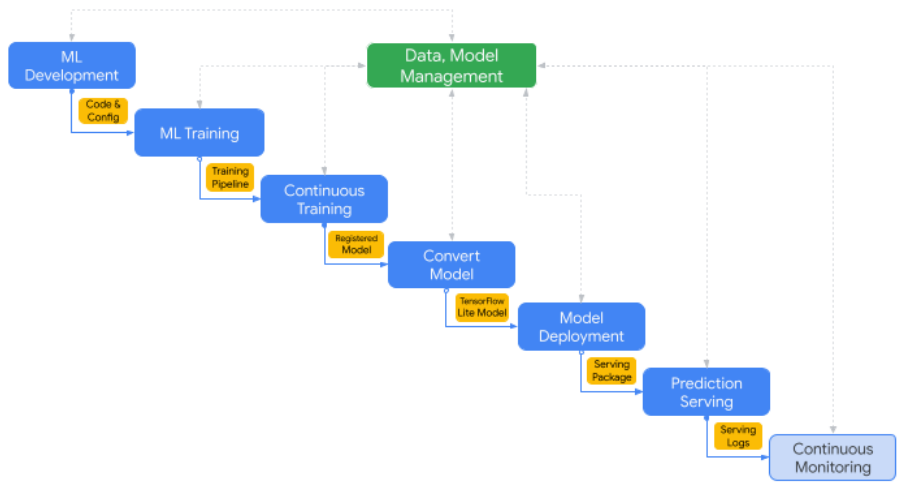
Overview of Continuous Monitoring



Objective

Once we have successfully deployed a model to our production environment, the next stage is to monitor its performance. This stage is often called continuous monitoring and is of great importance in MLOps. Primarily, continuous monitoring is done to validate model performance, often to ensure that it does not degrade over time. If the incoming data distribution gradually deviates over time, the model performance will gradually degrade due to a mismatch between the training and inference data distributions. In addition, there may be scenarios where other coupled services might experience errors or changes which result in poor predictions being outputted by our model. Continuous monitoring allows us to monitor our predictions and ensure that things are working as anticipated, and can raise alerts if any unexpected conditions occur. To this end, in this section, we will learn the following concepts:

- Problem with Model Drift
    - Concept drift
    - Data drift
    - Addressing drift
- Challenges with Continuous Monitoring for TinyML
- Potential solution via Federated Machine Learning

Motivation

As an example of continuous monitoring, an e-commerce model predicting a customer's propensity to buy a product might be expected to degrade over time as consumer preferences change. In such a situation, retraining triggers might be added where a specific metric is monitored and, if model performance drops below a designated threshold, then the model is retrained using the most up-to-date data. Having retraining triggers may be preferable to continuous training in some scenarios, especially if models are large and computationally intensive to train. To achieve this, a continuous monitoring process consists of the following:

Logging. Predictions served to the user are logged and designated “serving logs”. These predictions may be stored in their entirety or a subset of predictions may be stored if deemed suitable based on the context.

Periodic Checks. The monitoring engine checks the serving logs periodically and computes statistics for the serving data, such as the distribution of served values.

Serving Comparison. The schema generated by the monitoring engine is then compared to a “reference” schema which represents what we would expect from our prediction serving distribution. This stage allows the detection of distribution skews between the served data and the reference schema. In addition, if true labels are available for the serving data, these can be used to evaluate the effectiveness of the deployed model’s predictions.

Error/Anomaly Handling. In the event that an anomaly or error occurs, or a threshold is passed indicating decaying performance of a model, flags can be raised. For example, a decayed model might send an email to the team responsible for monitoring models with relevant information, such that the team can assess the error and decide whether any interventions are necessary.

Model Decay

Typically, model decay is described in terms of concept drift and data drift.

- **Data drift** describes a growing skew between the data distribution used for training, and that currently exhibited in the newly served predictions.
- **Concept drift** refers to an evolving relationship between the input parameters and the output of our model (i.e., the data distribution might be the same, but the fundamental mapping between the input data and our target variable has changed).

Metrics

Different techniques can be used to evaluate the presence of concept drift vs. data drift. Data drift might be assessed using novelty or outlier detection, while concept drift might be assessed using feature attributional changes. Common metrics that are monitored by the continuous monitoring process are:

- **Resource utilization.** This refers to the usage of CPU, GPU, and memory resources.
- **Latency.** This refers to how fast predictions are served for streaming deployments and is a good indication of model service health. Generally, you would prefer there to be a small latency between a user request and the model’s response. Latency may even inadvertently impact the data distribution (you could imagine not purchasing an item because a model is running too slowly on a website and you lose interest).

- **Throughput.** This refers to how many predictions are served in a given time and is a key deployment metric. As throughput changes, this might influence various aspects of the serving system, including the data distributions.
- **Error Rates.** For example, the number of times the model predicted an individual would purchase an item, but was incorrect, and vice versa.

In the remainder of this section, we will discuss the above aspects of continuous monitoring in greater detail, and see how these tie in with the other aspects of MLOps that we have already discussed in previous sections.

Additional Resources

Data Distribution Shifts and Monitoring - Stanford CS 329S