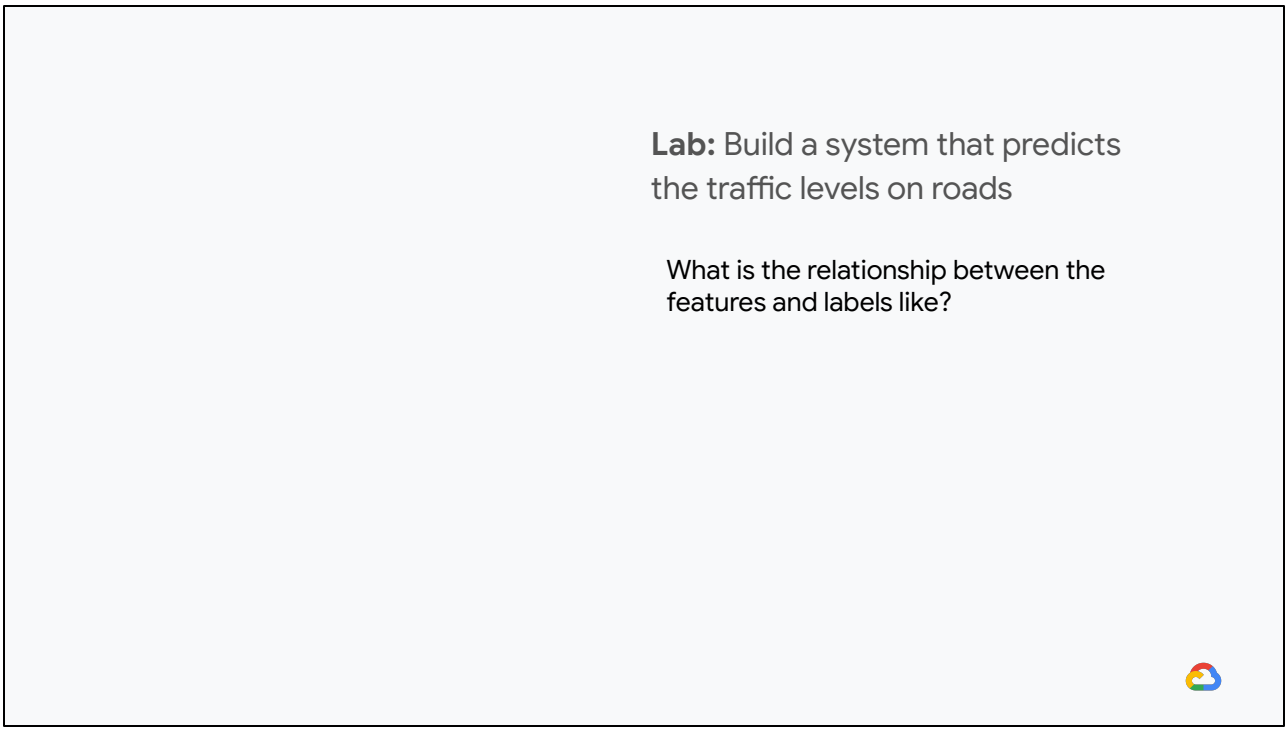
Available data: Traffic sensors deployed all over the city

The data available consist of sensors deployed all over the city which record whenever a car passes by. For each sensor, we know where it is. We also know the characteristics of the road that it's on.

Lab: Build a system that predicts the traffic levels on roads

What sort of training architecture is appropriate?

What sort of training architecture is appropriate?



Well, you should ask yourself, what is the relationship between the features and the labels? Is it more like physics or fashion?

In this case, it's more like fashion trends. Cities are very complex systems. If a train stops service, people will still need to get home. Technology is also always changing.

On-demand taxi services have reshaped urban transit in ways we didn't anticipate a decade ago. There are also episodic changes like sports events and parades, for example.

Lab: Build a system that predicts the traffic levels on roads

Which sort of serving architecture is appropriate?

For dynamic relationships, we need to use dynamic training.

Which serving architecture is appropriate?

Lab: Build a system that predicts the traffic levels on roads

Is the distribution of prediction requests likely to be more peaked or more flat?

Well, you should ask yourself, is the distribution of prediction requests likely to be more peaked or less peaked?

In this case is likely to be more peaked. The distribution of demand is peaked because it's likely to be dominated by the requests for the most heavily trafficked roads.

Lab: Build a system that predicts the traffic levels on roads

Is the cardinality of the set of all prediction requests likely to be low, moderate, high, need more info?

Is the cardinality of the set of all prediction requests likely to be low, moderate, high, or perhaps need more info?

In this case, you'll need more info.

Lab: Build a system that predicts the traffic levels on roads

Is the cardinality of the set of all prediction requests likely to be low, moderate, high, need more info?

What does it depend on?

A) Historical traffic data
B) Problem framing
C) Variance of Traffic Levels

Why is that? What does it depend upon?

Consider historical traffic data, problem framing or the variance of traffic levels.

Lab: Build a system that predicts the traffic levels on roads

Is the cardinality of the set of all prediction requests likely to be low, moderate, high, need more info?

What does it depend on?

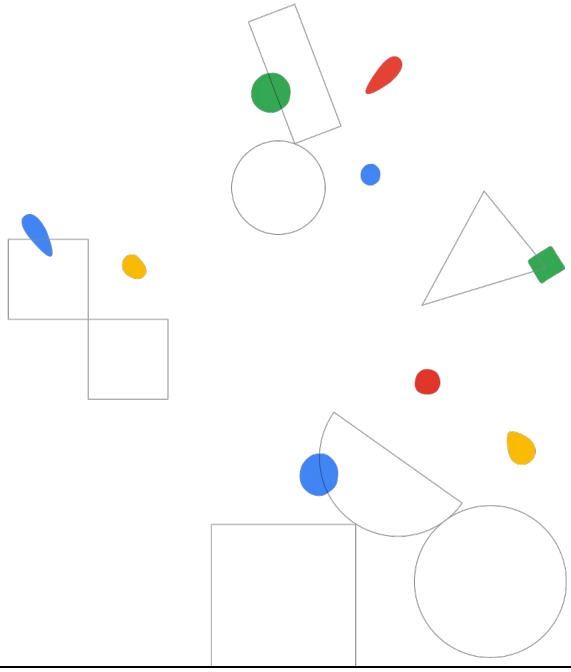
A) Historical traffic data
B) Problem framing
C) Variance of Traffic Levels

In this case, the answers are both historical traffic data and the problem framing, but not the variance of traffic levels. The reason that the cardinality depends upon the framing of the problem is that we don't know whether the task is to make predictions for every minute, hour, or day. Similarly, we don't know how big a region of space each prediction should correspond to. It could be anything from a few feet to a few blocks.

As we learned in the first specialization, machine learning is all about generalization, the leap of faith at the unseen input. What we don't know is whether our users want to generalize in space i.e by making predictions far away from the sensors in time, by making predictions in the future with finer granularity than the historical data or both. In all likelihood, you'd start conservatively, which corresponds to a lower rather than higher cardinality. Variants of traffic levels wouldn't matter because that's a label and not a feature.

Google Cloud

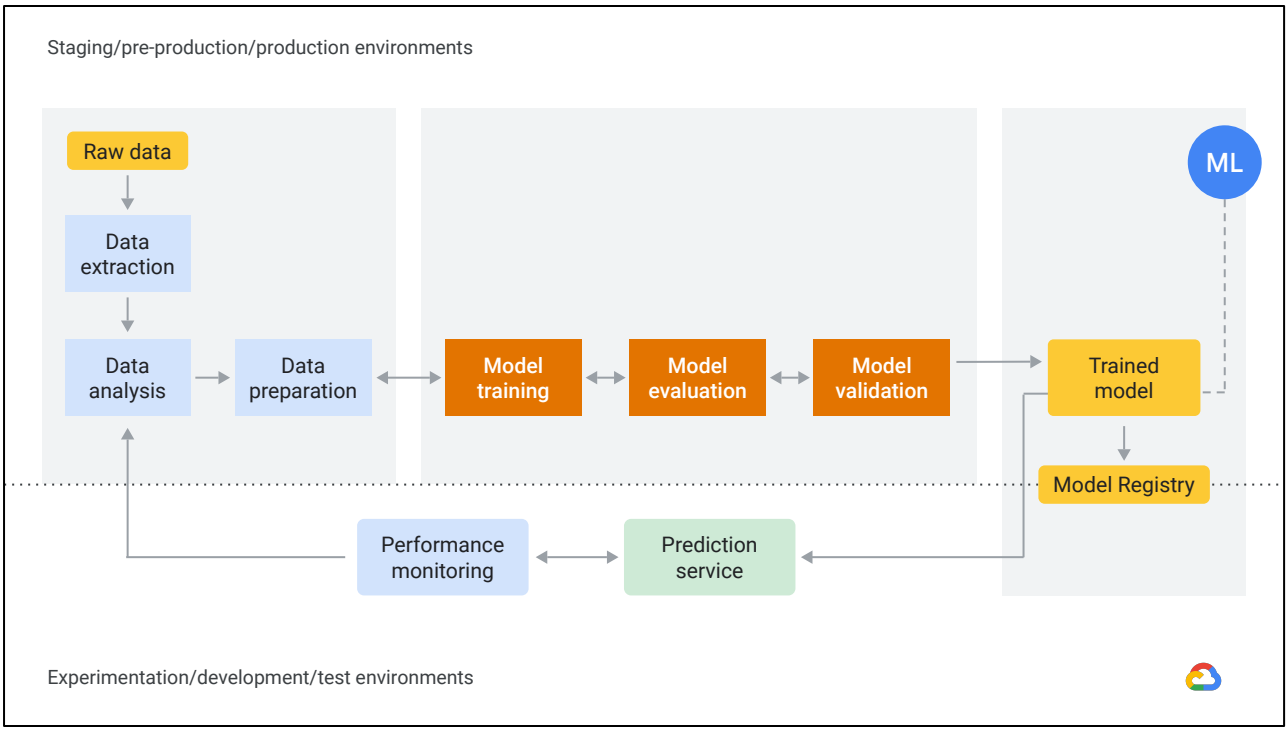
Using Vertex AI



Module 01

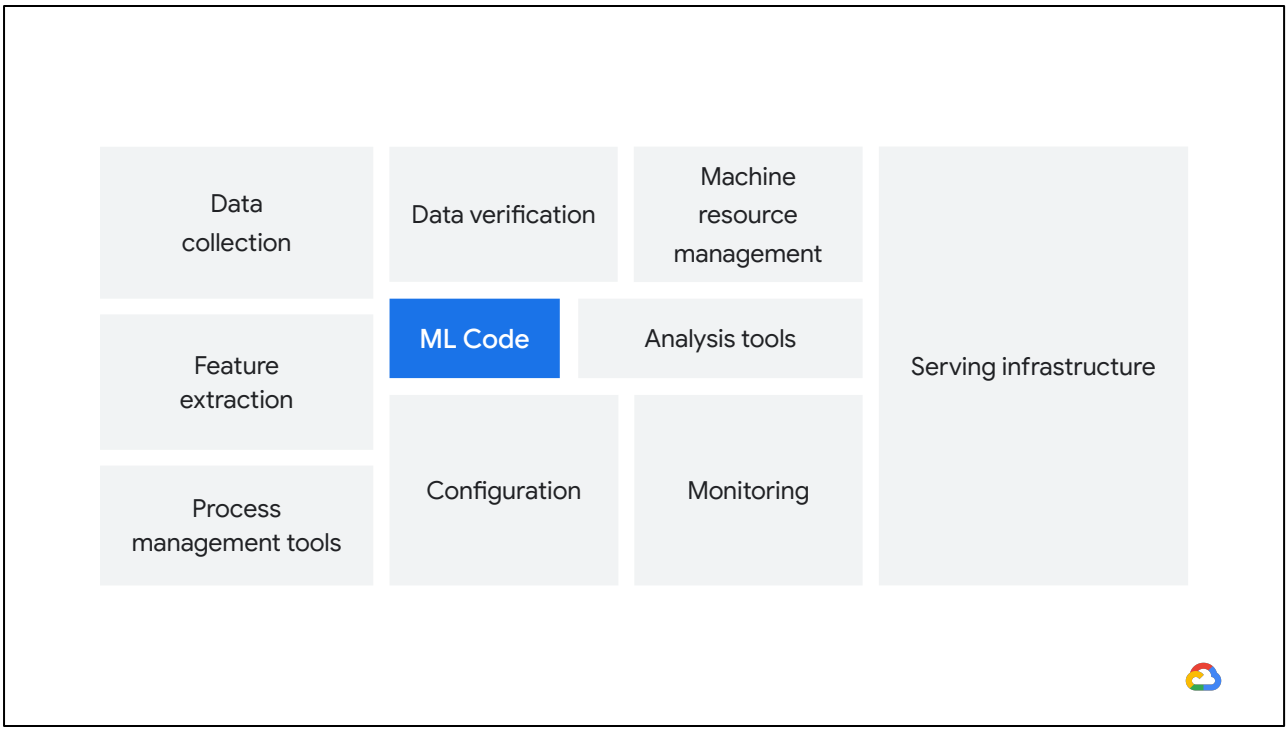
Architecting production ML systems

As you've seen from previous videos, the machine learning ecosystem requires decision making at every stage.



You need to determine how to handle and prepare data, and also how to design, build, evaluate, train, and monitor a model's performance.

Decisions around workflow processes, how to implement or execute those processes, and the management of the workflow itself are required to solve machine learning problems.



One of the most interesting details about the ML ecosystem is that ML code accounts for only a small percentage of it.

To keep a system running in production requires a lot more than just computing the model's outputs for a given set of inputs.

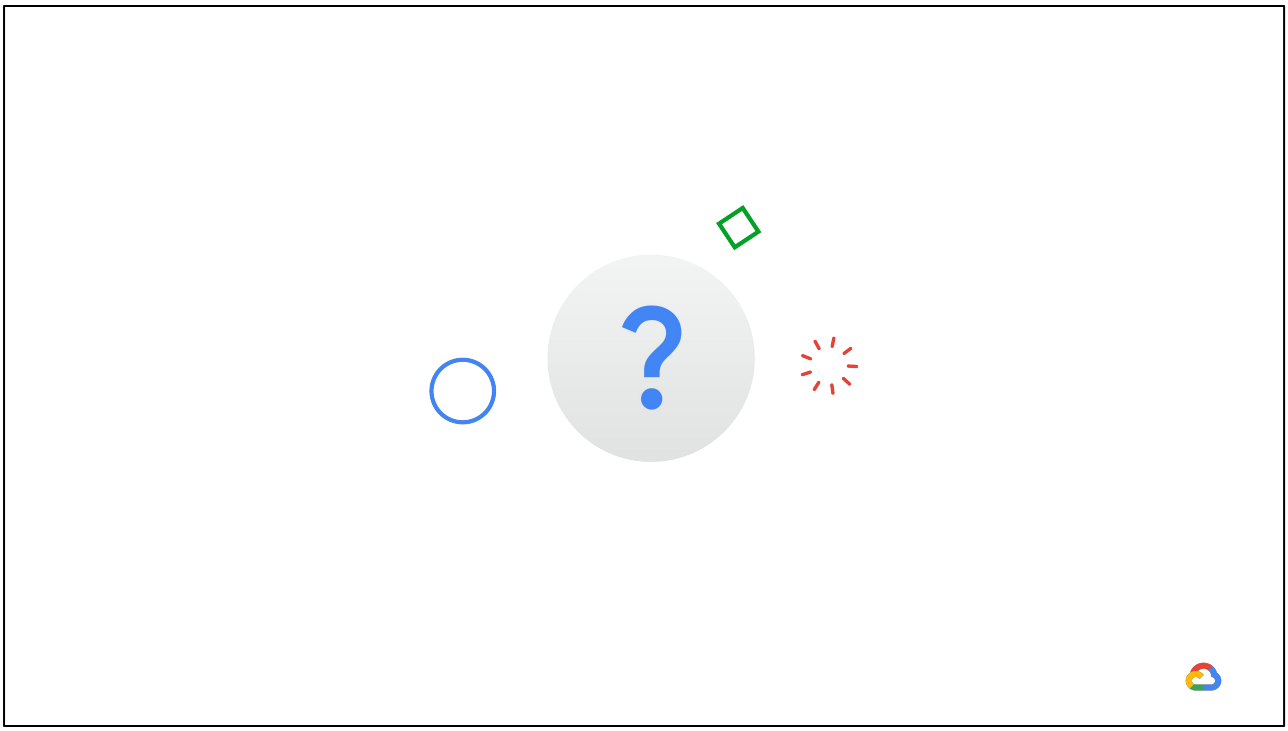
This means that each component of the ML ecosystem requires not only decisions and processes, but also people.

A lack of staff with the **right expertise**, a lack of **production-ready data**, and a lack of an **integrated development environment**

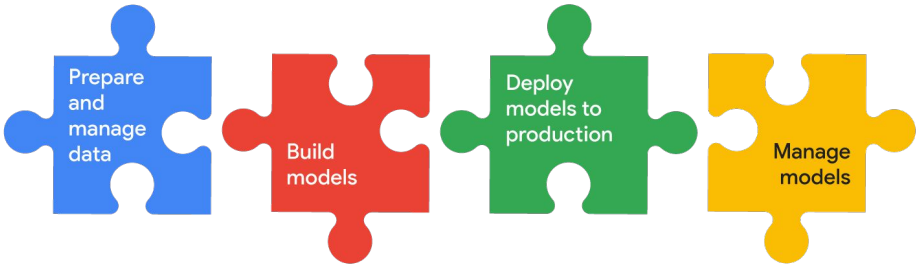
International Data Corporation, May 2020

According to the International Data Corporation, in 2020, a lack of staff with the right expertise, a lack of production-ready data, and a lack of an integrated development environment were reported as primary reasons that machine learning technologies fail.

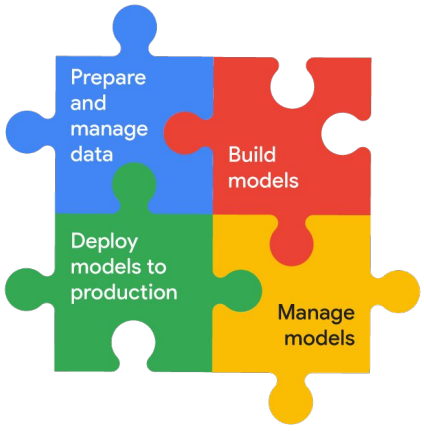
So, how do you ensure success for your machine learning or AI use case?



So, how do you ensure success for your machine learning or AI use case?



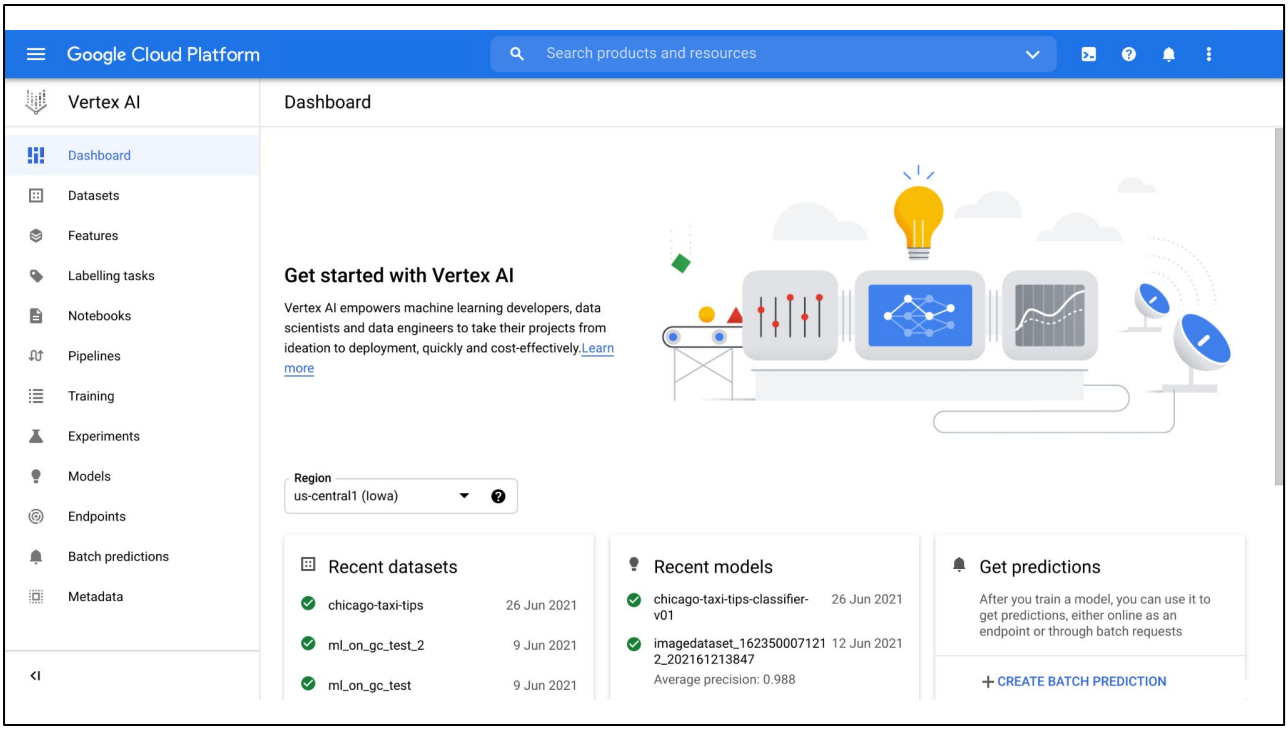
And how can you or your team prepare and manage your data, build your models, deploy them into production, and then manage them?



A solution is to use a unified platform that brings all the components of the machine learning ecosystem and workflow together.

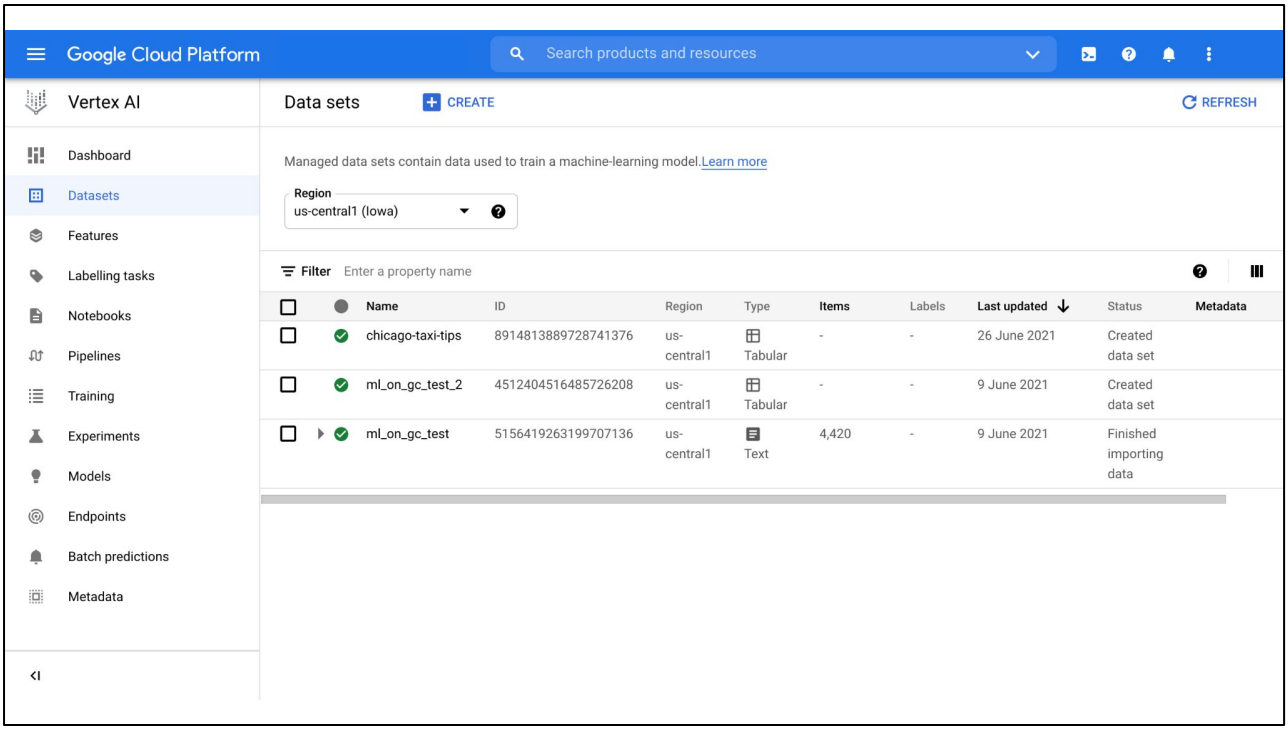
And in this case, that platform is Vertex AI.

Vertex AI brings together the Google Cloud services for building ML under one unified user interface and application programming interface, or API.



With Vertex AI, you can access a, dashboard, datasets, features, labeling tasks, notebooks, pipelines, training, experiments, models, endpoints, batch predictions, and metadata.

Let’s take a closer look at the Datasets, Notebooks, Training, and Models sections of the Vertex AI navigation bar that help you to prepare your data and build and deploy your models.



Vertex AI has a unified data preparation tool that supports image, tabular, text, and video content. Uploaded datasets are stored in a Cloud Storage bucket that acts as an input for both AutoML and custom training jobs.

row	movie_id	average_rating	title	genres	update_time
1	movie_01	4.9	The Shawshank Redemption	Drama	70-01-01 00:26:19
2	movie_02	4.2	The Shining	Horror	70-01-01 00:26:19
3	movie_03	4.5	Cinema Parasado	Romance	70-01-01 00:26:19
4	movie_04	4.6	The Dark Knight	Action	70-01-01 00:26:19

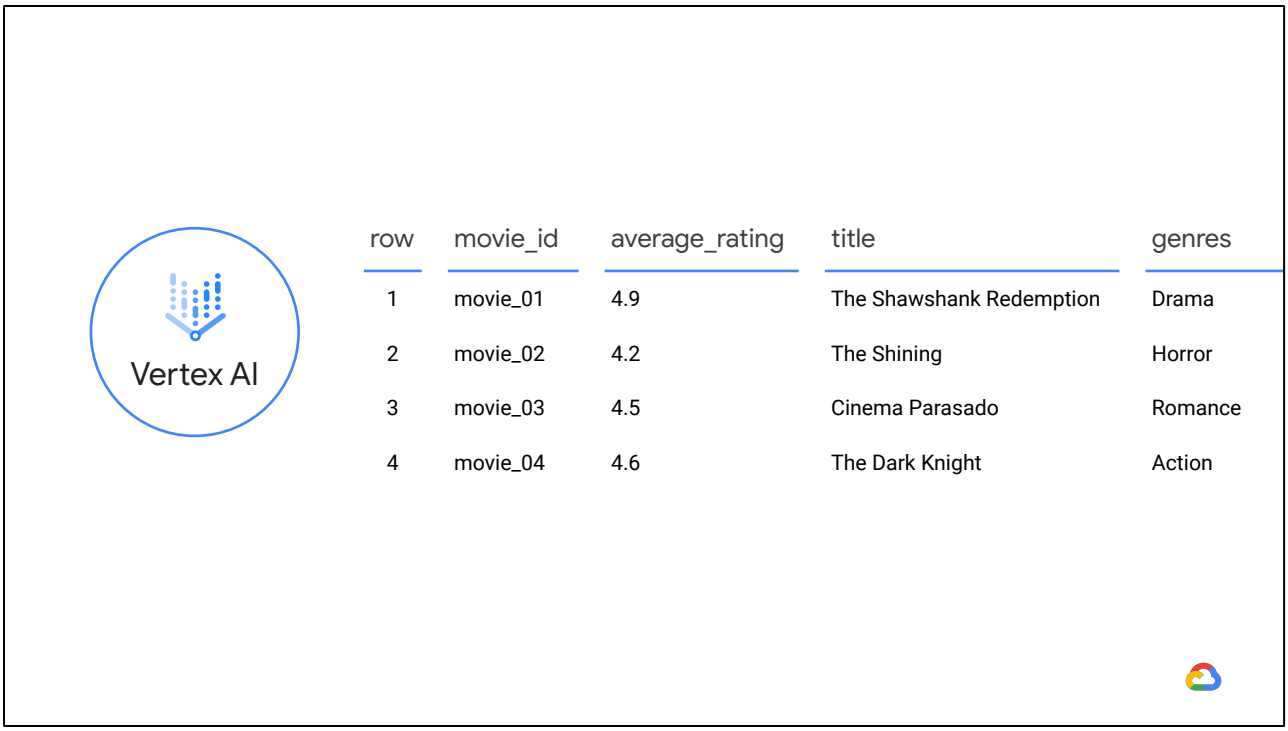
Let’s explore an example where you have sample source data from a BigQuery table about movies and their features.

First, there is a movie_id column header that can map to an entity type called *movie*. For each movie entity, features include an average_rating, title, and genres.

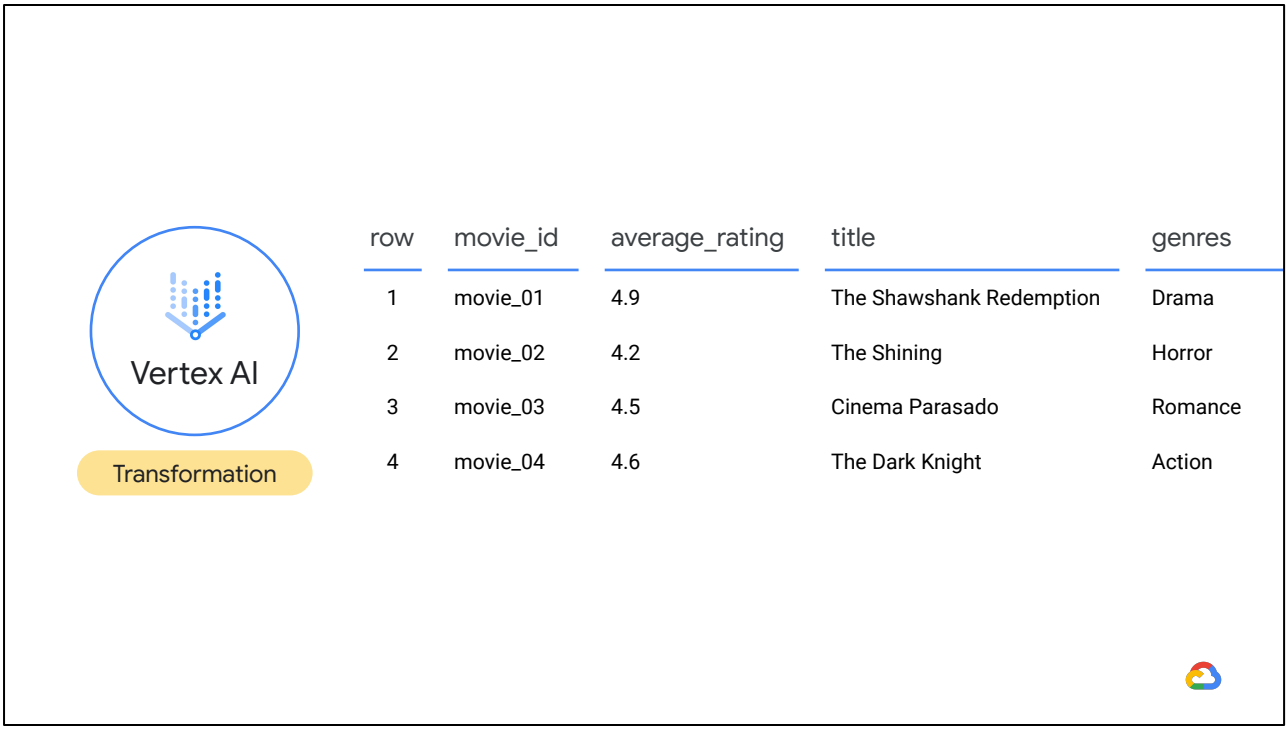
The values in each column map to specific instances of an entity type or features, which are called entities and feature values.

The update_time column indicates when the feature values were generated.

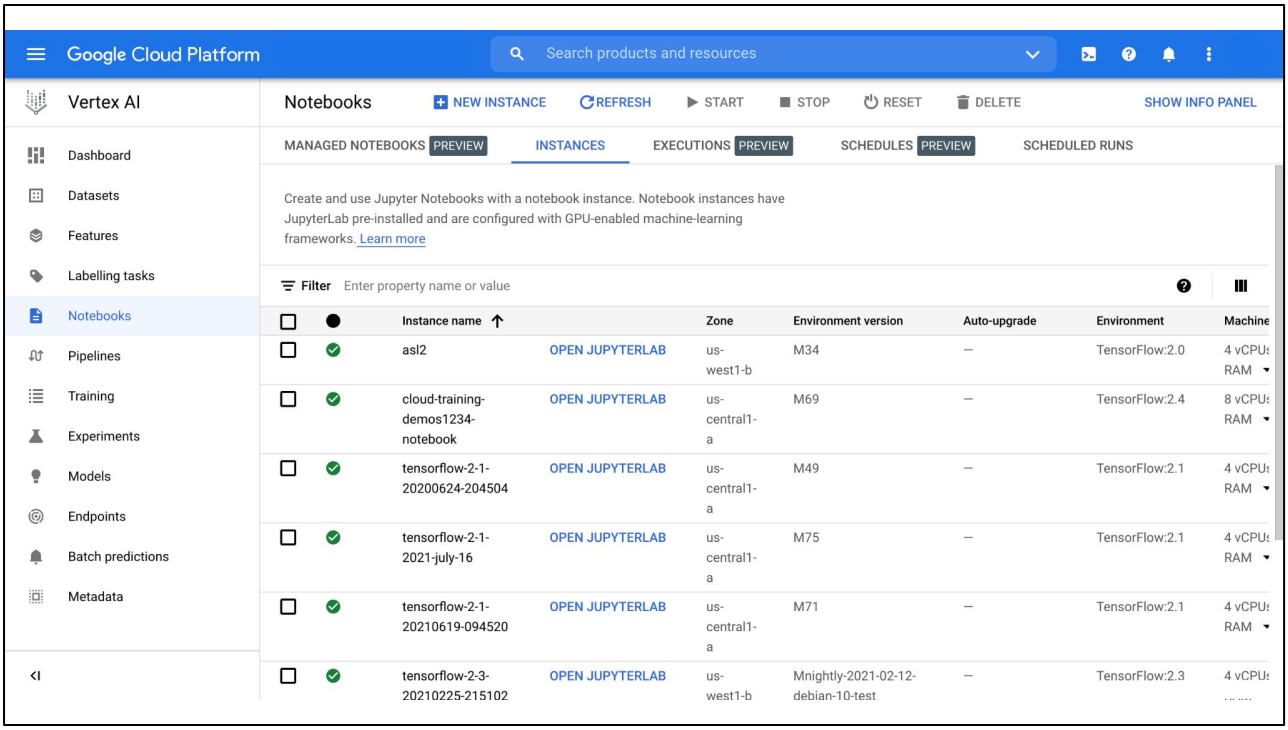
In the featurestore, the timestamps are an attribute of the feature values, not a separate resource type. If all feature values were generated at the same time, you don't need to have a timestamp column. You can specify the timestamp as part of your ingestion request.



When you use the data to train a model, Vertex AI examines the source data type and feature values and infers how it will use that feature in model training.

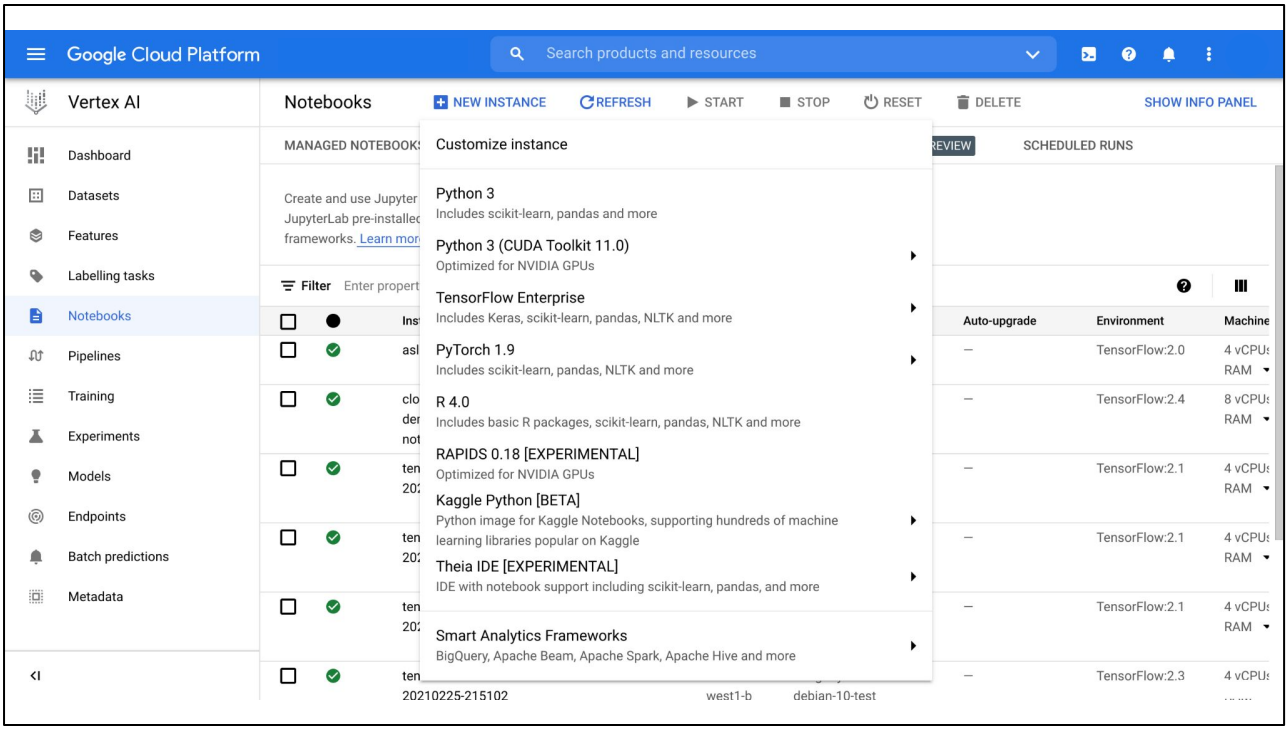


This is called the *transformation* for that feature. If needed, you can specify a different supported transformation for any feature.



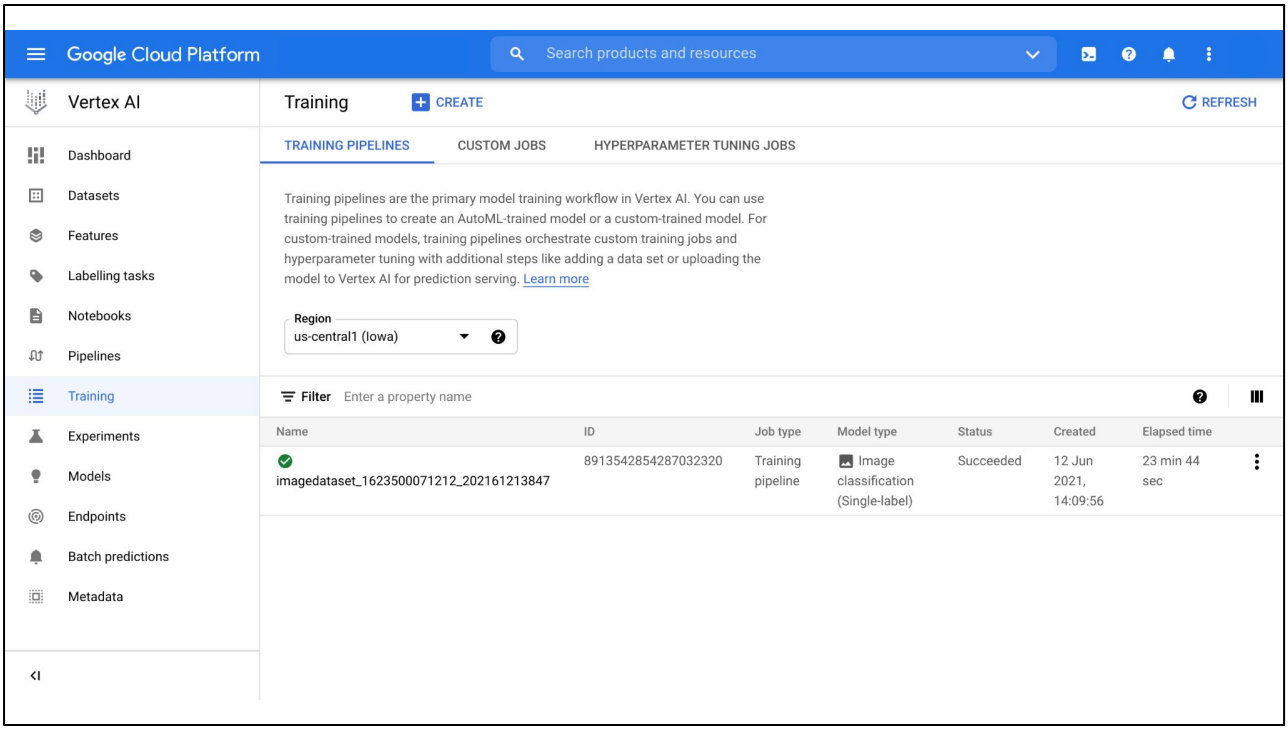
After you prepare your dataset, you can develop models using Notebooks. Notebooks is a managed service that offers an integrated and secure JupyterLab environment for data scientists and machine learning developers to experiment, develop, and deploy models into production.

Notebooks enable you to create and manage virtual machine (VM) instances because they come pre-installed with the latest data science and machine learning frameworks.



They also come with a pre-installed suite of deep learning packages, including support for the TensorFlow and PyTorch frameworks. Either can be configured for CPU-only or GPU-enabled instances.

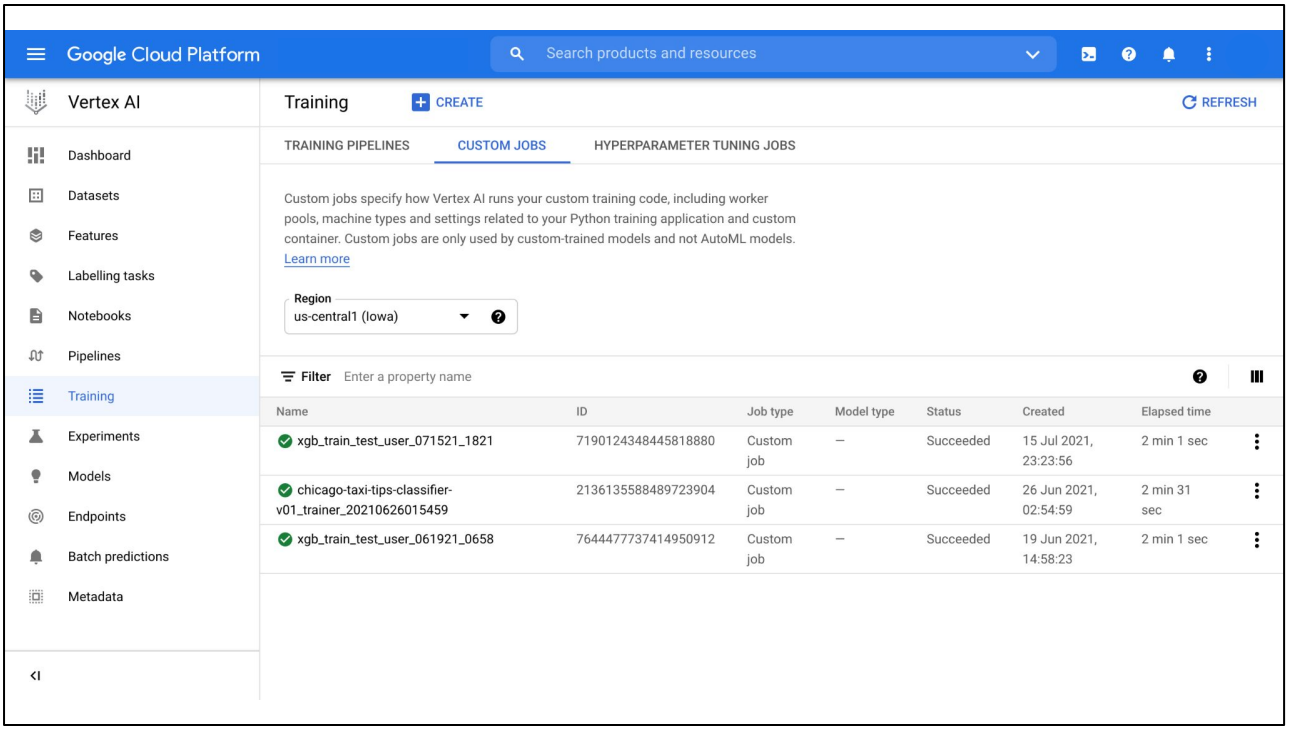
With regard to security, Notebooks instances are protected by Google Cloud authentication and authorization and are available using a Notebooks instance URL, which is part of the metadata of the VM.



Now let's shift our focus to training.

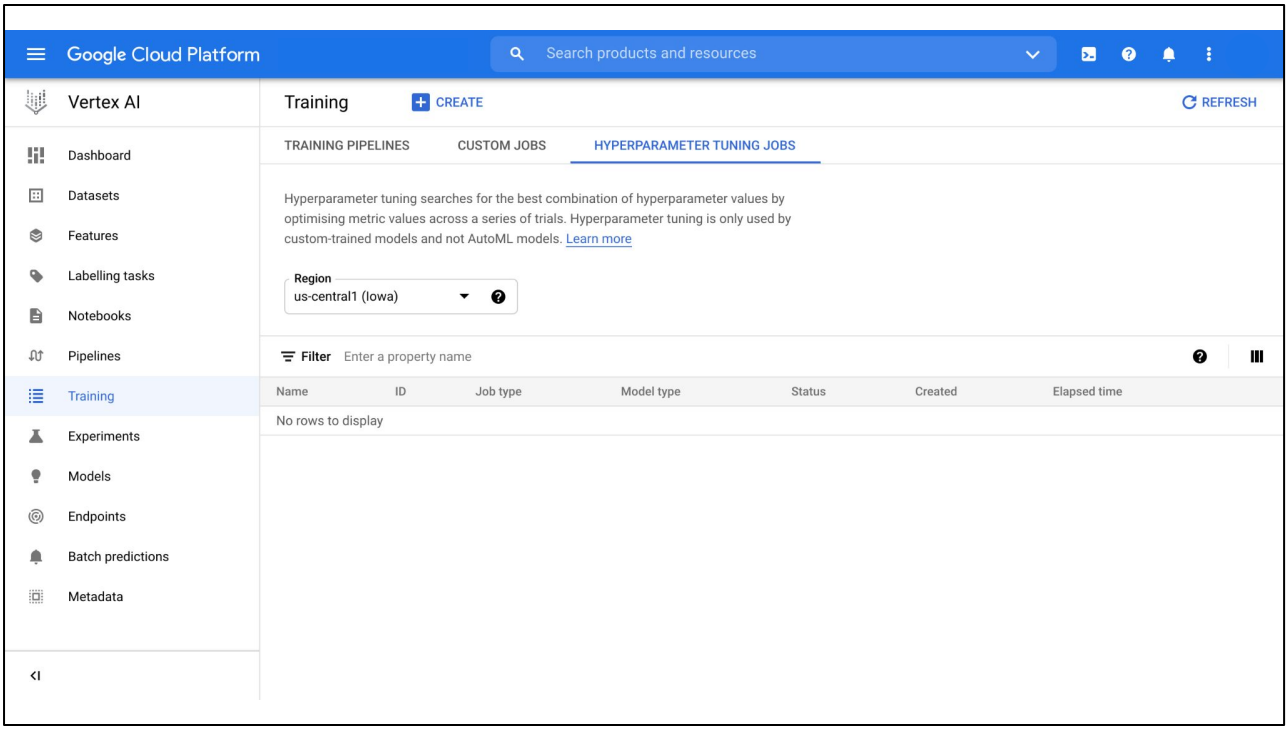
With Vertex AI, you can train and compare models using AutoML or custom code training, with all models stored in one central model repository.

Training pipelines are the primary model training workflow in Vertex AI, which can use training pipelines to create an AutoML-trained model or a custom-trained model.



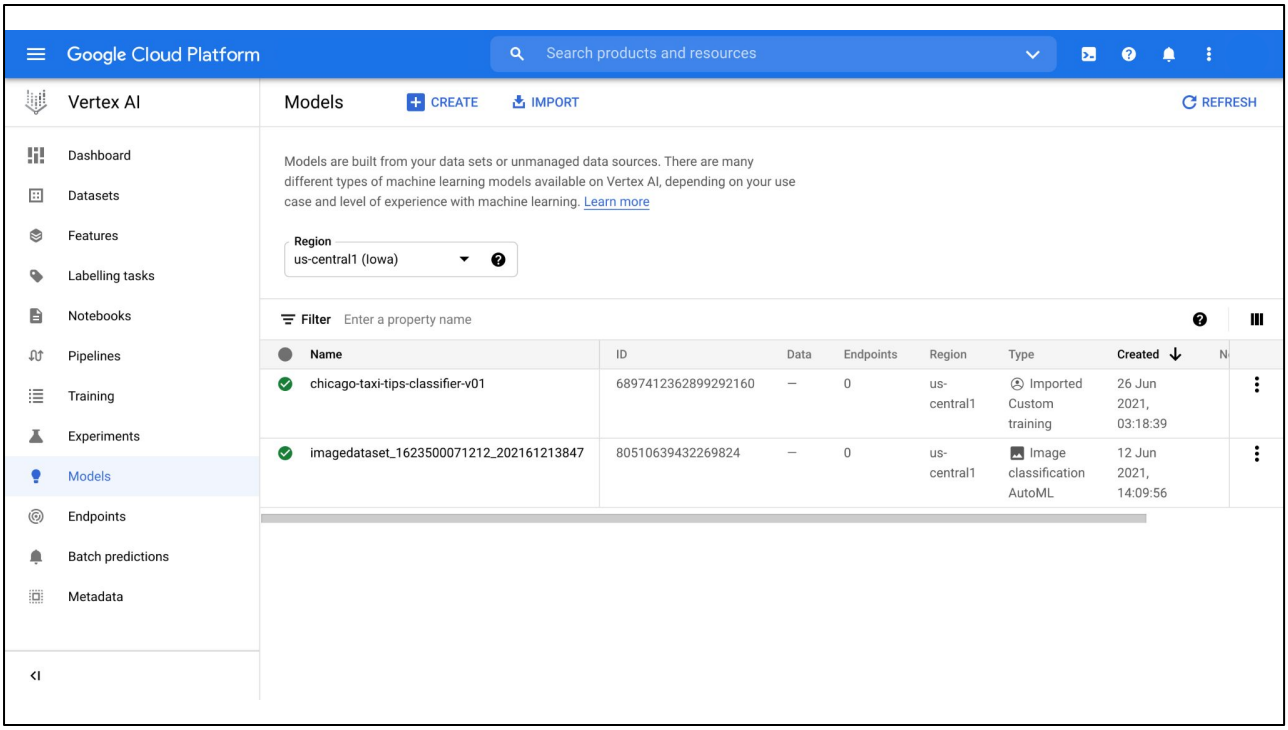
For custom-trained models, training pipelines orchestrate custom training jobs and hyperparameter tuning in conjunction with steps like adding a dataset or uploading the model to Vertex AI for prediction serving.

Custom jobs specify how Vertex AI runs custom training code, including worker pools, machine types, and settings related to a Python training application and custom container.



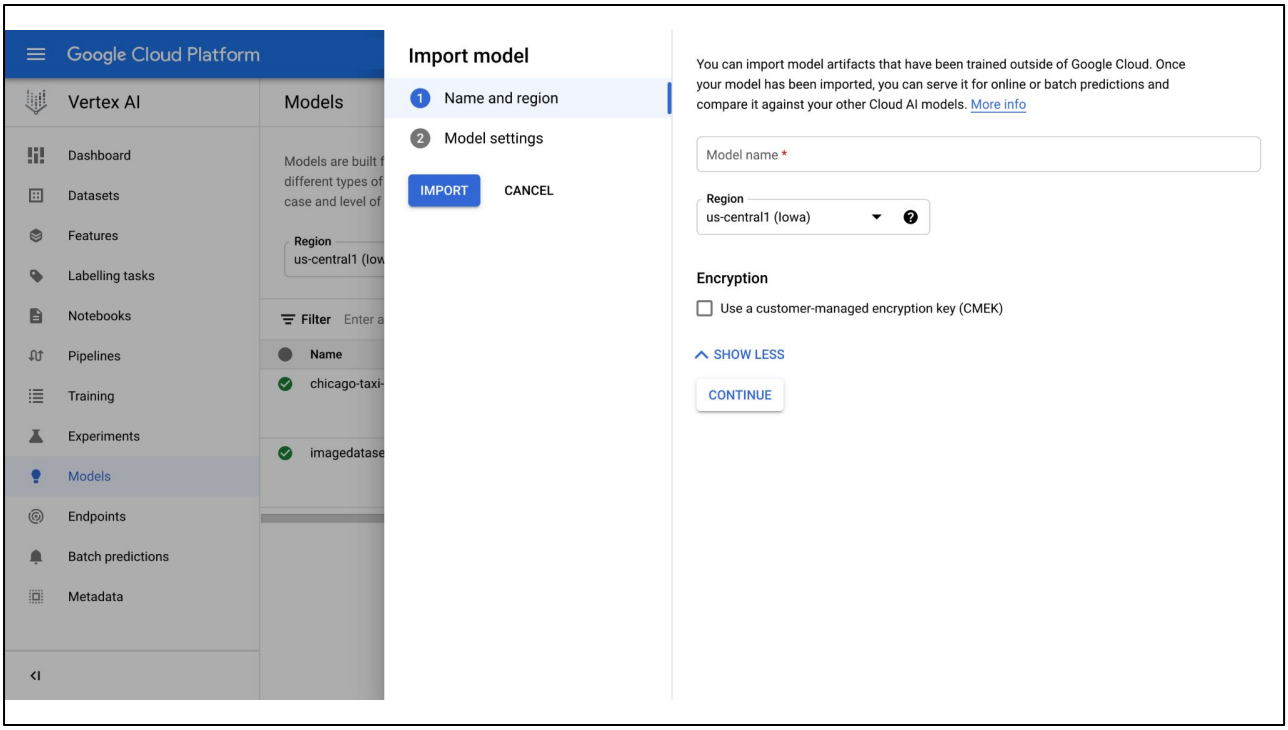
Alternatively, hyperparameter tuning searches for the best combination of hyperparameter values by optimizing metric values across a series of trials.

Both custom jobs and hyperparameter tuning, however, are only used by custom-trained models. They are not used by AutoML models.



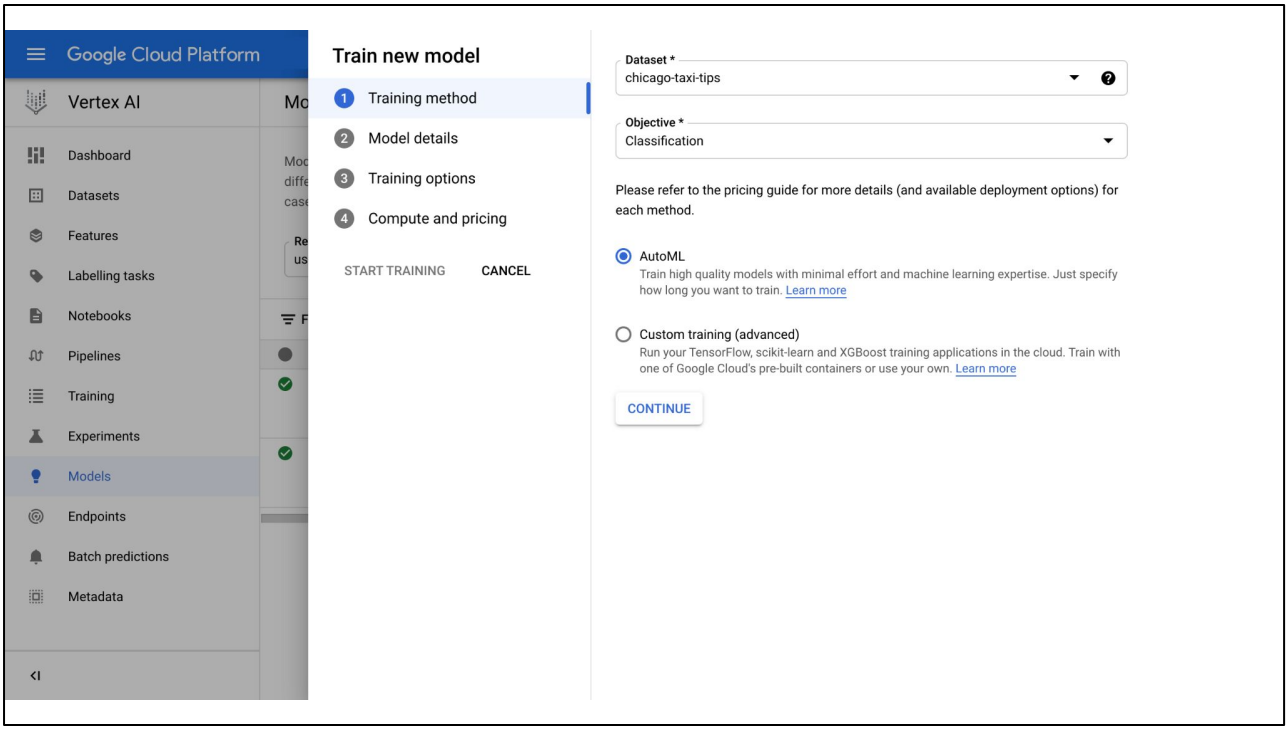
Next up are models.

Models are built from datasets or unmanaged data sources. Many different types of machine learning models are available with Vertex AI. The right choice will depend on the use case and your level of experience with machine learning.



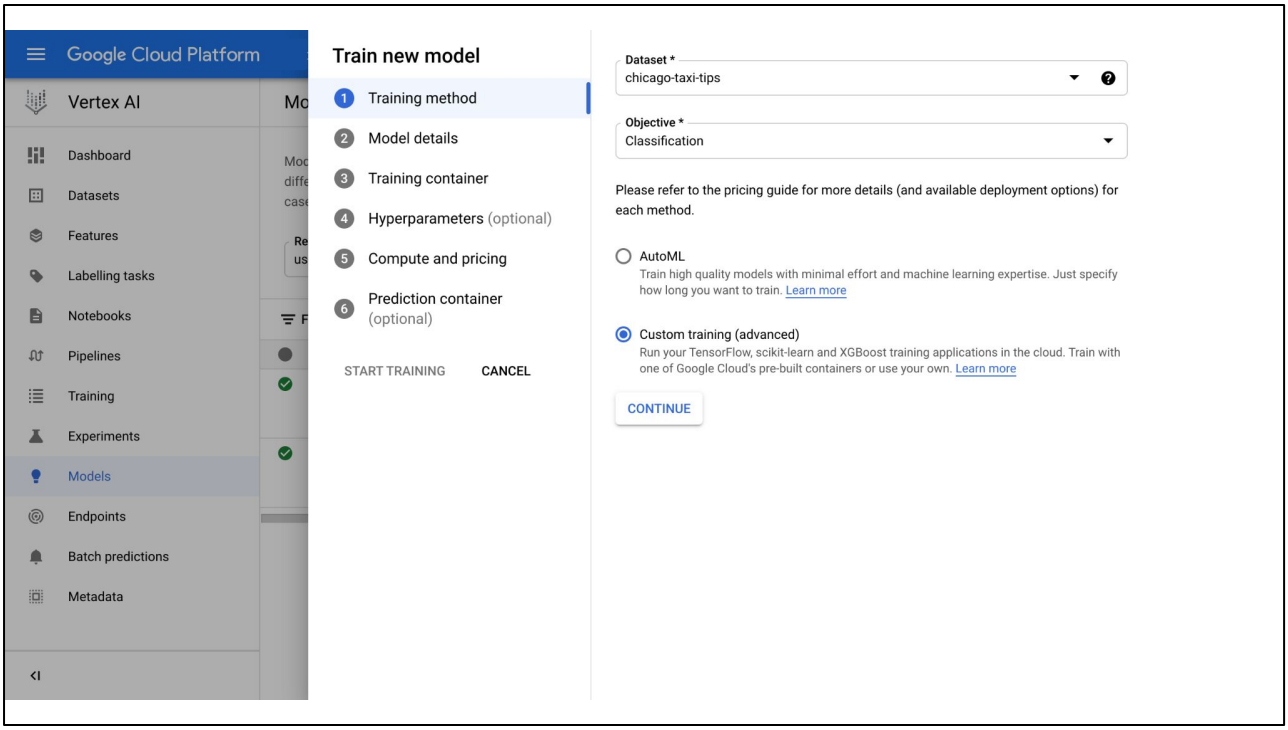
A new model can be trained, or an existing model can be imported.

After the model has been imported into Vertex AI, it can be deployed to an endpoint and then used to request predictions.

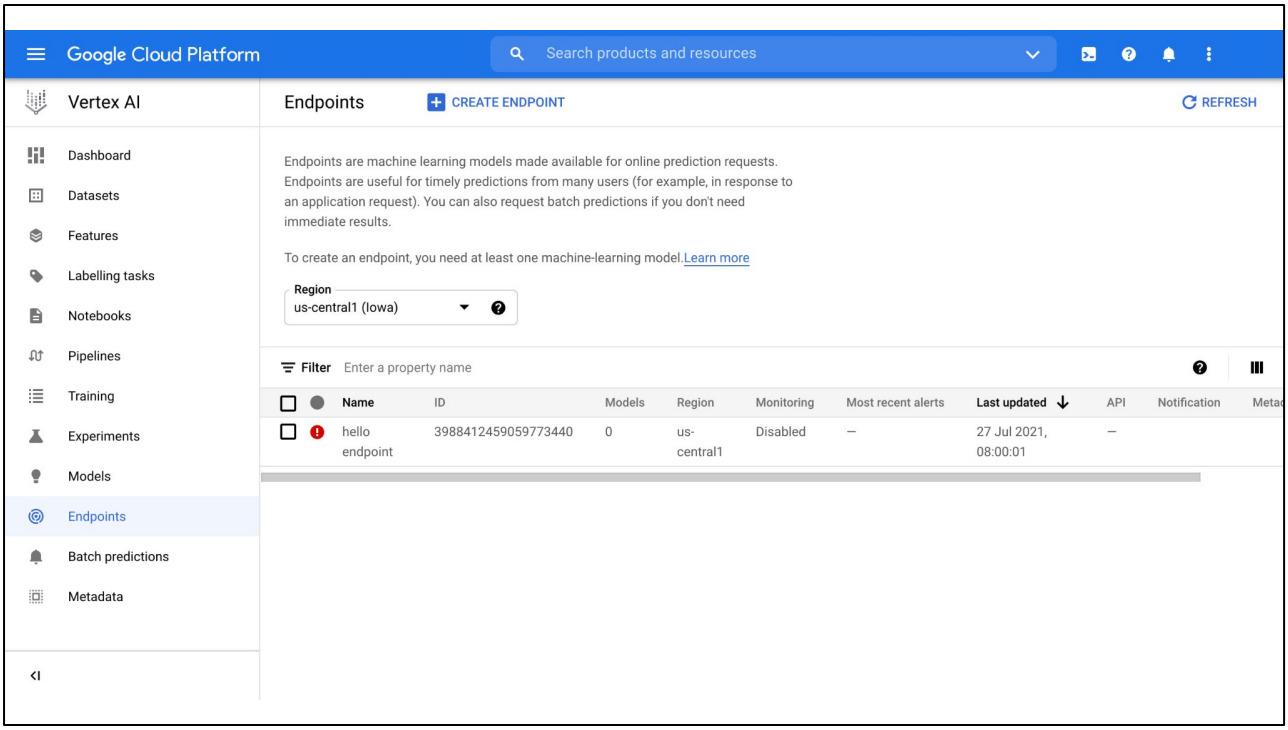


AutoML can be used to train a new model with minimal technical effort. It can be used to quickly prototype models and explore new datasets before investing in development.

For example, you might use AutoML to determine the good features in a dataset.



Generally speaking, custom training is used to create a training application optimized for a targeted outcome, because it allows for complete control over training application functionality. You can target any objective, use any algorithm, develop your own loss functions or metrics, or carry out any other customization.

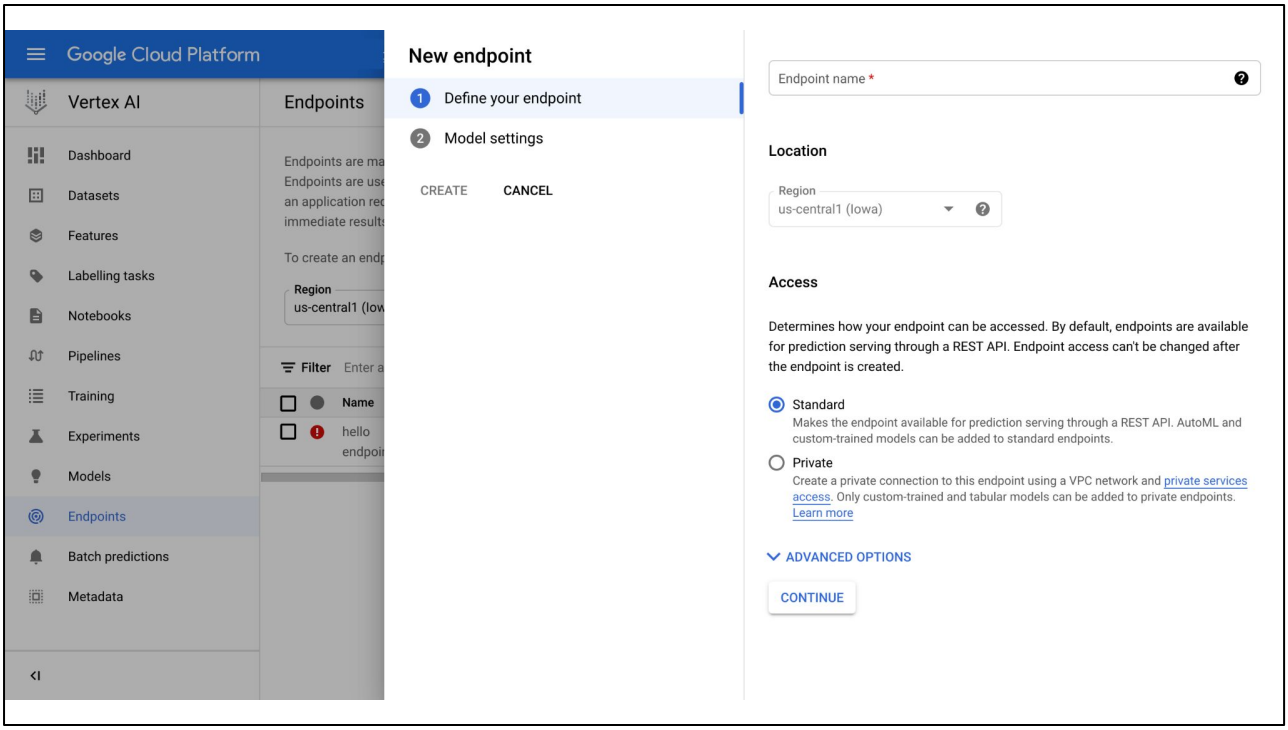


And finally, let's explore Endpoints.

Endpoints are machine learning models made available for online prediction requests. An endpoint is an HTTPS endpoint that clients can call to receive the inferencing (scoring) output of a trained model. They can provide timely predictions from many users, for example, in response to an application request. They can also request batch predictions if immediate results aren't required.

Multiple models can be deployed to an endpoint, and a single model can be deployed to multiple endpoints to split traffic. You might deploy a single model to multiple endpoints to test out a new model before serving it to all traffic.

Either way, it's important to emphasize that a model must be deployed to an endpoint before that model can be used to serve online predictions.

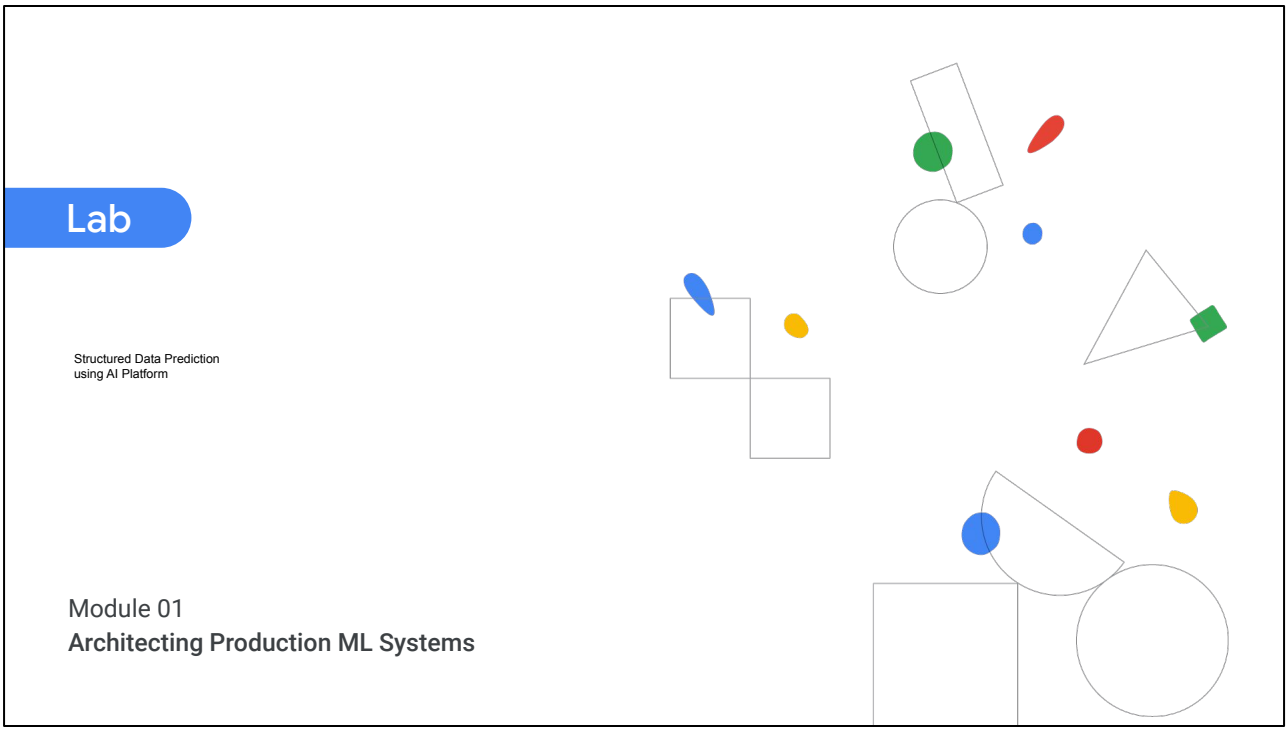


To make that happen, you must define an endpoint in Vertex AI by giving it a name and location and deciding whether the access is Standard, which makes the endpoint available for prediction serving through a REST API.

Vertex AI

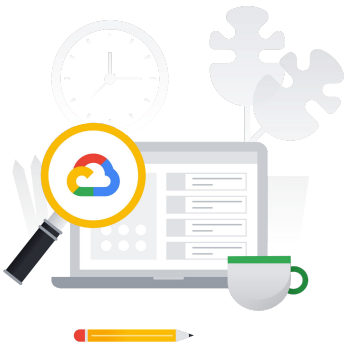
cloud.google.com/vertex-ai

This has been a brief introduction to Vertex AI, Google Cloud's unified ML platform.
For more information, please see cloud.google.com/vertex-ai.



This lab provides hands-on practice using Google Cloud's AI Platform.

Lab objectives



- Create a BigQuery Dataset
- Export the data into a GCS bucket and train using Cloud AI Platform
- Deploy the trained model using Cloud AI Platform

You'll start by creating a BigQuery Dataset, then Export the data into a GCS bucket and train using Cloud AI Platform.

And finally, deploy the trained model using Cloud AI Platform.