**Learn the Core of MLOps — Building Machine Learning (ML) Pipelines**

**A practical guide to implementing MLOps to make AI work**

**Why is MLOps Necessary**

It is quite well established that, in order to make AI work, the real challenge is not building a Machine Learning (ML) model, the challenge is building an integrated ML system, and to continuously operate it in production. This is why the concept of MLOps has been developed, and it is rapidly gaining momentum among Data Scientists, ML Engineers and AI enthusiasts.

Due to the robust research in ML algorithms in recent years, AI and ML have shown great potential to organizations in terms of creating new business opportunities, providing better customer experiences, improving operational efficiencies, and so on. However there is a huge technical gap between training an ML model in a Jupyter Notebook and deploying an ML model into a production system. As a consequence, many companies haven’t figured out how to achieve their ML/AI goals and they turn to MLOps for help, hoping that, through MLOps, they can make AI work in a real-world production environment to truly reap the benefits of AI and ML-driven solutions

Therefore, I believe it would be quite useful to develop a series of practical guides explaining to how to implement MLOps practices. These guides will include explanations of the key MLOps components, design considerations as well as sample codes for implementation.

If we view MLOps purely from an engineering and implementation perspective, there are three core parts for any end-to-end MLOps solution:

* The first part is — ***Data and Feature Engineering Pipeline***
* The second part is — ***ML Model Training and Re-training Pipeline***
* The third part is — ***ML Model Inference and Serving Pipeline***

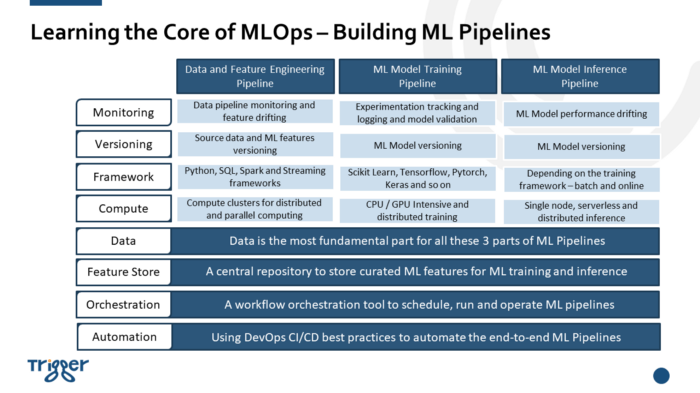
MLOps stiches together the above 3 pipelines in an ***automated*** manner and makes sure the ML solution is ***reliable***, ***testable*** and ***reproducible***. In the remaining part of this blog, I will explain these 3 pipelines, piece by piece.



Photo by [Hitesh Choudhary](https://unsplash.com/@hiteshchoudhary?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

**Key Building Blocks of Implementing a MLOps Solutions**

The image below, captures all the key components of the above mentioned 3 pipelines of MLOps. As you can see, it can be quite complex to build an end-to-end MLOps solution, but don’t worry, I will explain them one by one in detail, and demo how to implement each component, in the upcoming series of posts.



Learning the core of MLOps — Building ML Pipelines (Image by Author)

**Data and Feature Engineering Pipelines**

I’ll start with data and feature engineering pipelines, as data is the core of any ML system. Generally speaking, data pipelines refer to Extract, Transform and Load (ETL) pipelines, through which data engineers ingest raw data from source systems, then clean and transform the data into reliable and high-quality information for downstream data consumers. If you are interested in understanding how to build data pipelines, I have a seperate article on it. [Learn the Core of Data Engineering — Building Data Pipelines](https://medium.com/@weiyunna91/learn-the-core-of-data-engineering-building-data-pipelines-21a4be265cc0?source=friends_link&sk=a15ca2e70b29b46a33adc695a341349e). Please feel free to read.

What is unique about ML is that ***raw data*** needs to be converted into **features** so that ML models can effectively learn useful patterns from the data. The process of converting raw data into features is called ***feature engineering.*** Therefore the focus of this post will be around implementing feature engineering pipelines as well as providing an introduction of what a feature store is.

There are various feature engineering techniques, including imputation, handling outliers, binning, log transforming, one-hot encoding and so on . If you want to learn more about them, I am sure you can google and there will be many blogs on feature stores. However, what I want to emphasize here, is that generally for a Machine Learning (ML) project, data scientists put a significant amount of time and effort into feature engineering in order to get decent performance on their machine learning models. Therefore, it is valuable and necessary to store these features for discovery and reuse.Hence the concept of “feature stores” has been developed and there are quite a few — both open-source and commercial offerings — on feature stores. However, feature stores are more than for feature reuses. ***A feature store is data management layer for machine learning that allows you to share & discover features and create more effective machine learning pipelines.*** Feature stores can be leveraged in the two most important pieces of any MLOps solution — model training and model serving. In summary, feature store provides the following functions and benefits:

* Feature discovery and reuse
* Consistent feature engineering for model training and serving
* Monitoring for data and feature drifting
* Reproducibility for training datasets

For a complete MLOps solution, it is a must to set up feature engineering pipelines and feature stores for both model training and serving. This is a very high level introduction. I will publish a blog specifically on feature stores very soon. Please feel free to follow me if you’d like to receive a notification when new logs are published.

**ML Model Training and Re-training Pipelines**

Once the feature engineering is complete, the next step will be ML model training. ML model training is a highly iterative process, and this is why it is also called ***ML model experimentation***. Data scientists have to run many experiments with different parameters and hyperparameters in order to find the model with the best performance. Therefore, data scientists need a systematic way to log and track the hyperparameters and metrics of every experiment run, so that they can compare each run, and find the best one. There are some open-source libraries that can help data scientists log and track model experimentation, such as [mlflow](https://mlflow.org/). mlflow is an open source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry.

Other than model training, data scientists also need to ***evaluate and test*** the model before they are are comfortable with putting the model in a real production environment. They need to make sure the model will perform well with real world live data — as well as it does with training data. Therefore selecting the right test data set and selecting the most relevant performance metrics is quite important. With MLOps in place, both model training and model evaluation need to be automated.

The key challenge of an ML driven system is that the model performance is determined by the data used to train it. However, ***data always changes.*** Therefore re-training the model becomes absolutely necessary for most ML driven scenarios. Generally, there are several ways to trigger model re-training:

* ***Schedule-based*** — The model is re-trained with the latest data at a pre-defined interval, for example, once a week. Depending on how quickly the data changes and additional business requirements, the schedule frequency can vary significantly.
* ***Trigger-based*** — Model is triggered to be retrained when there is drifting identified, such as data drifting, feature drifting or model performance deterioration. In order to achieve a totally automated way to retrain the model, there needs to be a robust monitoring solution to monitor both data changes and model changes.

The ideal outcome of a MLOps solution is to not only automatically re-train the model, but also evaluate the newly trained model and pick the best run, based on the pre-defined model metrics.

**ML Model Inference/Serving Pipeline**

Once the model is trained and evaluated, the next step will be to put the model into a real production environment for use. Generally, there are 2 ways to serve the trained ML model:

* ***Offline batch inference*** — The ML model is called, for predictions at a certain frequency. The frequency can be as high as daily, weekly, or even longer, but it can also be as low as every minute. When the frequency is very low, (very often), you can integrate streaming data pipelines with ML model inferences. When the data volume for batch inferences is very large, a distributed computation framework will be needed. For example, you can load the model as a Spark User Defined Function (UDF) and apply it at scale using distributed computing for parallel inferences.
* ***Online real-time inference*** — The ML model is packaged as a REST API endpoint. For online real-time inferences, the packaged ML model is generally embedded into an application, where model prediction is generated upon a request. When the request volume is large, the model can be packaged to a container image and deployed into a Kubernetes environment for auto scaling, to respond to ***large volume of prediction requests.***

As explained a little earlier, data determines the model performance. In the real world, data always changes and inevitably the model performance will also change, often deteriorating. Therefore ***having a monitoring solution to track the changes of the production models as well as the data/features used to feed them, is very necessary.*** Once significant changes are identified, the monitoring solution needs to be able to either trigger model retraining or send notifications to the relevant teams so that fixes can be done immediately. This is particularly necessary for business critical ML driven applications. Generally ML model monitoring includes the following 4 categories:

* Prediction Drifting
* Data / Feature Drifting
* Target Drifting
* Data Quality

There are both open source and commercial offerings on monitoring ML solutions. For example, [Evidently AI](https://www.evidentlyai.com/) is an open-source Python library to evaluate, test and monitor ML model performance from validation to production.

So far we have covered the 3 key pipelines of a complete MLOps solution — data and feature engineering pipeline, ML model training and re-training pipeline and ML model inference pipelines, as well as ML model monitoring.

**The Scale Spectrum of an MLOps Solution**

MLOps is a quite new concept and you probably can tell from the above introduction that MLOps involves piecing together quite a few different components in order to make AI work in the real world. Therefore, many people think MLOps is daunting and complex. However there is a **scale spectrum** that needs to be drawn, when we talk about implementing an end-to-end MLOps solution.

For example, if your ML solution is small scale and batch-based — batch data pipeline, batch training, batch inference and the data volume does not require distributed computing — then implementing MLOps is not that difficult, and even a data scientist can become “full-stack” and own the entire solution. However, if you are talking about large scale, continual training, and real-time inference, it could be quite complex, requiring multiple teams and different sets of infrastructure to work together.

Therefore, in the upcoming posts, I will explain some ML reference architectures and the best practices for implementing MLOps at different scales, as each scale could involve very different skill sets and different infrastructure setups. Stay tuned.