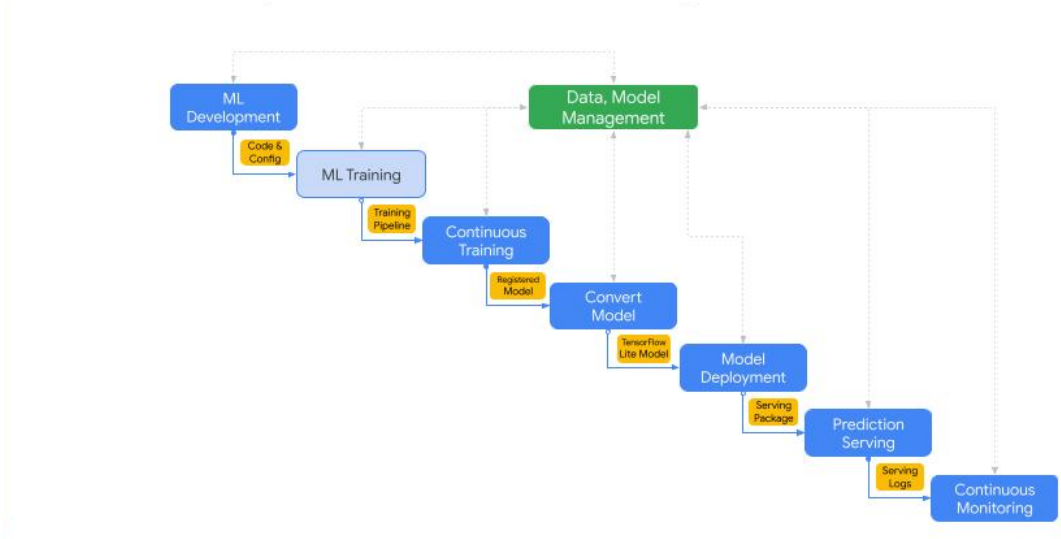
Overview of Training Operationalization



Objective

ML Training operationalization sounds like an intimidating term at first, but it is essentially the process of automating and improving upon aspects of the standard ML workflow that we have come to know and love. The procedure is shown above. The topics we will cover include:

- What is ML training operationalization and why is it important?
- Continuous integration
    - What is continuous integration?
    - What are the sources and various roles that CI plays in ML?
    - What are some challenges for TinyML with CI?
    - Why is testing TinyML challenging compared to traditional ML systems?
- Continuous delivery
    - What is continuous delivery?
    - What are the various stages of continuous delivery?
    - What is staging and why is it helpful?
    - Who does acceptance testing?
- Production deployment
    - What is it about?
    - What are smoke tests?
    - What are commonly used ML deployment strategies for releases?

Motivation

Take, for example, a medium-sized company with around a hundred employees. At this organizational level, we may have dozens or even hundreds of models being developed. All of these models will be at different stages; some in development, some ready to be tested, and some already deployed. These models may have been developed by different individuals, perhaps using different packages and software versions. As you might suspect, from all of this heterogeneity in the model development process, problems are likely to ensue. These problems may just be inefficiencies, with individuals spending time poring over an unfamiliar PyTorch implementation while everyone else in the company is using TensorFlow. However, sometimes these can be more severe problems, giving rise to errors in production that might have been missed during the testing stage due to an inconsistent testing regime, or perhaps a model that has not been tested yet being put into deployment, or even losing a model because it was not stored somewhere central for the rest of the team to access!

Training Operationalization

How can we tackle these kinds of problems in an efficient way? Training operationalization! The idea of training operationalization is that the training and deployment processes are streamlined and automated in such a way that consistent testing and monitoring procedures are applied to each and every model. For example, a model registry can be created (e.g., using MLflow) to store model versions and relevant artifacts (e.g., model parameters, dependencies) that can be compared later. A model registry is very similar to a GitHub or DockerHub repository for machine learning models. Continuous integration/delivery capabilities (e.g., Jenkins, GitLab, Ansible) can be introduced in order to dynamically update models and pass them through automated testing procedures (e.g., unit tests, acceptance tests, smoke tests) to ensure all of their functionalities remain intact after they have been edited. Not only can these be tested for a single system, but they can be tested across a range of hardware platforms, operating systems, and package versions. This is very similar to how DevOps is used to streamline software development, but reimagined for streamlining the training, development, and deployment of machine learning models.

The specific procedure for ML training operationalization that an organization might take is highly variable. Smaller companies may only have a single testing stage, whereas a larger organization may have multiple stages and a procedure more complex than that outlined above. Over the remainder of this chapter, we will become more familiar with training operationalization and how it can help us to mitigate errors and improve the efficiency of our ML workflow.