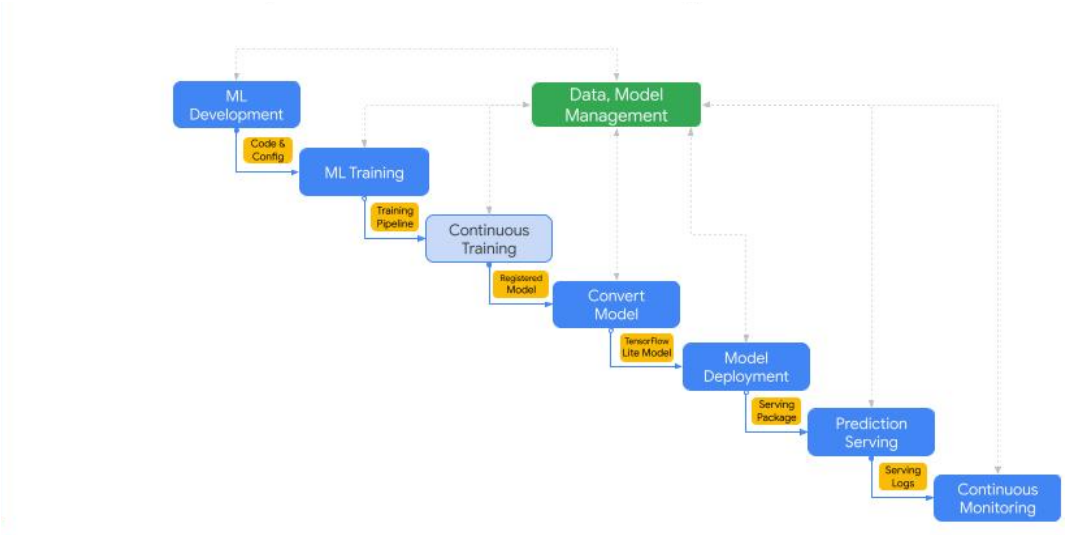
Overview of Continuous Training



Objective

In most respects, our world is in a constant state of flux. These constant changes transfer over to our data, which we can consider a simple “snapshot” of reality that our machine learning algorithms use to build models that describe the physical world. This presents us with a problem: if the world, and hence our data, is always changing, won’t our models always be wrong? In some respects, the answer to our question is yes. To that end, we will first start with a broad overview of Continuous Training and then dive into the specifics of each of these:

- Data Processing Engine
    - Data ingestion
    - Data validation
    - Data transformation
    - Keyword spotting datasets
- Model Training Engine
    - AutoML
    - Neural Architecture Search
    - Hyperparameter tuning
    - Why these are important and how they can be helpful
- Evaluation
    - Use-case driven metrics
    - Continuous Training pipeline metrics

We learn these concepts through the lens of Keyword Spotting as a TinyML use case.

Motivation

Our models will likely never perfectly reflect reality - just think, when was the last time you trained a model that was 100% accurate? However, for most applications, our models do not need to be perfect, just consistently accurate. Accurate implies that our model performs well for a given task; for example, we might decide that a malware detection algorithm with 90% accuracy is sufficiently accurate to implement as part of an intrusion detection system. Consistent implies that our model performance does not change significantly over time. In the case of malware detection, it is clear that malware changes quite rapidly in response to different software vulnerabilities that are discovered (see the CVE List for examples). If our algorithm's accuracy falls to 30% after three months of being in use, this is not great, and you will probably end up with all manner of software viruses unless you make sure to periodically update the model using more recent data. So far, we have motivated the need for periodically updating our models, which we call retraining. While this will help to resolve our consistency problem, it presents us with several more questions. How often should I retrain my model? How can I tell if my model is underperforming? How can I translate this into monetary costs?

Continuous Training

The answers to these questions are highly context-dependent. For example, a model trying to model customer preferences will change significantly over time in response to various stimuli, including socioeconomic and political factors. These changes may be chronic, occurring over a long time frame such as years or decades, or immediate, occurring perhaps on the order of just minutes or hours. Clearly, these situations would have to be approached radically differently. In contrast, modeling defects within a manufacturing process might only change if the process is altered. Hence, retraining this model over time would largely be unnecessary (other than to improve the model's accuracy by training on a larger set of data). Therefore, the frequency of retraining is largely data-dependent. In the case of modeling consumer preferences, a more advanced form of retraining might be implemented, called continuous training.

Continuous training, as the name suggests, refers to the procedure of retraining our model on a continuous basis. This might be achieved in different ways. Training can be scheduled, as in the case of periodic retraining, using a cron-like scheduler. This might tell a system to retrain a model using the most recent data once every night, similar to the nightly build of TensorFlow. Alternatively, training might be event-driven, by what we call retraining triggers. These triggers might be a metric that is monitored, and once the model has fallen below a specific threshold, the model is retrained, or when a new threshold amount of data has been collected. The metrics or thresholds selected for this purpose are, again, context-dependent. For example, a binary classification algorithm might care more about monitoring the F1 score or true positive rate as opposed to raw accuracy. Training might also be manually triggered, such as in the case of a new system update.

Retraining models is very important for some applications, and less so for others. Taking COVID-19 as an example, a large number of commercial models in the finance, e-commerce, and business sectors failed during this period due to rigid models that were unable to accommodate rapid and profound changes in socioeconomic conditions. Machine learning models are great when they work, but can cause a lot of headaches when their assumptions are violated! In this section, we will delve deeper into the concept of continuous training and how it can be added to our arsenal of MLOps tools.