MLOps for PHM Systems

Mikael Yemane

Raytheon Technologies - Collins Aerospace, Windsor Locks, CT, 06096, USA
mikael.yemane@collins.com

ABSTRACT

Advances in machine learning (ML) techniques allow practitioners to generate substantial predictive value from historical data. Modern sensors generate vast amounts of data which inform prognostic health management (PHM) systems. As ML techniques continue to grow in importance for PHM, the system that manages and deploys ML models becomes critical for successful production software. Machine Learning Operations (MLOps) is centered around implementing continuous integration and deployment (CI/CD) practices in the context of ML applications. We will present MLOps designs for deploying machine learning based PHM software and discuss ML pipelines that automate data ingestion, model training, testing, deployment, and monitoring. The principles we will examine ensure model quality, performance, and software stability. We will call attention to important design considerations and demonstrate solutions for the full model lifecycle when building MLOps pipelines for PHM systems.

1. INTRODUCTION

Machine learning operations (MLOps) paradigms are revolutionizing data driven industries, allowing for more reliable and scalable ML software. When Designing ML systems, it is important to spend time considering model objectives and how to design a system that will help achieve such goals. In this paper we will discuss designing state of the art MLOps systems and its applications in prognostic health management (PHM). We will emphasize design considerations when processing raw sensor data and building machine learning based predictive systems.

2. FROM DEVOPS TO MLOPS

There are many goals when building MLOps systems. Model performance, stability, and reliability are often top priority. To discuss MLOps we must start with the ideas borrowed from Development Operations (DevOps). Many of the concepts and considerations in DevOps carryover to MLOps. We will summarize important DevOps concepts then discuss the specific ML tasks we must consider for MLOps. MLOps design principles use many of the core principals of DevOps to deliver reliable software. In a typical DevOps workflow, we design automated pipelines that initiate the building, testing, and deployment of software source code. Mojtaba Shahin et al (2017) provided a thorough review of approaches for continuous integration, delivery, and deployment. The expectations for MLOps systems are similar, ML models go through a process of training, testing, and deployment. Georgios Symeonidis et al (2022) published an overview of common tools used and two popular definitions of MLOps maturity levels. Although MLOps can encompass different standards and tools the goal is to automate as much of the process as possible. Both MLOps and DevOps strive for continuous testing and deployment to support software stability and rapid development. There is a lot of overlap between DevOps and MLOps. However, the specifics within pipeline steps and software requirements are quite different. In the next few paragraphs, we will discuss the two core elements of DevOps Continuous integration (CI) and Continuous Deployment (CD) how they are applied in the context of MLOps. We will also consider components that are exclusive to MLOps system designs.

2.1. Continuous Integration

Continuous integration (CI) is focused on pipelines that build and test software throughout development lifecycle. Continuously testing the source code as new features are added leads to cohesive software that can be deployed rapidly. The software will be built and automatically tested with every new push to release branches. If all tests pass, then the overall build will be successful, and the software version can proceed to the next step in the process. Dev teams must have a common code repository where all the app features and tests reside. When new code is committed to the repo it is important to set requirements such that all unit and integration test must pass before merging with the main branch. This will help protect the production code. Code should be committed often, and builds should be run regularly to ensure the app functions as expected, this allows for quick feedback from the automated software tests and dependent teams. These concepts directly apply to MLOps systems. When new features are added, hyperparameters are updated or any of the code related to the model has changed developers to set automated tests that validate the model and associated code. For PHM systems model assertions should also include schema expectations and task specific assertions. Assertions based on expected database schemas and data quality will mitigate and help isolate data processing issues. This is particularly important when processing raw signal data. The specific tests will for an individual system will vary widely depending on the use case. In all cases, proper testing of our models and data prior to release will allow for more reliable models and predictions.

2.2. Continuous Delivery/Deployment

Continuous delivery and Continuous deployment (CD) are the two approaches for deploying software. In both approaches the goal is to regularly ensure that the software is ready for deployment and can served at any time. The main difference is that continuous delivery entails a manual process of promoting software to production while continuous deployment is a fully automated process. Depending on the use case there may be a preference for one over the other. However, we will focus on continuous deployment as most practitioners are striving for as much automation in the software development process as possible. We suggest striving for automictic deployment of updated source code once workflow and testing pipelines are complete. CD principals apply directly to MLOps. The differences for ML systems will be further discussed in the model deployment section.

2.3. Data Validation

Data quality is one of the most important factors we can monitor and control in ML systems. Robust data pipelines should be a high priority in producing reliable MLOps pipelines. The most common method for ensuring data quality is having specific schema requirements and tests that enforce expected schemas and data properties. Eric Breck et al (2019) focused on the topic of ensuring data quality. When ingesting raw sensor data, it is particularly important to specify constraints and expectations for the input data. This will help ensure that invalid signals do not reach the model. It is also suggested that all functions involved in the modeling process have unit test coverage. Often sample data sets or simulated data is used to test relevant functions and assertions are made for the expected results. It is common to aim for 70-100% code coverage. Assertions can also be placed on code coverage prior to release.

3. MLOps System Design

When designing MLOps systems there are many important design considerations. Making the right decisions early in the design process will help ensure stable deployment of ML software. Fig 1. an example MLOps design for pushing models to production. In this section, each of the major components will be discussed in detail. We will emphasize requirements that minimize bugs in ML systems and allow for confident deployments to production.

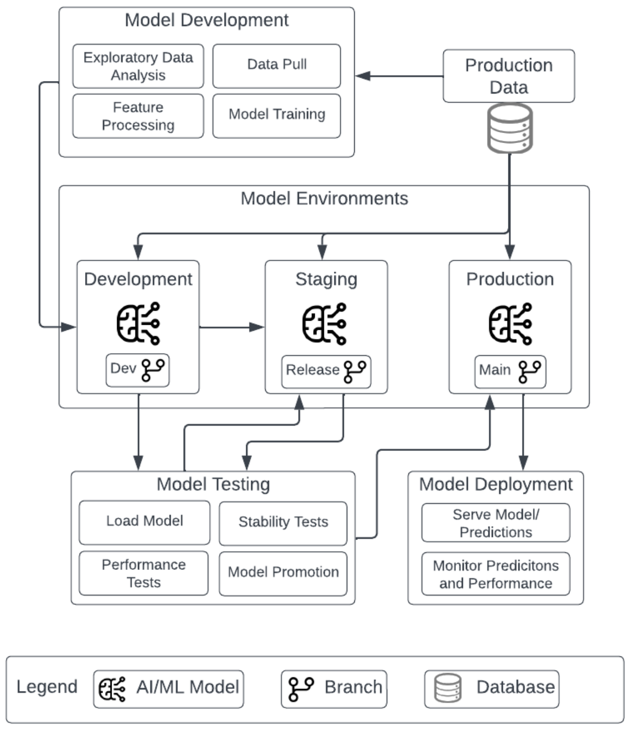

Figure1. Example MLOps Design

3.1. Model Development

In the model development stage, there is a lot of flexibility. Many different modeling approaches can be explored. At the beginning of this stage, we need to define the initial data pull. If you are dealing with raw signals, there will need to be refinement and initial standardization of the data. Once the initial schema is defined, we can begin exploring the data. When developing a new model for your prediction task it is important to conduct exploratory data analysis (EDA). Visualizing your data and generating summary statistics often helps uncover unique characteristics of the data. These characteristics can help inform the feature processing and modeling process. Before developing new models, it is important to at least have three separate sets of the data. Prior to generating features for the data, we should have separate data sets for training, testing, and validation. Feature generation should be done after splitting the data. This helps ensure that information form the test and validation sets aren't making their way into the training process. Feature processing and encoding should be conducted based on the characteristics of the data and goals of the modeling process.

Once feature generation is completed on the training set, we can move to training new models. We then use the test and validation data to estimate model performance. Various model types and hyperparameters can be evaluated to decide on a preferred model. If the model has sufficient performance metrics on the test and validation data, we can pass the model through our unit and integration tests. If the model passes all the tests, we can then mode the model into the staging environment in preparation for production deployment.

3.2. Model Environments

The three common stages for the model are development, staging and production. Most software can be deployed reliably in that set up. However, there are variations with additional environments. There are different branching strategies we can consider. Rakshith Subramanya et al (2022) discussed common branching strategies and the process for deploying code. For the model environments shown in Fig 1. we will consider development, release, and main branches. Where the development branch allows for model experimentation data modifications, and new features with minimal constraints. The release branch should contain planned features for release and is expected to be more stable and pass all automated tests. The release branch should be tested regularly and maintained to be ready for production deployment. The Main branch holds the code that is deployed live in production. It should be built and deployed regularly in the production environment serving predictions to customers or stakeholders.

In the development model environment, we can make various modifications to the code and test out hyperparameters without concern of harming production. This model environment should be where model code in development branches are tested, model performance is measured, and the model is considered for progression to the staging environment. In the staging environment we will load the model and the model code tied to the release branch to thoroughly test performance and stability. Unit tests should also be conducted for code relevant to the model. Any user interface or API endpoint tied to the ML software should also be thoroughly tested at this stage. After the model has completed testing in both the development and staging environment, we can promote the model to production. In the production environment we can serve predictions and monitor live performance. In our example we discussed the deployment of both the model and associated code. It is important to note that there are deployment strategies that focus on deploying one or the other. It is not always required to pass both the model and associated code through all model environments. In many cases it is sufficient to only pass the model object or model code.

3.3. Model Testing

Model testing is dependent on the model use case but generally we want to consider model performance, stability, and data validation. Performance is often a high priority so we should set a threshold for model performance as the object is passed through the model environments. Performance metrics vary depending on the task. In the context of classification metrics such as accuracy, precision, recall, F1, and F2 scores are potential measurements. For regression tasks mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), mean absolute percent and error (MAPE) are commonly used. There are many other regression and classification metrics to also consider. Naeem Seliya et al (2009) published a study on classifier performance metrics that cover many options. Alexei Botchkarev (2019) has published a similar analysis looking at performance metrics in the context of regression algorithms. The data validation steps discussed earlier should also be considered. Schama assertions and data quality tests should also be included in automated tests. There are many tools used to automate tests.

Some programs may require prediction stability tests to ensure that there aren't large variances in predictions while others should allow to predictions. We an also compare the performance of the production model to performance of the model in staging. Performance thresholds for the promotion of a new model can also be set here. As we design our pipelines we want to automate as much of the process as possible with the goal of crating fully automated MLOps pipelines. Model reproducibility is also something we want to keep in mind when designing the product. This will help with debugging and reverting model versions if there are unforeseen production issues with newer models. Odd Gundersen et al (2022) published a review of current ML platforms and conducted an analysis of reproducibility for each of the platforms discussed.

3.4. Model Deployment

Model deployment is the last stage in an ML model lifecycle. After the model and associated code have gone through testing in the development and staging environments the model should be ready for production deployment. There are many approaches for serving predictions two common methods are serving an API endpoint that generates predictions and creating databases that store predictions that the app references. The approach will vary depending on broader software requirements. There are many cloud systems and tools to choose from. The most widely used cloud providers are Amazon Web Services, Azure, and Google Cloud. We recommend designing a system to be cloud agnostic. It is common for large scale systems to interact with multiple different cloud services. Nicolas Ferry et al (2013) and Laura Savu (2011) provide detailed information on cloud deployment models and security considerations.

We can streamline deployment by scheduling pipelines that automatically promote source code from the main branch and reference models in the production model environment. We can also automate training and testing of resulting models. We can promote the model and relevant code through the model environments given performance is above specified thresholds and all data validation and tests are successful at each stage. Once a model is released to production it is important to monitor live performance. Live monitoring helps identify production issues. Estimating model drift and setting alerts or automated training jobs in response to drift can help avoid performance issues in production.

4. Conclusion

In this paper we discussed methods for building MLOps systems. We reviewed the relationship between DevOps and MLOps, the areas of overlap such as the expectations for continuous integration, continuous delivery, and continuous deployment. We also discussed the specifics to be considered for MLOps systems including data validation, model development, model testing, model environments, and model deployment. MLOps designs will vary based on the objective and use case for each individual team. When building ML based PHM systems, data validation and model testing will be paramount. This will minimize software downtime and help mitigate production issues. MLOps system deigns should aim for as much automation as possible to improve the ML development process.

References

Mojtaba Shahin, Muhammad Babar, and Liming Zhu (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access* vol. 5, pp. 3909-3943. doi:10.1109/ACCESS.2017.2685629

Georgios Symeonidis, Evangelos Nerantzis, Apostolos Kazakis, and George Papakostas (2022). MLOps - Definitions, Tools, and Challenges. *IEEE 12th Annual Computing and Communication Workshop and Conference.* Las Vegas, NV, USA doi:10.1109/CCWC54503.2022.9720902

Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich (2019). Data Validation for Machine Learning. *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA

Rakshith Subramanya, Seppo Sierla, and Valeriy Vyatkin (2022). From DevOps to MLOps: Overview and Application to Electricity Market Forecasting. *Applied Sciences 2022* vol. 12. doi:10.3390/app12199851

Naeem Seliya, Taghi M. Khoshgoftaar, and Jason Hulse (2009). A Study on the Relationships of Classifier Performance Metrics. 2009 *21st IEEE International Conference on Tools with Artificial Intelligence*. doi:10.1109/ICTAI.2009.25

Alexei Botchkarev (2019). A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 14, pp. 45-79. doi:10.28945/4184

Odd Gundersen, Saeid Shamsaliei, and Richard Juul Isdah (2022). Do machine learning platforms provide out-of-the-box reproducibility? *Future Generation Computer Systems*. vol. 126, pp. 34-47

Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, Arnor Solberg (2013). Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. *2013 IEEE Sixth International Conference on Cloud Computing.* Santa Clara, CA, USA

Laura Savu (2011). Cloud Computing, Deployment models, Delivery Models, Risks and Research Challenges. *2011 International Conference on Computer and Management (CAMAN)* doi:10.1109/CAMAN.2011.5778816