MLOps: Monitoring phase

Mitali *Agrawal*

A path to build reliable, robust, and consistent ML models

Introduction

We published five blogs of a six-part series about the importance and necessity of having a solid MLOps implementation where we looked at the different phases of MLOps namely, Problem Formulation, DataOps, Modeling, and CI/CD. In this final blog post we aim to elaborate on a quintessential MrealxLOps phase that is the crown of the entire process — Monitoring.

We illustrate the importance of Monitoring and introduce the players involved in this part. We then discuss the main activities and expand on some design principles to keep in mind while setting up this part of the MLOps process. Finally, we look at the principles that govern the monitoring to understand this phase in a broader context.

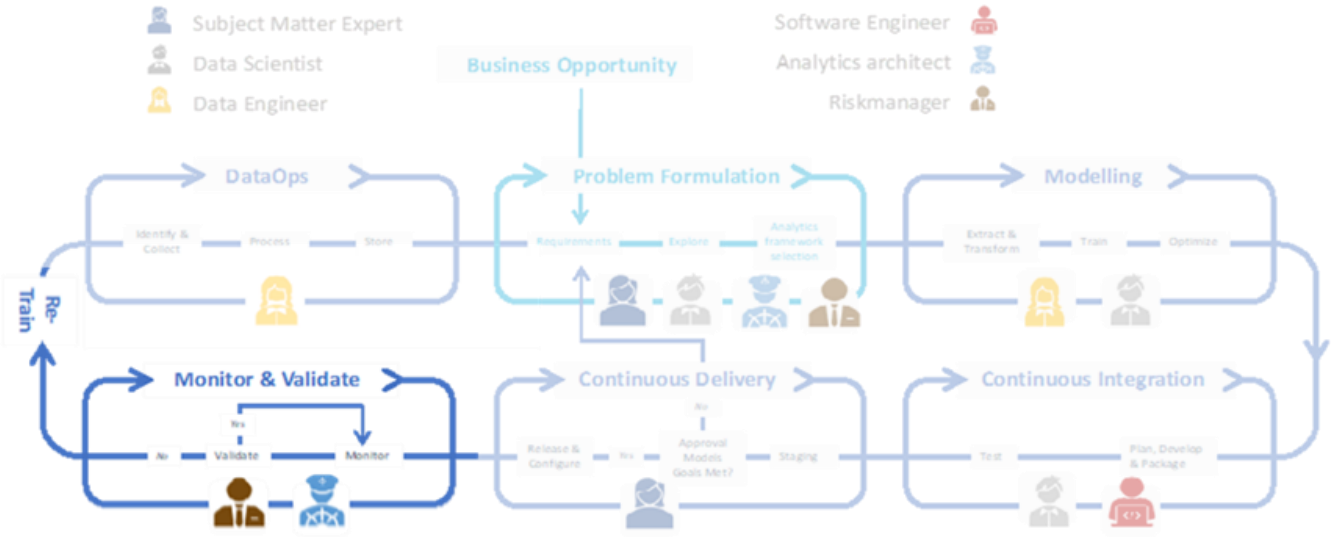

***Figure 1**: High level overview MLOps — phases and the involved players with an emphasis on Monitoring.*

The why and the who of monitoring

Why did we call this phase the crown of the entire ML pipeline? Because this is where we get to see and analyze the outputs of our ML model. Monitoring is where the seeds of improvement lie for the experimental nature of ML algorithms. But ML pipelines come with their own technical debts, i.e., long-term costs incurred by moving quickly in development.

From the ML side, while training and deploying ML models is relatively cheap and can follow fast release cycles, the maintenance is difficult and expensive. Like any other software pipeline, they need a health check on interacting services, cloud deployments, system performance, network traffic, latency, and so on, which are not discussed in this blog. In this blog, we want to focus on the ML debts, i.e., model and data, and additional hidden technical debt ML has in terms of unpredictable behavior in production over time.

Data-dependency is a critical aspect where the ML pipeline differs from traditional software. The data is continuously subject to change, and business requirements also constantly shift. This instability in input data can lead to model decay which is the hidden technical debt of ML, the risk of **concept** and **data drift**. AI models are only as good as their ability to simulate real-world phenomena to predict an output. As AI is increasingly being used in decision processes, it is unacceptable for the model to drift thereby giving less relevant predictions.

Entanglement is another technical debt. The ML models use multiple input signals, entangling them to calculate meaningful features. So, if the distribution of any input signal changes, its feature weights will change, which can change the weights of all other remaining features. This scenario is also called CACE: changing anything changes everything issue. Hence even some silent 'improvements' or small hardware 'calibrations' can lead to sudden ramifications for the model. Therefore, it is important to make sure we know what to monitor and how to mitigate the technical debt of ML.

The **key players** in this phase are the *data analysts*, *data analytics architects*, and *risk managers*. The data-analytics architect advises the right technologies to analyze and visualize the data from the wide range available on the modern market, from commercial to open source. Furthermore, the analytics architect evaluates non-functional attributes, such as security, utility, and stability. Thus, within MLOps, the analytics architect demands an overview of the models and their related data resources. Since the data can be multi-dimensional, data analysts often manipulate and aggregate the data before performing fundamental analysis or monitoring. The risk manager's task is to assess — from beginning to end of the MLOps process — whether the ML model carries a risk for the organization from a regulatory or privacy aspect.

Although data analysts have the technical skills to manipulate and visualize the data, they often lack a deeper understanding of their business domain. For this reason, *Data Scientists* and *SMEs* are needed in this phase as well who provide the team with the proper context and data understanding. During the monitoring process, they closely observe whether the ML model's outputs make sense. The SME must grasp the model evolution from a business perspective. Furthermore, the SME has a sharp eye to identify ML technical debts and adjust the team's course where needed.

This way, SMEs have a mechanism for providing feedback to the data scientists to improve the model performance. Thus, monitoring is required to observe and understand the ML model results and the data. It ensures consistency and robustness of the ML system and instills trust in the user.

Activities in monitoring

How we store the data impacts how we query it, even for analysis and monitoring. Our previous blogs from this series also illustrate fundamental principles that ensure smooth data handling at this last stage of the MLOps pipeline. The activities in the monitoring phase are described below:

· **Creating dashboards:** Model analysis often involves high-dimensional visualization and use-case-specific data-slicing. A dashboard consists of the visualizations of all the essential metrics that need constant monitoring. It provides vital insights for detecting changes in model behavior over time, influencing business decisions and model improvements.

· **Analytics metrics:** Metrics like data stability (i.e., distribution over time), model performance, and operations metrics. These three categories encompass all the relevant parts of an MLOps setup; hence, appropriate thresholds are defined for a healthy ML system.

· **Creating alerts:** For active monitoring, a mechanism must exist to raise the alarm when a failure happens. A failure occurs whenever an observation value reaches the defined threshold. An alert is usually an automated message sent to the task owner whenever a failure occurs.

· **Feedback loops:** To put it simply, you place the model in production, measure all relevant metrics, and use them to improve the model iteratively in a cycle. Providing feedback is the primary motivation for monitoring — improvement over time.

· **Creating training triggers:** Automatic retraining of the model. How do we trigger the retraining? If the performance changes are quantified, and specific thresholds are applied based on the use case, triggers can be created to re-train and fine-tune the model hyperparameters using the old data as well as the new data collected since the model was put to production.

· **Data-logging:** Logging and documentation for all software development come under good coding practices, which can be a lifesaver in debugging. Furthermore, it can provide valuable information to fine-tune ML algorithms better. Alerts can also be created by automated parsing of data log files.

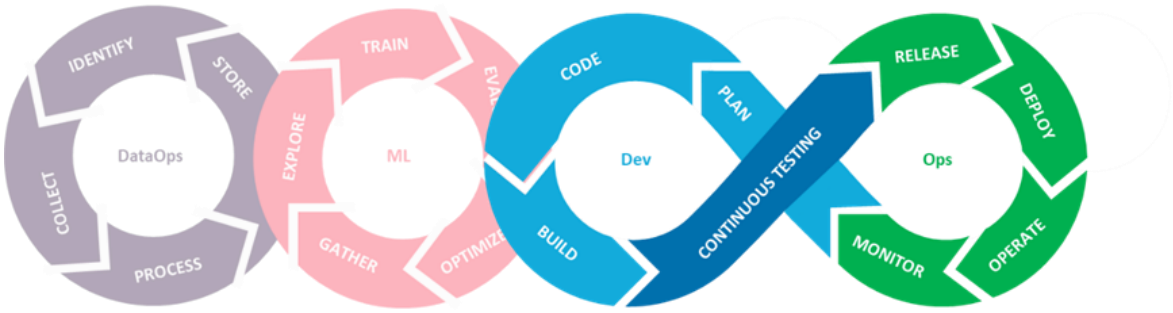

***Figure 2**: An overview of DevOps activities (including monitoring) and illustrated interaction with other MLOps players.*

To implement the above steps for the efficient monitoring of ML models, some reliable enterprise solutions already exist: Azure, AWS, and Google Vertex AI monitoring services. These platforms provide active monitoring to detect feature and data skews/drifts, creating dashboards and alerts etc. Although these pipelines can also be set up by data-analysts using Microsoft Power BI, Neptune, or Tableau, and guidance from data scientists to include the domain-specific visualizations can be a bonus.

While implementing these functional monitoring steps, it is important to recall the principles which govern the design principles of this phase: reliability, robustness, and consistency.

Reliability principle

Reliability focuses on assessing and governing how trustworthy and fail-safe the ML system is, rather than how well it performs "on paper". Under the monitoring phase's reliability principle, we must ensure that the models placed in production don't produce any technical debt such as availability, system performance, security, data drift, latency, etc. Since the reliability of ML systems depends on data and code, both aspects must be thoroughly scrutinized in monitoring and debugging. Tracking system behavior and model performance in real-time, with automated alerts for any threshold violation, ensures long-term ML system reliability. Apart from this, we can choose to continuously monitor and validate the extracted features to create valuable logs for analysis and improvements. All the up-stream producers of the data (used for data filtering and manipulation) should be monitored after every pre-processing step for added reliability.

Since the ML models can negatively impact the downstream processes, it can be wise to continuously assess the impact of the model staleness to ensure reliability. Models should be tweaked and retrained over time to address the changing data and business needs. This can lead to correction cascades resulting in improvement deadlocks. One way of achieving reliability is to avoid correction cascades by doing small A/B testing with older model versions on current data. Older model versions are sometimes more generalized and can perform better on the changing data. Re-visiting ML model versions for fresh improvements can help break the improvement deadlock of models. Another way to ensure the reliability of the whole ML system is to have rollback procedure ready, especially when deployment dependencies are high, to deploy-back the older version of model in case of failures.

It is always nice to have a rubric handy for ensuring reliable development of ML systems in production environment, which is exactly what a team from Google documented and can be found here.

Robustness

This principle is tightly connected to the previous one, but it is also essential to look at the overall model performance over time and on unseen data. A robust ML algorithm will have a low testing error, i.e., low bias, even under unforeseen system changes. Following the continuous testing phase described in our previous blog ensures robustness through a test strategy. The test results and final tested model versions create ground truth subsets of data to validate against. The ground truth labels are then used to perform targeted analysis to ensure robust ML model performance.

One way is to analyze the prediction bias. In a healthy ML system, predicted labels' distribution is equal to observed labels' distribution. In case of multiple input data sources, data entanglement leads to hidden feature dependencies across such multi-dimensional data. Continuous visualization across data slices, like different data sources, across different feature space, often provides valuable insights in creating robust models. Another way to increase model robustness is by monitoring that the model performance is similar for all these data sources. Testing ML models with considerations of inclusion (in training data) can also positively impact the robustness of the ML system.

Consistency

A consistent pipeline will perform as expected over time. We need to re-train ML models because of their data-dependent nature, to minimize the risk of concept and data drift. The statistical distribution of data and corresponding features can change with time, and COVID is a perfect example, where all prediction models were affected drastically. The data shift may be prominently visible in the monitoring of accuracy and performance metrics of the model or can be hidden in the underlying shift of configuration assumptions. Thus, statistical tools like SAS is widely used to check and analyze data and detect data-drift using techniques like Kolmogorov-Smirnov and chi-squared tests.

One common way to deal with data dependencies is to create the versioned copy of input data over different periods and freeze the feature mappings until updated mappings are made available per the changing data. Versioning is, however, prone to model staleness and requires frequent re-trained model deployments, thus making it a costly solution. RStudio is another widely used open-source tool, which helps data scientists and analysts to work together, integrate dashboards and analyze hidden trends in the input signal.

Conclusion

In this blog, we complete the ML pipeline cycle and focus on a crucial phase of MLOps — Monitoring. Monitoring is required to ensure the reliability, robustness, and consistency of ML models and to analyze the model performance and data to improve the system. In this blog, we saw the importance of monitoring and how people with different expertise again come together during this last phase to enable the operations and analytics in the ML pipeline. We also saw the list of essential activities involved in monitoring and the principles that play a vital role in designing these activities. This blog also concludes the six-blog series, which highlights all the critical phases in the MLOps cycle. Ultimately, every step in the MLOps process builds the way for more trusted, transparent and performant AI models at scale.

But does the decision-making end here? We wish it were that easy! There is one more hidden aspect in ML systems — which differs entirely from traditional software development. That is the different maturity levels in which an ML project can be. Although our next post does not directly dictate the MLOps principles as this blog series does, it focuses on different ML development environments in different development stages. So, stay tuned for some fun relatable examples to understand how MLOps infrastructure and design choices vary with different maturity levels an ML system can have!