Today, demands on architects are much greater, and they often have to be knowledgeable not only on the ins and outs of data storage and consumption, but on how ML models work in tandem. This adds a lot of complexity to the role and increases their responsibility in the MLOps life cycle, and it's why in this section, we have called them machine learning architects instead of the more traditional "data architect" title.

Machine learning architects play a critical role in the ML model life cycle, ensuring a scalable and flexible environment for model pipelines. In addition, data teams need their expertise to introduce new technologies (when appropriate) that improve ML model performance in production. It is for this reason that the data architect title isn't enough; they need to have an intimate understanding of machine learning, not just enterprise architecture, to play this key role in the ML model life cycle.

This role requires collaboration across the enterprise, from data scientists and engineers to DevOps and software engineers. Without a complete understanding of the needs of each of these people and teams, machine learning architects cannot properly allocate resources to ensure optimal performance of ML models in production.

When it comes to MLOps, the machine learning architects' role is about having a centralized view of resource allocation. As they have a strategic, tactical role, they need an overview of the situation to identify bottlenecks and use that information to find long-term improvements. Their role is one of pinpointing possible new technology or infrastructure for investment, not necessarily operational quick fixes that don't address the heart of the scalability of the system.

Closing Thoughts

MLOps isn't just for data scientists; a diverse group of experts across the organization has a role to play not only in the ML model life cycle, but the MLOps strategy as well. In fact, each person—from the subject matter expert on the business side to the most technical machine learning architect—plays a critical part in the maintenance of ML models in production. This is ultimately important not only to ensure the best possible results from ML models (good results generally lead to more trust in ML-based systems as well as increased budget to build more), but, perhaps more pointedly, to protect the business from the risks outlined in Chapter 1.

CHAPTER 3

Key MLOps Features

Mark Treveil

MLOps affects many different roles across the organization and, in turn, many parts of the machine learning life cycle. This chapter introduces the five key components of MLOps (development, deployment, monitoring, iteration, and governance) at a high level as a foundation for Chapters 4 through 8, which delve into the more technical details and requirements of these components.

A Primer on Machine Learning

To understand the key features of MLOps, it's essential first to understand how machine learning works and be intimately familiar with its specificities. Though often overlooked in its role as a part of MLOps, ultimately algorithm selection (or how machine learning models are built) can have a direct impact on MLOps processes.

At its core, machine learning is the science of computer algorithms that automatically learn and improve from experience rather than being explicitly programmed. The algorithms analyze sample data, known as training data, to build a software model that can make predictions.

For example, an image recognition model might be able to identify the type of electricity meter from a photograph by searching for key patterns in the image that distinguish each type of meter. Another example is an insurance recommender model, which might suggest additional insurance products that a specific existing customer is most likely to buy based on the previous behavior of similar customers.

When faced with unseen data, be it a photo or a customer, the ML model uses what it has learned from previous data to make the best prediction it can based on the assumption that the unseen data is somehow related to the previous data.

ML algorithms use a wide range of mathematical techniques, and the models can take many different forms, from simple decision trees to logistic regression algorithms to much more complex deep learning models (see "What Is a Machine Learning Model?" on page 42 for details).

Model Development

Let's take a deeper look into ML model development as a whole for an even more complete understanding of its components, all of which can have an impact on MLOps after deployment.

Establishing Business Objectives

The process of developing a machine learning model typically starts with a business objective, which can be as simple as reducing fraudulent transactions to < 0.1% or having the ability to identify people's faces on their social media photos. Business objectives naturally come with performance targets, technical infrastructure requirements, and cost constraints; all of these factors can be captured as key performance indicators, or KPIs, which will ultimately enable the business performance of models in production to be monitored.

It's important to recognize that ML projects don't happen in a vacuum. They are generally part of a larger project that in turn impacts technologies, processes, and people. That means part of establishing objectives also includes change management, which may even provide some guidance for how the ML model should be built. For example, the required degree of transparency will strongly influence the choice of algorithms and may drive the need to provide explanations together with predictions so that predictions are turned into valuable decisions at the business level.

Data Sources and Exploratory Data Analysis

With clear business objectives defined, it is time to bring together subject matter experts and data scientists to begin the journey of developing the ML model. This starts with the search for suitable input data. Finding data sounds simple, but in practice, it can be the most arduous part of the journey.

Key questions for finding data to build ML models include:

- What relevant datasets are available?
- Is this data sufficiently accurate and reliable?
- How can stakeholders get access to this data?
- What data properties (known as *features*) can be made available by combining multiple sources of data?

- Will this data be available in real time?
- Is there a need to label some of the data with the "ground truth" that is to be predicted, or does unsupervised learning make sense? If so, how much will this cost in terms of time and resources?
- What platform should be used?
- How will data be updated once the model is deployed?
- Will the use of the model itself reduce the representativeness of the data?
- How will the KPIs, which were established along with the business objectives, be measured?

The constraints of data governance bring even more questions, including:

- Can the selected datasets be used for this purpose?
- What are the terms of use?
- Is there personally identifiable information (PII) that must be redacted or anonymized?
- Are there features, such as gender, that legally cannot be used in this business context?
- Are minority populations sufficiently well represented that the model has equivalent performances on each group?

Since data is the essential ingredient to power ML algorithms, it always helps to build an understanding of the patterns in data before attempting to train models. Exploratory data analysis (EDA) techniques can help build hypotheses about the data, identify data cleaning requirements, and inform the process of selecting potentially significant features. EDA can be carried out visually for intuitive insight and statistically if more rigor is required.

Feature Engineering and Selection

EDA leads naturally into feature engineering and feature selection. Feature engineering is the process of taking raw data from the selected datasets and transforming it into "features" that better represent the underlying problem to be solved. "Features" are arrays of numbers of fixed size, as it is the only object that ML algorithms understand. Feature engineering includes data cleansing, which can represent the largest part of an ML project in terms of time spent. For details, see "Feature Engineering and Selection" on page 47.

Training and Evaluation

After data preparation by way of feature engineering and selection, the next step is training. The process of training and optimizing a new ML model is iterative; several algorithms may be tested, features can be automatically generated, feature selections may be adapted, and algorithm hyperparameters tuned. In addition to—or in many cases because of—its iterative nature, training is also the most intensive step of the ML model life cycle when it comes to computing power.

Keeping track of the results of each experiment when iterating becomes complex quickly. Nothing is more frustrating to data scientists than not being able to re-create the best results because they cannot remember the precise configuration. An experiment tracking tool can greatly simplify the process of remembering the data, the features selection, and model parameters alongside the performance metrics. These enable experiments to be compared side-by-side, highlighting the differences in performance.

Deciding what is the best solution requires both quantitative criteria, such as accuracy or average error, and qualitative criteria regarding the explainability of the algorithm or its ease of deployment.

Reproducibility

While many experiments may be short-lived, significant versions of a model need to be saved for possible later use. The challenge here is reproducibility, which is an important concept in experimental science in general. The aim in ML is to save enough information about the environment the model was developed in so that the model can be reproduced with the same results from scratch.

Without reproducibility, data scientists have little chance of being able to confidently iterate on models, and worse, they are unlikely to be able to hand over the model to DevOps to see if what was created in the lab can be faithfully reproduced in production. True reproducibility requires version control of all the assets and parameters involved, including the data used to train and evaluate the model, as well as a record of the software environment (see "Version Management and Reproducibility" on page 56 for details).

Responsible AI

Being able to reproduce the model is only part of the operationalization challenge; the DevOps team also needs to understand how to verify the model (i.e., what does the model do, how should it be tested, and what are the expected results?). Those in highly regulated industries are likely required to document even more detail, including how the model was built and how it was tuned. In critical cases, the model may be independently recoded and rebuilt.

Documentation is still the standard solution to this communication challenge. Automated model document generation, whereby a tool automatically creates documentation associated with any trained model, can make the task less onerous. But in almost all cases, some documentation will need to be written by hand to explain the choices made.

It is a fundamental consequence of their statistical nature that ML models are challenging to understand. While model algorithms come with standard performance measures to assess their efficacy, these don't explain how the predictions are made. The "how" is important as a way to sanity-check the model or help better engineer features, and it may be necessary to ensure that fairness requirements (e.g., around features like sex, age, or race) have been met. This is the field of explainability, which is connected to Responsible AI as discussed in Chapter 1 and which will be discussed in further detail in Chapter 4.

Explainability techniques are becoming increasingly important as global concerns grow about the impact of unbridled AI. They offer a way to mitigate uncertainty and help prevent unintended consequences. The techniques most commonly used today include:

- Partial dependence plots, which look at the marginal impact of features on the predicted outcome
- Subpopulation analyses, which look at how the model treats specific subpopulations and that are the basis of many fairness analyses
- Individual model predictions, such as Shapley values, which explain how the value of each feature contributes to a specific prediction
- What-if analysis, which helps the ML model user to understand the sensitivity of the prediction to its inputs

As we've seen in this section, even though model development happens very early on, it's still an important place to incorporate MLOps practices. Any MLOps work done up front during the model development stage will make the models easier to manage down the line (especially when pushing to production).

Productionalization and Deployment

Productionalizing and deploying models is a key component of MLOps that presents an entirely different set of technical challenges than developing the model. It is the domain of the software engineer and the DevOps team, and the organizational challenges in managing the information exchange between the data scientists and these teams must not be underestimated. As touched on in Chapter 1, without effective collaboration between the teams, delays or failures to deploy are inevitable.

Model Deployment Types and Contents

To understand what happens in these phases, it's helpful to take a step back and ask: what exactly is going into production, and what does a model consist of? There are commonly two types of model deployment:

Model-as-a-service, or live-scoring model
: Typically the model is deployed into a simple framework to provide a REST API endpoint (the means from which the API can access the resources it needs to perform the task) that responds to requests in real time.

Embedded model
: Here the model is packaged into an application, which is then published. A common example is an application that provides batch-scoring of requests.

What to-be-deployed models consist of depends, of course, on the technology chosen, but typically they comprise a set of code (commonly Python, R, or Java) and data artifacts. Any of these can have version dependencies on runtimes and packages that need to match in the production environment because the use of different versions may cause model predictions to differ.

One approach to reducing dependencies on the production environment is to export the model to a portable format such as PMML, PFA, ONNX, or POJO. These aim to improve model portability between systems and simplify deployment. However, they come at a cost: each format supports a limited range of algorithms, and sometimes the portable models behave in subtly different ways than the original. Whether or not to use a portable format is a choice to be made based on a thorough understanding of the technological and business context.

Containerization

Containerization is an increasingly popular solution to the headaches of dependencies when deploying ML models. Container technologies such as Docker are lightweight alternatives to virtual machines, allowing applications to be deployed in independent, self-contained environments, matching the exact requirements of each model.

They also enable new models to be seamlessly deployed using the blue-green deployment technique.[1] Compute resources for models can be scaled elastically using multiple containers, too. Orchestrating many containers is the role of technologies such as Kubernetes and can be used both in the cloud and on-premise.

1 Describing the blue-green deployment technique will require more space than we have here. For more information, see Martin Fowler's blog.

Model Deployment Requirements

So what about the productionalization process between completing model development and physically deploying into production—what needs to be addressed? One thing is for sure: rapid, automated deployment is always preferred to labor-intensive processes.

For short-lifetime, self-service applications, there often isn't much need to worry about testing and validation. If the maximum resource demands of the model can be securely capped by technologies such as Linux cgroups, then a fully automated single-step push-to-production may be entirely adequate. It is even possible to handle simple user interfaces with frameworks like Flask when using this lightweight deployment mode. Along with integrated data science and machine learning platforms, some business rule management systems may also allow some sort of autonomous deployment of basic ML models.

In customer-facing, mission-critical use cases, a more robust CI/CD pipeline is required. This typically involves:

1. Ensuring all coding, documentation and sign-off standards have been met
2. Re-creating the model in something approaching the production environment
3. Revalidating the model accuracy
4. Performing explainability checks
5. Ensuring all governance requirements have been met
6. Checking the quality of any data artifacts
7. Testing resource usage under load
8. Embedding into a more complex application, including integration tests

In heavily regulated industries (e.g., finance and pharmaceuticals), governance and regulatory checks will be extensive and are likely to involve manual intervention. The desire in MLOps, just as in DevOps, is to automate the CI/CD pipeline as far as possible. This not only speeds up the deployment process, but it enables more extensive regression testing and reduces the likelihood of errors in the deployment.

Monitoring

Once a model is deployed to production, it is crucial that it continue to perform well over time. But good performance means different things to different people, in particular to the DevOps team, to data scientists, and to the business.

DevOps Concerns

The concerns of the DevOps team are very familiar and include questions like:

- Is the model getting the job done quickly enough?
- Is it using a sensible amount of memory and processing time?

This is traditional IT performance monitoring, and DevOps teams know how to do this well already. The resource demands of ML models are not so different from traditional software in this respect.

Scalability of compute resources can be an important consideration, for example, if you are retraining models in production. Deep learning models have greater resource demands than much simpler decision trees. But overall, the existing expertise in DevOps teams for monitoring and managing resources can be readily applied to ML models.

Data Scientist Concerns

The data scientist is interested in monitoring ML models for a new, more challenging reason: they can degrade over time, since ML models are effectively models of the data they were trained on. This is not a problem faced by traditional software, but it is inherent to machine learning. ML mathematics builds a concise representation of the important patterns in the training data with the hope that this is a good reflection of the real world. If the training data reflects the real world well, then the model should be accurate and, thus, useful.

But the real world doesn't stand still. The training data used to build a fraud detection model six months ago won't reflect a new type of fraud that has started to occur in the last three months. If a given website starts to attract an increasingly younger user base, then a model that generates advertisements is likely to produce less and less relevant adverts. At some point, the performance will become unacceptable, and model retraining becomes necessary. How soon models need to be retrained depends on how fast the real world is changing and how accurate the model needs to be, but also, importantly, on how easy it is to build and deploy a better model.

But first, how can data scientists tell a model's performance is degrading? It's not always easy. There are two common approaches, one based on ground truth and the other on input drift.

Ground truth

The ground truth, put simply, is the correct answer to the question that the model was asked to solve—for example, "Is this credit card transaction actually fraudulent?"

In knowing the ground truth for all the predictions a model has made, one can judge how well that model is performing.

Sometimes ground truth is obtained rapidly after a prediction—for example, in models that decide which advertisements to display to a user on a web page. The user is likely to click on the advertisements within seconds, or not at all. However, in many use cases, obtaining the ground truth is much slower. If a model predicts that a transaction is fraudulent, how can this be confirmed? In some cases, verification may only take a few minutes, such as a phone call placed to the cardholder. But what about the transactions the model thought were OK but actually weren't? The best hope is that they will be reported by the cardholder when they review their monthly transactions, but this could happen up to a month after the event (or not at all).

In the fraud example, ground truth isn't going to enable data science teams to monitor performance accurately on a daily basis. If the situation requires rapid feedback, then input drift may be a better approach.

Input drift

Input drift is based on the principle that a model is only going to predict accurately if the data it was trained on is an accurate reflection of the real world. So if a comparison of recent requests to a deployed model against the training data shows distinct differences, then there is a strong likelihood that the model performance is compromised. This is the basis of input drift monitoring. The beauty of this approach is that all the data required for this test already exists, so there is no need to wait for ground truth or any other information.

Identifying drift is one of the most important components of an adaptable MLOps strategy, and one that can bring agility to the organization's enterprise AI efforts overall. Chapter 7 will go into more technical depth about data scientists' concerns when it comes to model monitoring.

Business Concerns

The business has a holistic outlook on monitoring, and some of its concerns might include questions like:

- Is the model delivering value to the enterprise?
- Do the benefits of the model outweigh the cost of developing and deploying it? (And how can we measure this?)

The KPIs identified for the original business objective are one part of this process. Where possible, these should be monitored automatically, but this is rarely trivial. The objective of reducing fraud to less than 0.1% of transactions in our previous example is reliant on establishing the ground truth. But even monitoring this doesn't answer the question: what is the net gain to the business in dollars?

This is an age-old challenge for software, and with ever-increasing expenditure on ML, the pressure for data scientists to demonstrate value is only going to grow. In the absence of a "dollar-o-meter," effectively monitoring the business KPIs is the best option available (see "Design and Manage Experiments" on page 138). The choice of the baseline is important here and should ideally allow for differentiation of the value of the ML subproject specifically, rather than that of the global project. For example, the ML performance can be assessed with respect to a rule-based decision model based on subject matter expertise to set apart the contribution of decision automation and of ML per se.

Iteration and Life Cycle

Developing and deploying improved versions of a model is an essential part of the MLOps life cycle, and one of the more challenging. There are various reasons to develop a new model version, one of which is model performance degradation due to model drift, as discussed in the prior section. Sometimes there is a need to reflect refined business objectives and KPIs, and other times, it's just that the data scientists have come up with a better way to design the model.

Iteration

In some fast-moving business environments, new training data becomes available every day. Daily retraining and redeployment of the model are often automated to ensure that the model reflects recent experience as closely as possible.

Retraining an existing model with the latest training data is the simplest scenario for iterating a new model version. But while there are no changes to feature selection or algorithm, there are still plenty of pitfalls. In particular:

- Does the new training data look as expected? Automated validation of the new data through predefined metrics and checks is essential.
- Is the data complete and consistent?
- Are the distributions of features broadly similar to those in the previous training set? Remember that the goal is to refine the model, not radically change it.

With a new model version built, the next step is to compare the metrics with the current live model version. Doing so requires evaluating both models on the same development dataset, whether it be the previous or latest version. If metrics and checks suggest a wide variation between the models, automated scripts should not be redeployed, and manual intervention should be sought.

Even in the "simple" automated retraining scenario with new training data, there is a need for multiple development datasets based on scoring data reconciliation (with ground truth when it becomes available), data cleaning and validation, the previous model version, and a set of carefully considered checks. Retraining in other scenarios is likely to be even more complicated, rendering automated redeployment unlikely.

As an example, consider retraining motivated by the detection of significant input drift. How can the model be improved? If new training data is available, then retraining with this data is the action with the highest cost-benefit ratio, and it may suffice. However, in environments where it's slow to obtain the ground truth, there may be little new labeled data.

This case requires direct invention from data scientists who need to understand the cause of the drift and work out how the existing training data could be adjusted to more accurately reflect the latest input data. Evaluating a model generated by such changes is difficult. The data scientist has to spend time assessing the situation—time that increases with the amount of modeling debt—as well as estimate the potential impact on performance and design custom mitigation measures. For example, removing a specific feature or sampling the existing rows of training data may lead to a better-tuned model.

The Feedback Loop

In large enterprises, DevOps best practices typically dictate that the live model scoring environment and the model retraining environment are distinct. As a result, the evaluation of a new model version on the retraining environment is likely to be compromised.

One approach to mitigating this uncertainty is shadow testing, where the new model version is deployed into the live environment alongside the existing model. All live scoring is handled by the incumbent model version, but each new request is then scored again by the new model version and the results logged, but not returned to the requestor. Once sufficient requests have been scored by both versions, the results can be compared statistically. Shadow scoring also gives more visibility to the SMEs on the future versions of the model and may thus allow for a smoother transition.

In the advertisement generation model previously discussed, it is impossible to tell if the ads selected by the model are good or bad without allowing the end user the chance to click on them. In this use case, shadow testing has limited benefits, and A/B testing is more common.

In A/B testing, both models are deployed into the live environment, but input requests are split between the two models. Each request is processed by one or the other model, not both. Results from the two models are logged for analysis (but never for the same request). Drawing statistically meaningful conclusions from an A/B test requires careful planning of the test.

Chapter 7 will cover the how-to of A/B testing in more detail, but as a preview, the simplest form of A/B testing is often referred to as a fixed-horizon test. That's because in the search for a statistically meaningful conclusion, one has to wait until the carefully predetermined number of samples have been tested. "Peeking" at the result before the test is finished is unreliable. However, if the test is running live in a commercial environment, every bad prediction is likely to cost money, so not being able to stop a test early could be expensive.

Bayesian, and in particular multi-armed bandit, tests are an increasingly popular alternative to the "frequentist" fixed-horizon test, with the aim of drawing conclusions more quickly. Multi-armed bandit testing is adaptive: the algorithm that decides the split between models adapts according to live results and reduces the workload of underperforming models. While multi-armed bandit testing is more complex, it can reduce the business cost of sending traffic to a poorly performing model.

Iterating on the Edge

Iterating on an ML model deployed to millions of devices, such as smartphones, sensors, or cars, presents different challenges to iteration in a corporate IT environment. One approach is to relay all the feedback from the millions of model instances to a central point and perform training centrally. Tesla's autopilot system, running in more than 500,000 cars, does exactly this. Full retraining of their 50 or so neural networks takes 70,000 GPU hours.

Google has taken a different approach with its smartphone keyboard software, GBoard. Instead of centralized retraining, every smartphone retrains the model locally and sends a summary of the improvements it has found to Google centrally. These improvements from every device are averaged and the shared model updated. This federated learning approach means that an individual user's personal data doesn't need to be collected centrally, the improved model on each phone can be used immediately, and the overall power consumption goes down.

Governance

Governance is the set of controls placed on a business to ensure that it delivers on its responsibilities to all stakeholders, from shareholders and employees to the public and national governments. These responsibilities include financial, legal, and ethical obligations. Underpinning all three of these is the fundamental principle of fairness.

Legal obligations are the easiest to understand. Businesses were constrained by regulations long before the advent of machine learning. Many regulations target specific industries; for example, financial regulations aim to protect the public and wider economy from finance mismanagement and fraud, while pharmaceutical industries must comply with rules to safeguard the health of the public. Business practice is impacted by broader legislation to protect vulnerable sectors of society and to ensure a level playing field on criteria such as sex, race, age, or religion.

Recently, governments across the world have imposed regulations to protect the public from the impact of the use of personal data by businesses. The 2016 EU General Data Protection Regulation (GDPR) and the 2018 California Consumer Privacy Act (CCPA) typify this trend, and their impact on ML—with its total dependency on data—has been immense. For example, GDPR attempts to protect personal data from industrial misuse with a goal of limiting the potential discrimination against individuals.

GDPR Principles

The GDPR sets out principles for the processing of personal data, and it's worth noting that the CCPA was built to closely mirror its principles, though it does have some significant differences.[2] Processing includes the collection, storage, alteration, and use of personal data. These principles are:

- Lawfulness, fairness, and transparency
- Purpose limitation
- Data minimization
- Accuracy
- Storage limitation
- Integrity and confidentiality (security)
- Accountability

Governments are now starting to turn their regulatory eye to ML specifically, hoping to mitigate the negative impact of its use. The European Union is leading the way with planned legislation to define the acceptable uses of various forms of AI. This is not necessarily about reducing use; for example, it may enable beneficial applications of facial recognition technology that are currently restricted by data privacy regulations. But what is clear is that businesses will have to take heed of yet more regulation when applying ML.

Do businesses care about moral responsibilities to society, beyond formal legislation? Increasingly, the answer is yes, as seen in the current development of environmental,

2 Delve into the differences between GDPR and CCPA.

social, and governance (ESG) performance indicators. Trust matters to consumers, and a lack of trust is bad for business. With increasing public activism on the subject, businesses are engaging with ideas of Responsible AI, the ethical, transparent, and accountable application of AI technology. Trust matters to shareholders, too, and full disclosure of ML risks is on its way.

Applying good governance to MLOPs is challenging. The processes are complex, the technology is opaque, and the dependence on data is fundamental. Governance initiatives in MLOps broadly fall into one of two categories:

Data governance
: A framework for ensuring appropriate use and management of data

Process governance
: The use of well-defined processes to ensure all governance considerations have been addressed at the correct point in the life cycle of the model and that a full and accurate record has been kept

Data Governance

Data governance concerns itself with the data being used, especially that for model training, and it can address questions like:

- What is the data's provenance?
- How was the original data collected and under what terms of use?
- Is the data accurate and up to date?
- Is there PII or other forms of sensitive data that should not be used?

ML projects usually involve significant pipelines, consisting of data cleaning, combination, and transformation steps. Understanding the data lineage is complex, especially at the feature level, but it is essential for compliance with GDPR-style regulations. How can teams—and more broadly organizations, as it matters at the top as well—be sure that no PII is used to train a given model? Anonymizing or pseudo-anonymizing data is not always a sufficient solution to managing personal information. If these processes are not performed correctly, it can still be possible to single out an individual and their data, contrary to the requirements of GDPR.[3]

Inappropriate biases in models can arise quite accidentally despite the best intentions of data scientists. An ML recruitment model famously discriminated against women by identifying that certain schools—all-female schools—were less well represented in

3 For more on anonymization, pseudo-anonymization, and why they don't solve all data privacy woes, we recommend *Executing Data Privacy-Compliant Data Projects* by Dataiku.

the company's upper management, which reflected the historical dominance of men in the organization.[4] The point is that making predictions based on experience is a powerful technique, but sometimes the consequences are not only counter-productive, but illegal.

Data governance tools that can address these problems are in their infancy. Most focus on answering these two questions about data lineage:

- Where did the information in this dataset come from, and what does this tell me about how I can use it?
- How is this dataset used, and if I change it in some way, what are the implications downstream?

Neither question is easy to answer fully and accurately in real-world data preparation pipelines. For example, if a data scientist writes a Python function to in-memory process several input datasets and output a single dataset, how can one be sure from what information each cell of the new dataset was derived?

Process Governance

Process governance focuses on formalizing the steps in the MLOps process and associating actions with them. Typically these actions are reviews, sign-offs, and the capture of supporting materials, such as documentation. The aim is twofold:

- To ensure every governance-related consideration is made at the correct time and correctly acted upon. For example, models should not be deployed to production until all validation checks have been passed.
- To enable oversight from outside of the strict MLOps process. Auditors, risk managers, compliance officers, and the business as a whole all have an interest in being able to track progress and review decisions at a later stage.

Effective implementation of process governance is hard:

- Formal processes for the ML life cycle are rarely easy to define accurately. The understanding of the complete process is usually spread across the many teams involved, often with no one person having a detailed understanding of it as a whole.
- For the process to be applied successfully, every team must be willing to adopt it wholeheartedly.

4 In 2018, Amazon famously scrapped an AI recruiting tool because of its bias against women.

- If the process is just too heavy-weight for some use-cases, teams will certainly subvert it, and much of the benefit will be lost.

Today, process governance is most commonly found in organizations with a traditionally heavy burden of regulation and compliance, such as finance. Outside of these, it is rare. With ML creeping into all spheres of commercial activity and with rising concern about Responsible AI, we will need new and innovative solutions to this problem that can work for all businesses.

Closing Thoughts

Given this overview of features required for and processes affected by MLOps, it's clearly not something data teams—or even the data-driven organization at large—can ignore. Nor is it an item to check off of a list ("yes, we do MLOps!"), but rather a complex interplay between technologies, processes, and people that requires discipline and time to do correctly.

The following chapters go deeper into each of the ML model life cycle components at play in MLOps, providing a look at how each should be done to get closer to the ideal MLOps implementation.

PART II

MLOps: How

CHAPTER 4

Developing Models

Adrien Lavoillotte

Anyone who wants to be serious about MLOps needs to have at least a cursory understanding of the model development process, which is presented in Figure 4-1 as an element of the larger ML project life cycle. Depending on the situation, the model development process can range from quite simple to extremely complex, and it dictates the constraints of subsequent usage, monitoring, and maintenance of models.

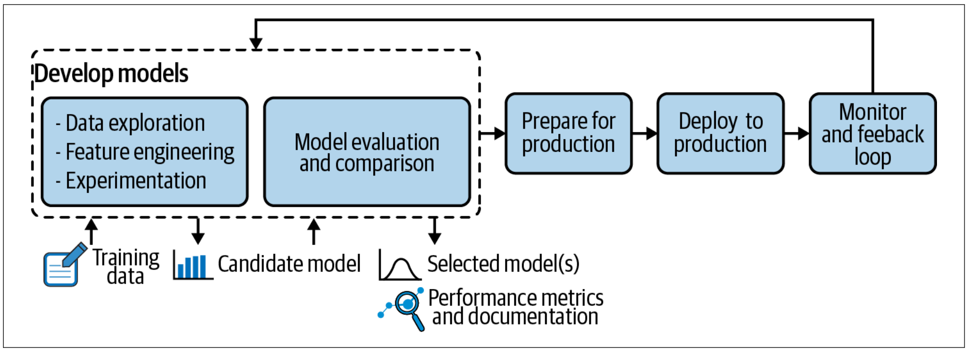

Figure 4-1. Model development highlighted in the larger context of the ML project life cycle

The implications of the data collection process on the rest of the model's life is quite straightforward, and one easily sees how a model can become stale. For other parts of the model, the effects may be less obvious.

For example, take feature creation, where feeding a date to the model versus a flag indicating whether the day is a public holiday may make a big difference in performance, but also comes with significantly different constraints on updating the model. Or consider how the metrics used for evaluating and comparing models may enable