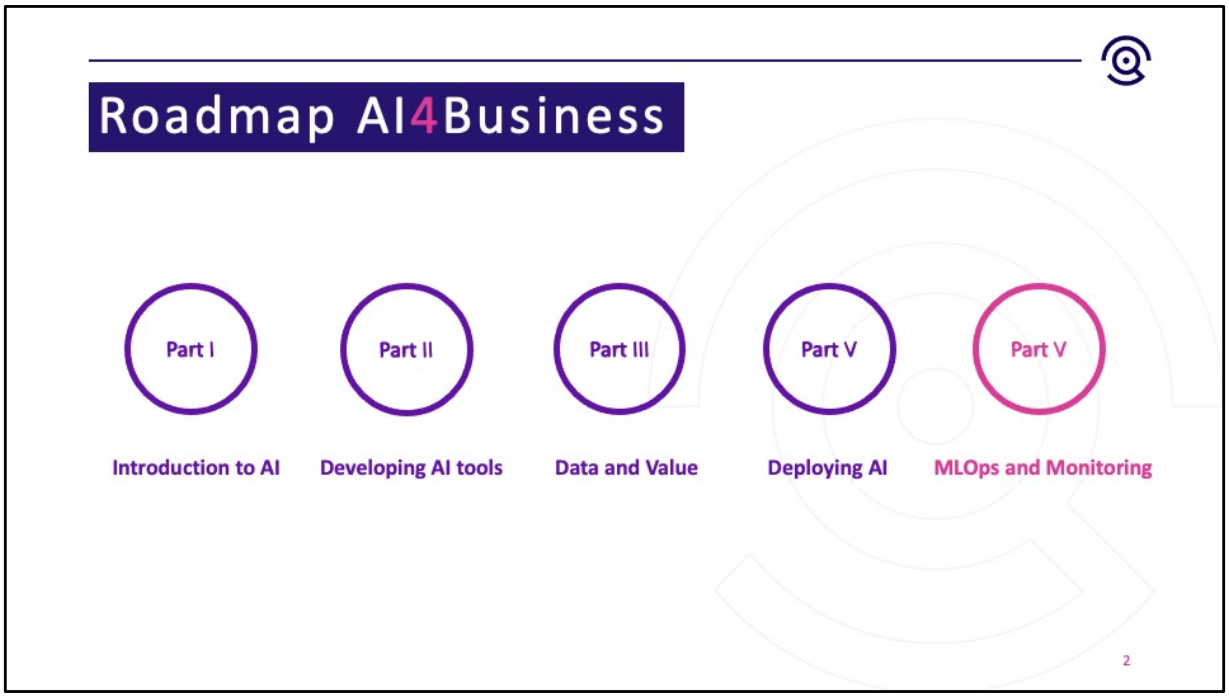
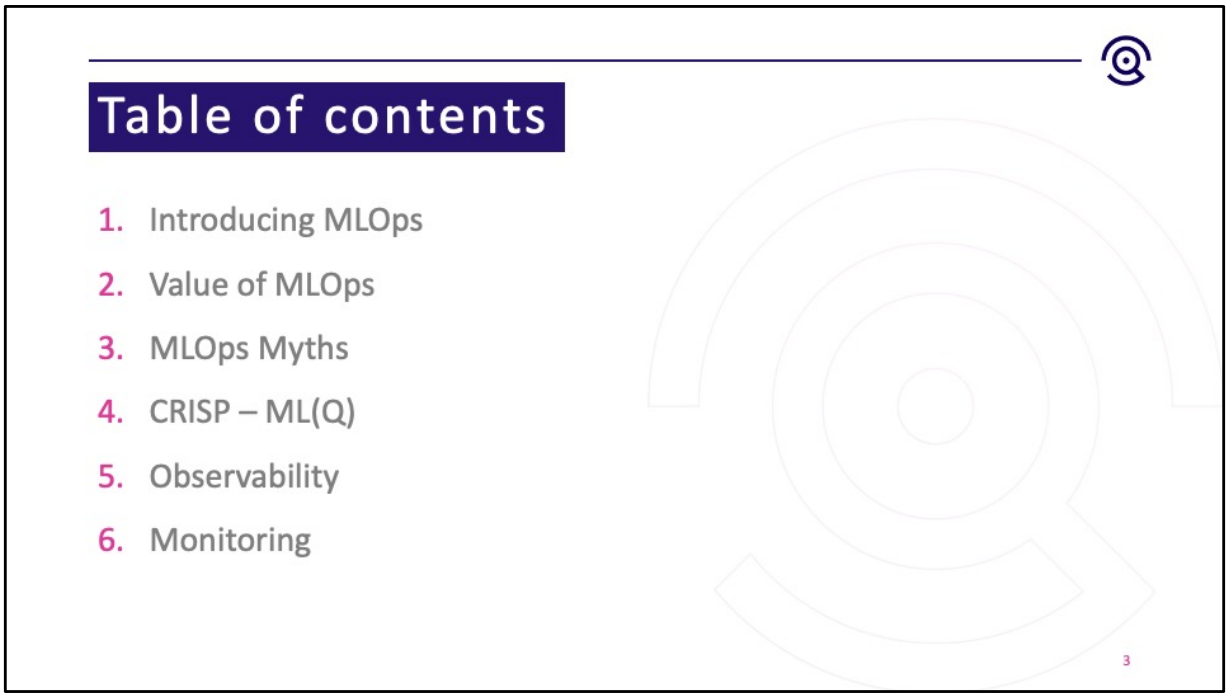
Welcome to the fifth and also final module of the AI4Business course. In this module we aim to provide a general introduction to Machine Learning Operations (or MLOps) as the practice that makes developing and maintenance of machine learning systems in production smooth and efficient. We also briefly introduce the monitoring aspect which is very important for all your AI solutions running in production and generating positive business value.

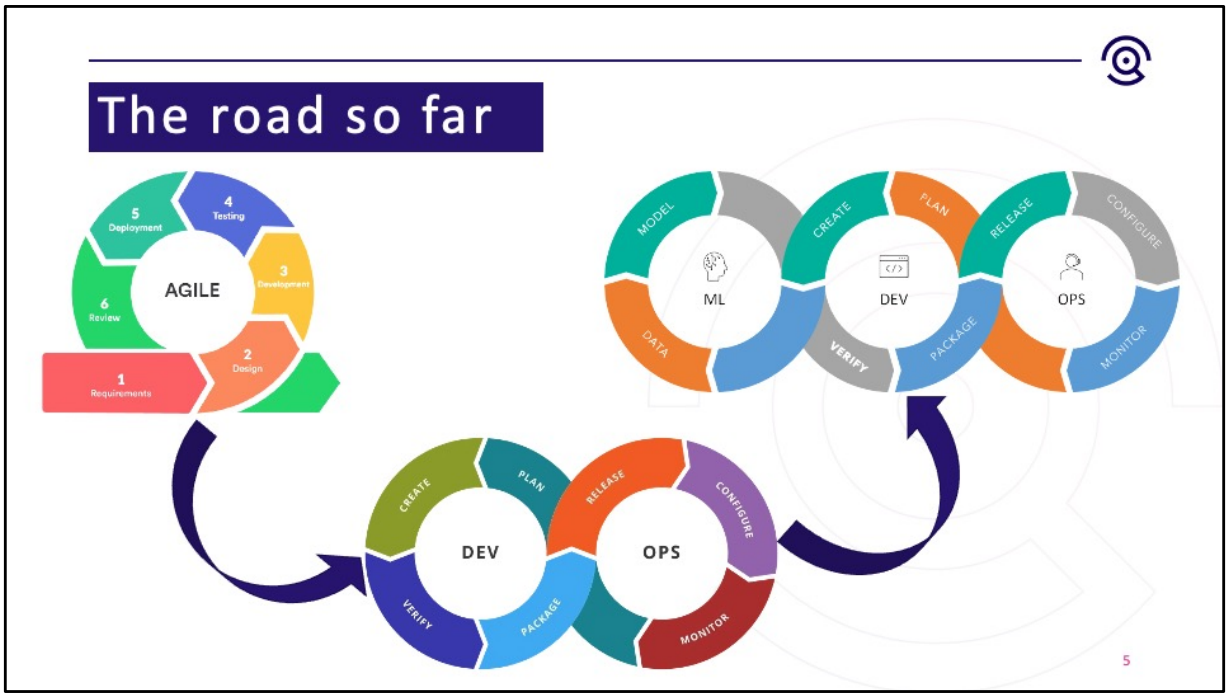


This module is the last out of five modules in the AI4Business course. This final module will put focus on guidelines, best practices and frameworks that assist you in maximizing the business value that you can capture from your AI solutions in production. Proper model deployment, maintenance and monitoring are extremely important to keep capturing that value over the lifetime or your AI system.



Today we cover the basics of MLOps and monitoring. We start by introducing MLOps, or Machine Learning Operations, by discussing the difference with DevOps and the four MLOps pillars. Next, we highlight the value of MLOps and we demystify some of the common myths. Afterwards, we introduce the new process model CRISP-ML(Q) for the development of machine learning applications. We briefly touch upon the idea of AI observability, a very important concept when it comes to monitoring performance of AI in real-life business situations. We finish with an overview of monitoring best practices with a focus on data quality. A lot of these topics are quite advanced and therefore not the most important thing to do in your very first AI project, but then they are very important to keep in mind for the future.

Let’s start with an introduction to MLOps.



If you want your organization to be able to operationalize your AI solutions, then you should understand that, at its core, ML engineering shares a lot of commonalities with software development. During the first tech boom, the agile framework helped to operationalize the product life cycle. After this, the adoption of DevOps took this to new high levels, helping further optimize the production lifecycle while fast-tracking the introduction of a brand new element: Big Data.

Nowadays, with more and more business trying to leverage their data through advanced analytics and AI, we seem to be in another wave of operationalization. However, processes and tools that we already perfected for software development do not seem to be enough when combining machine learning and operations together. In this part of the course, we are going to learn how to improve the whole project life cycle of AI systems by introducing Machine Learning Operations or MLOps tools and principles.

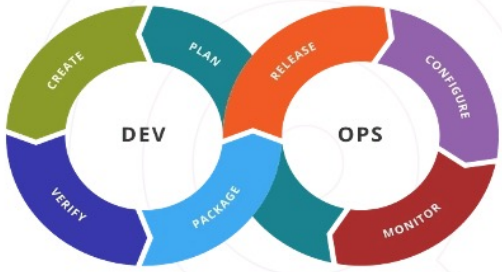
DevOps

What is DevOps?

Tools and best practices to improve software development process and operations

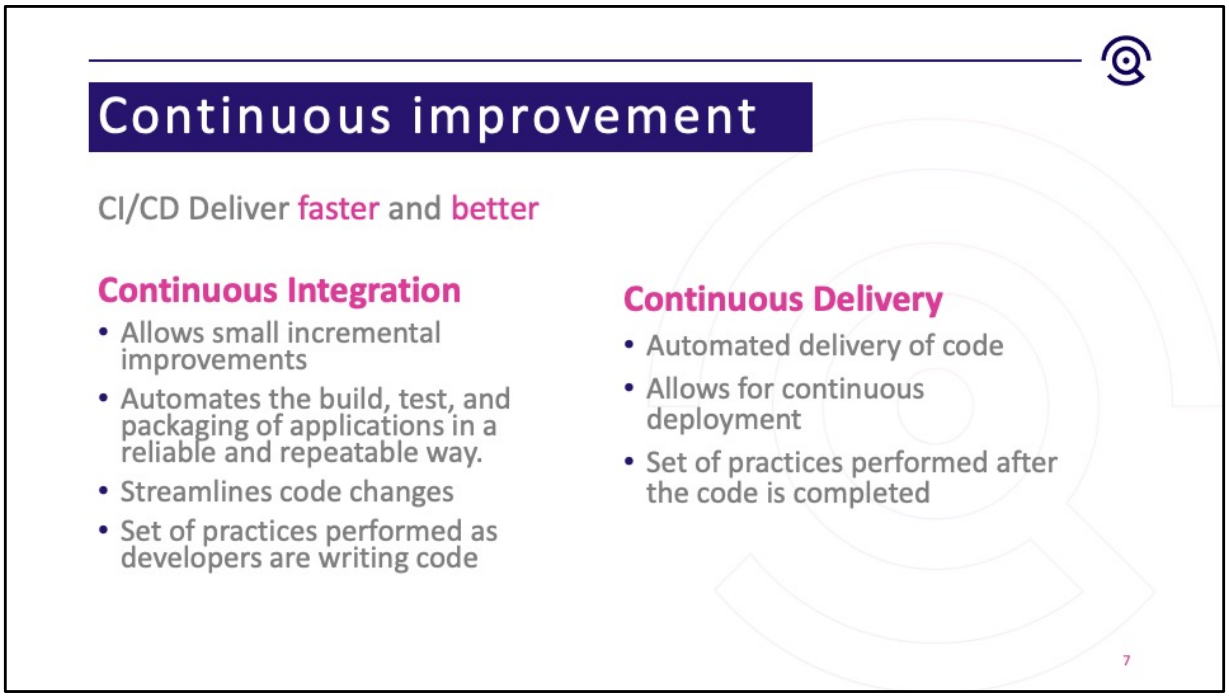
Advantages:

- Speed
- Rapid Delivery
- Reliability
- Improved collaboration
- Security



6

- Moreover, it improves collaboration. Under a DevOps model, developers and operations teams collaborate closely, share responsibilities, and combine their workflows. This reduces inefficiencies and saves time.
- And last but not least, there is the issue of security. You can adopt a DevOps model without sacrificing security by using automated, integrated security testing tools.



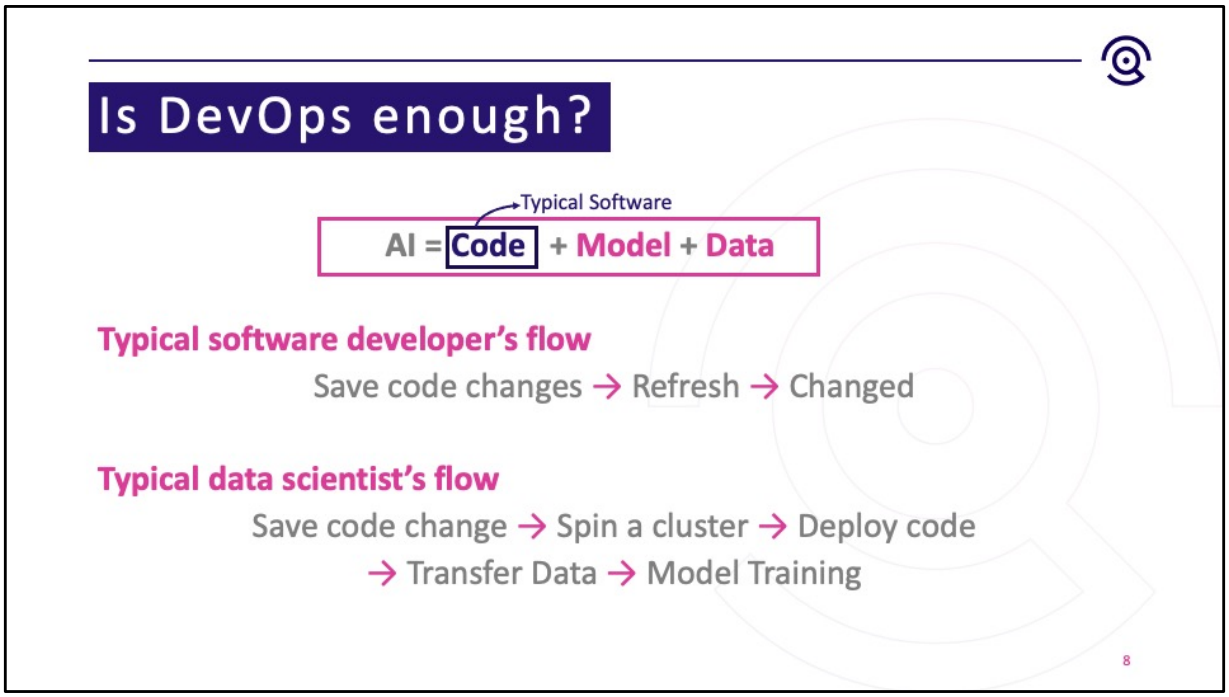
CI/CD stands for continuous integration and continuous delivery and forms an important part of DevOps. CI/CD processes bring a lot of reliability to software development, which is probably what most development teams appreciate the most about this model. This allows to exploit the benefits of agile development by continuously improving the solution in small iterations, which means that software teams can not only deliver results faster but also with a better quality.

Let's start with Continuous Integration or CI:

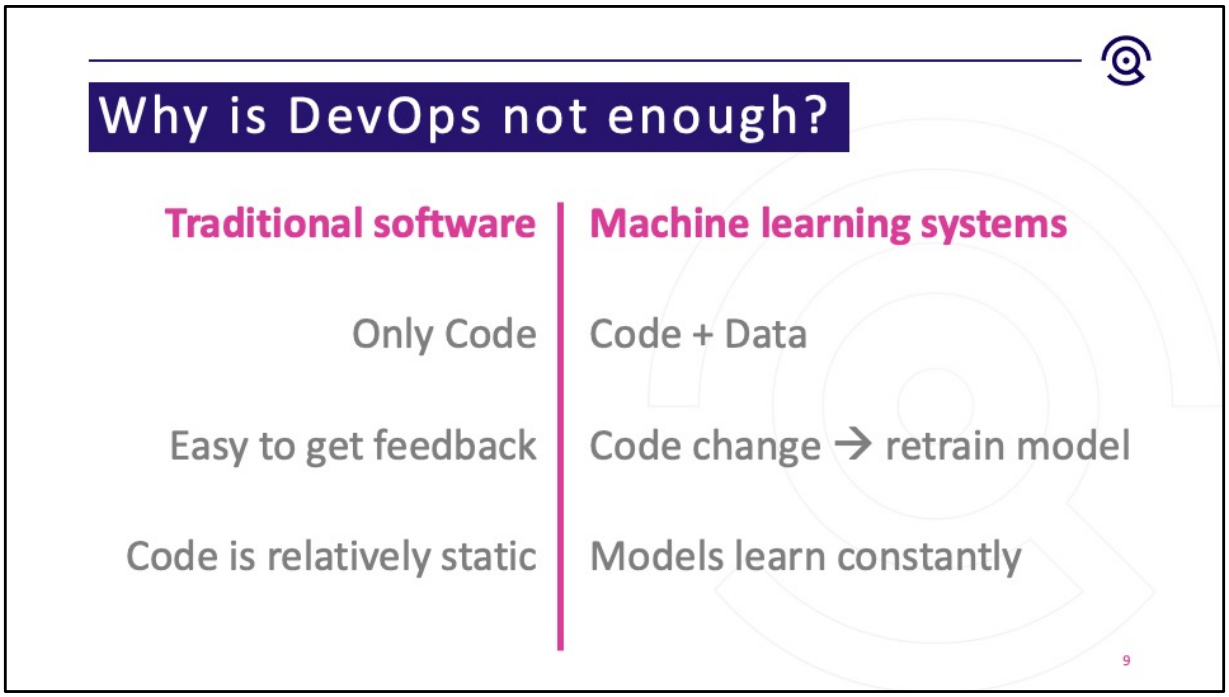
- CI is the practice that involves developers making small changes and checks to their code, which allows for small incremental improvements
- Due to the scale of the requirements and the number of steps involved, this process is automated to ensure that teams can build, test, and package their applications in a reliable and repeatable way.
- It also helps streamline code changes, thereby increasing time for developers to make changes and contribute to improved software.
- In short, CI can be seen as the set of practices performed as developers are writing code.

Now lets look at the CD side of the equation or Continuous Delivery:

- CD is the automated delivery of completed code to environments like testing and development. CD provides an automated and consistent way for code to be delivered to these environments.
- It also allows for continues deployment, meaning that every change that passes the automated tests is automatically placed in production, resulting in many production deployments.
- In contrast to CI, CD can be summarized as the set of practices performed after the code is completed.



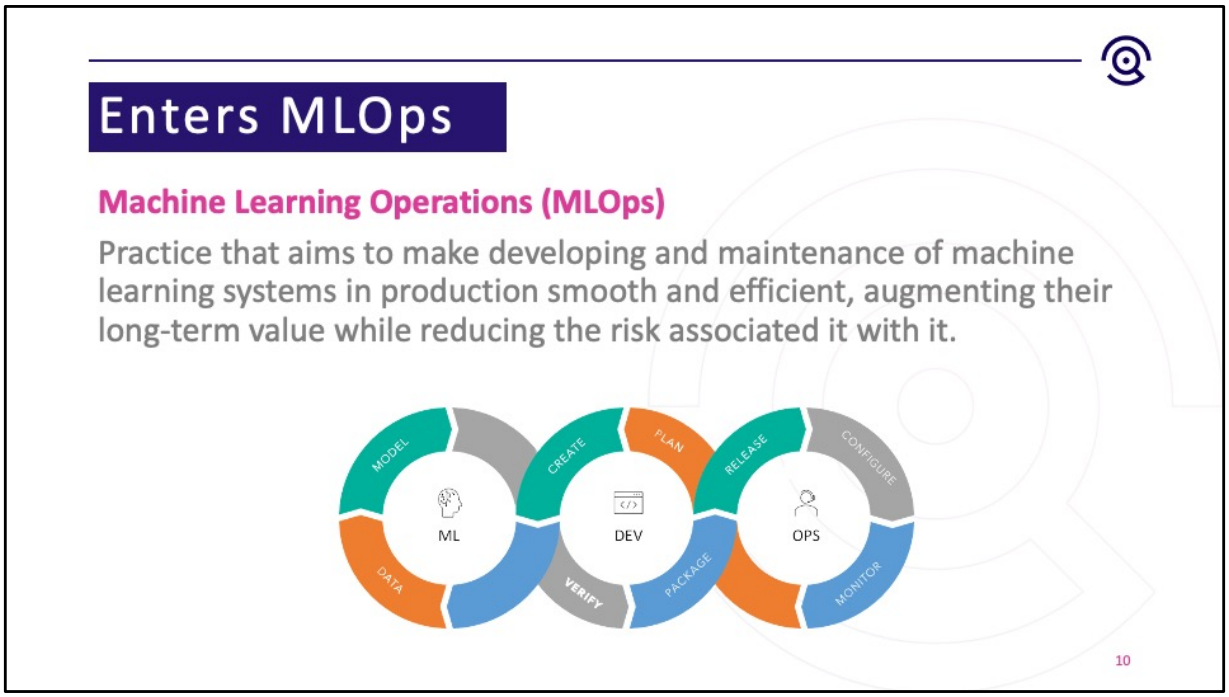
Now comes an important question. Can we take DevOps as is and just apply it to AI problems? As we have mentioned before, unlike typical software development, ML systems do not only contain software code/ ML also deals with two extra artifacts, namely models and data. This makes the ML and software development already very different form each other, with the ML workflow being more convoluted and messy.
A software developer would save the code changes, refresh those changes and voila, the code is already in a new state.
When a data scientist would like to improve the code similarly after saving the code changes, this probably requires to spin a cluster, deploy the code, transfer the data and train a model before the change takes effect.
All this also tends to translate into the operations after the code have been industrialized. Especially as many AI/ML professional are not always specifically trained for software developments.



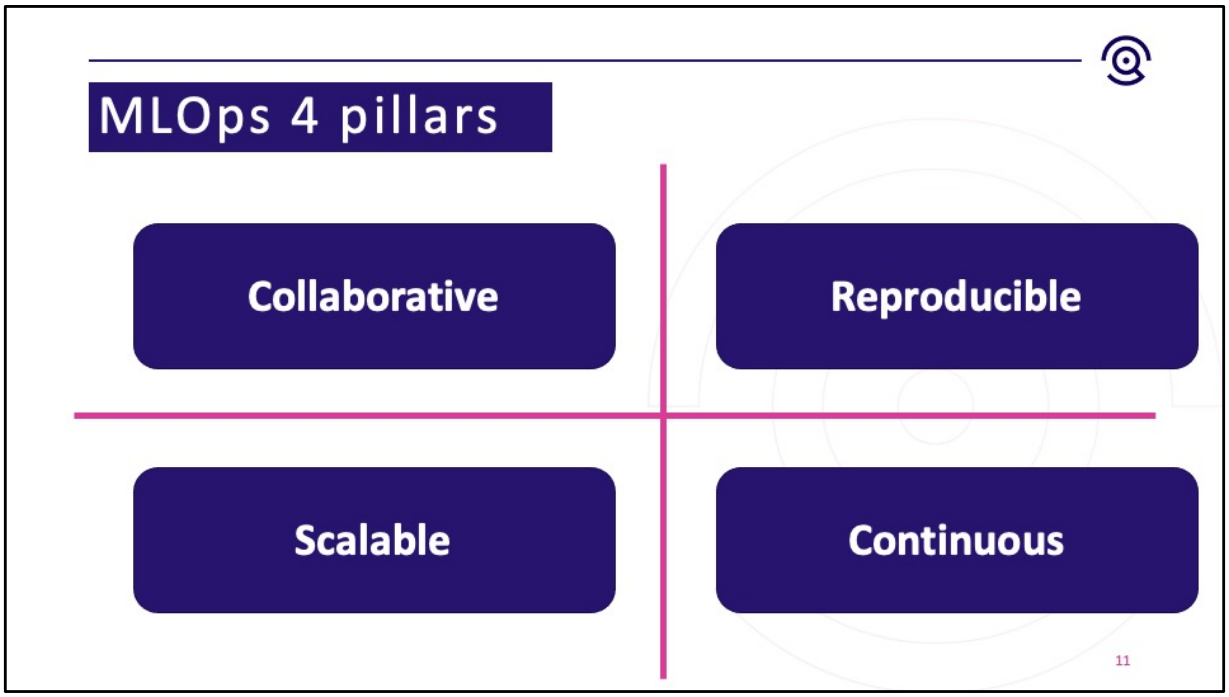
So why is DevOps not enough? What makes it so difficult to apply DevOps to AI solutions?

- In the first place, as we already know, traditional software is mostly exclusively code. ML is not just code, it's code plus data and some other artifacts like models.
- Moreover, while developing software it is normally easy to get instant feedback. In most cases, when developing ML systems, a change in code implies retraining the model making feedback not instant.
- And finally, software code is relatively static. ML software is constantly learning and accommodating for the changes in the real world.

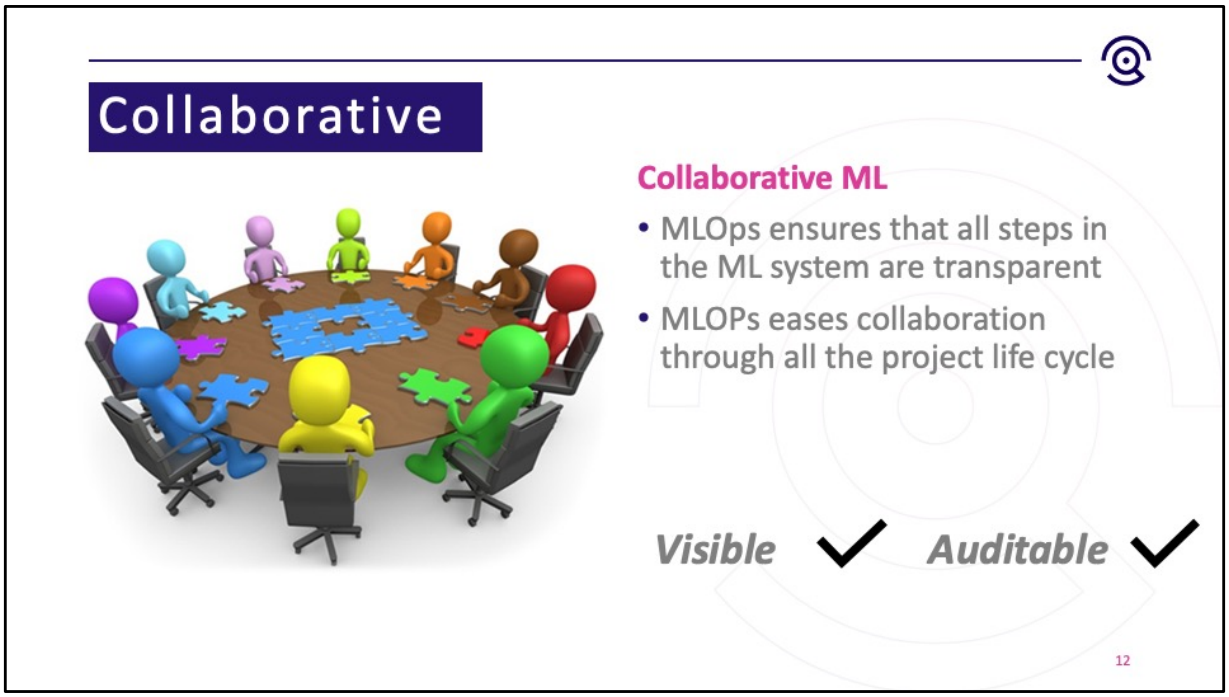
Because of these reasons, and in order to serve the AI project life cycle better, there was a need to expand the scope of the DevOps principles. Enters MLOps.



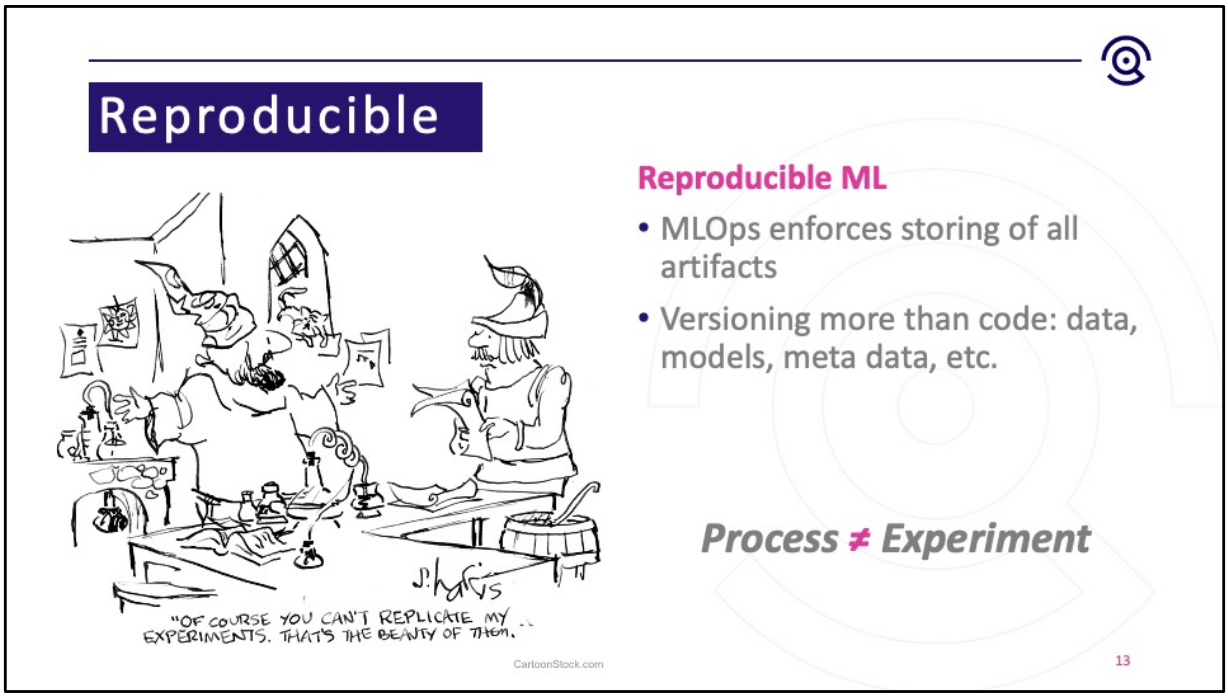
Machine Learning Operations (or MLOps) is the practice that aims to make developing and maintenance of machine learning systems in production smooth and efficient, thereby augmenting their long-term value while reducing the risk associated it with it.



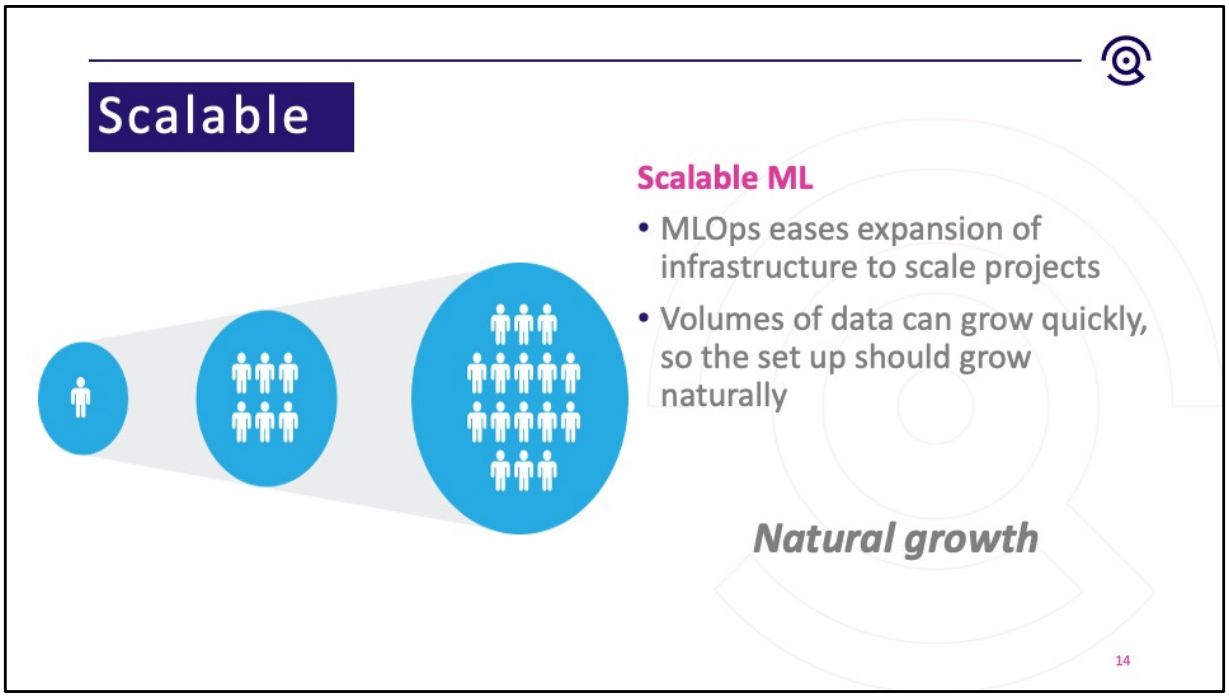
MLOps is not particularly simple to implement and the strategy adopted by any particular team will depend on their individual context. So there is no one size fits all solution. However, regardless of personal flavour, the MLOps paradigm has four main pillars as guiding principles which should always be followed. MLOps puts focus on ensuring that machine learning should be collaborative, reproducible, scalable and continuous. Let's dig a little bit deeper on each of these four principles.



It is very common for a data scientist to create ML models without takin into account all the subtleties related to operations. This might be completely fine when developing ML models, especially when creating prototypes in order to quickly validate an idea. However, it becomes a very significant problems when moving to actual production and deployment.
Under MLOps, collaboration is enforced. Ideally, it must begin from day one by making sure that everything is visible and fully auditable. MLOps ensures that all steps in the ML system are transparent, so every member of the teams can understand why and how every step is taken. Adopting MLOps eases the collaboration through the whole project life cycle as all relevant parties remain aware of everything. The high level of visibility ensures a strategic deployment of machine learning models where everyone is aware of even the most granular of detail.

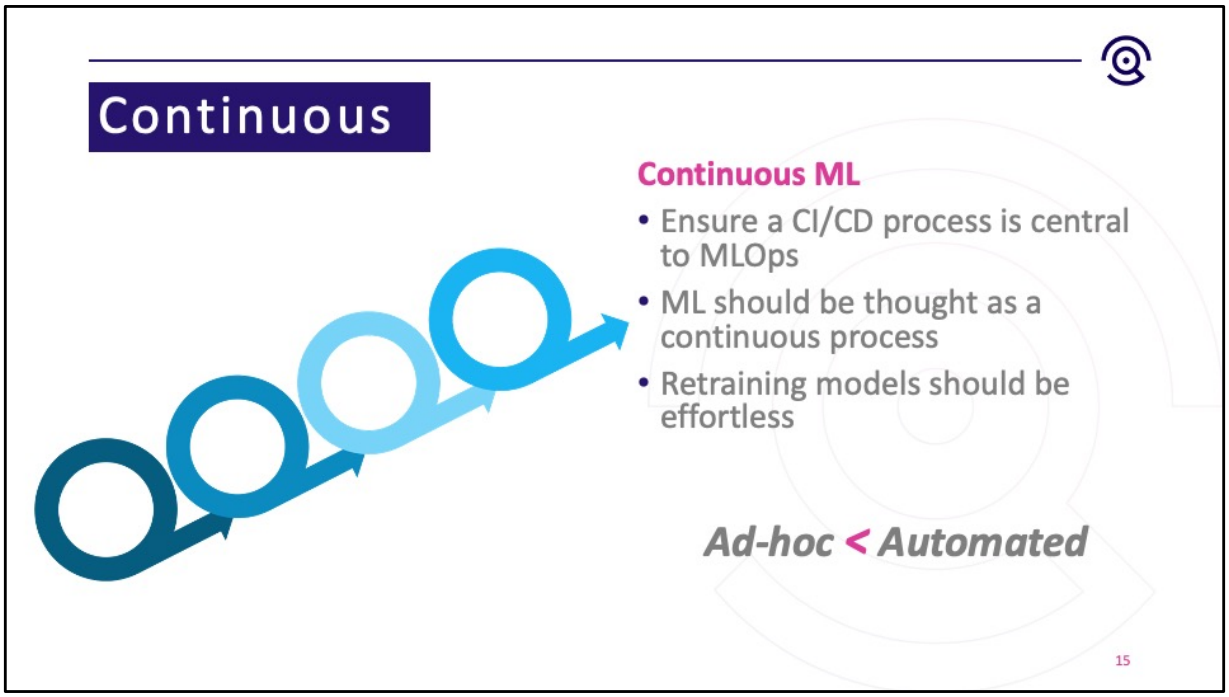


Data is not generalizable by nature, i.e., it doesn't have a one-size fits all solution model. Machine learning systems are therefore designed to be unique as they solve a very particular problem. Nonetheless, if exactly replicating the results obtained via a particular model at a particular point in time would not be feasible, then we have a problem. We need a well though out framework and some tooling to support this. There needs to be an auditable trace of the data used, the version of the framework, the code, parameters and all model artifacts. MLOps enforces the storing of all artifacts as well versioning more than just code, but also data, models, meta data, etc.
Reproducibility ensures that we are dealing with a well-defined process and not simply an experiment.



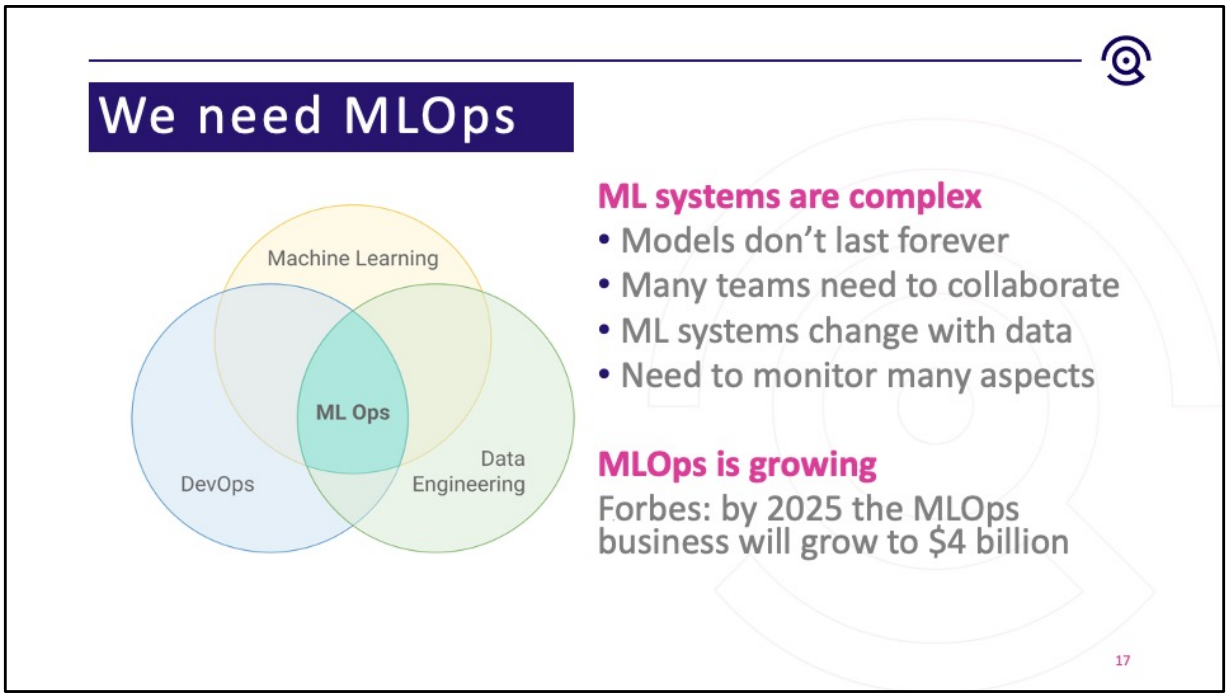
As ML systems become more complex, scalability quickly becomes an issue. This is not only endogenous to the size of the data, but also the computing power necessary to train a model on the available infrastructure. Machine learning engineers and data scientists need to have an infrastructure layer that allows them to scale work without needing to be an expert in networking. And although volumes of data can grow extremely fast, data teams need the right setup for them to grow naturally, allowing ML systems to adapt organically to these expansions.

MLOps does not only deal with the software and experimentation part of the story, but it also eases expansion of the infrastructure to scale projects while minimizing the friction.



Finally, ML should be thought of as continuous process. For MLOps to be effective, continuous improvement must be a given. Having proper CI/CD processes is one of the vital aspects of a successful MLOps strategy. As we add new code or data, automated development and testing should be triggered.
Extracting the complexity from the context, retraining ML models should be effortless in general. Each time that a modification for an existing model or even a new model is required, data scientists should not spend time on infinite ad-hoc manual tasks that could easily be automated.

Let’s discover the value of MLOps and see what good it brings us.



People often think that once a ML system is built, it can just be deployed and be operative forever. But nothing can be further from the truth. In fact, 87% of projects don't even get past the experiment phase and therefore never make it into production. One of the things that makes ML systems so complex is that they require intensive collaboration across multidisciplinary teams covering machine learning, DevOps and data engineering among others. In order for this to work, we need to ensure that good processes, tools and validation mechanisms are put in place. Putting ML models in production, operating models, and scaling use cases has been very challenging for companies due to technology sprawl and silo-ing. Similar to any other machine that is susceptible to environmental challenges, models need to be maintained and monitored to ensure they are still providing the business impact they were built to provide.

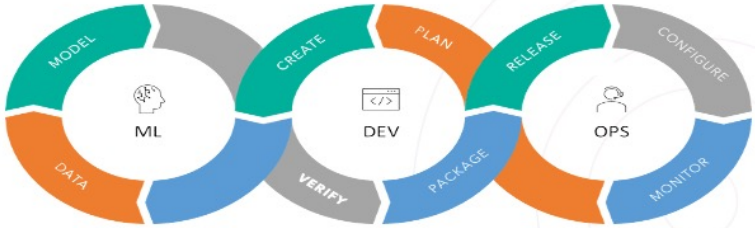
MLOps is so valuable because it does not only provide us with a coherent framework to swiftly intervene when the model's performance is decaying, but also allows organizations to build them faster and more reliably. MLOps allows organizations to have a multifaceted approach to look at the performance of ML systems by looking at them from many different perspectives. One big sign that organizations are starting to realize the value of MLOps is that according to Forbes, by 2025 the market for MLOps

is expected to reach 4 billion dollars.

Before we continue, it is worth taking a minute to stress that MLOps is not about one single software solution, nor is it a one size fits all kind of recipe either. MLOps depends of the context and each organization might need its own flavour.
MLOps is basically the combination of tools and best practices we need to set in place in order to efficiently develop and maintain ML systems.
MLOps puts focus on:

- Speeding up innovation through experimentation
- Creating reproductible workflows
- Streamlining the deployment process
- Managing the complete ML project lifecycle
  Let's dig deeper on these aspects

Speeding up innovation



How:

- Brings all development and operational teams together
- Focuses on incremental improvements in development
- Uses monitoring validation and management systems to check progress

19

At a conceptual level, MLOps' main difference with DevOps is that it allows the iterative nature of experimentation to be a part of the project life cycle. Even more, it makes experimentation systematic.

It makes collaboration possible not only for data teams, but also for analysis professionals and IT engineers. By bringing all development and operational teams together, it is easier to monitor progress and understand what is the general status of a system rather than just some of its components. This way, teams can focus on making incremental progress that is always being tracked for example. If some change happens to have unexpected consequences, it is always easier to roll back to the last working version. Moreover, through experimentation teams can keep track of what works and what does not, avoiding the replication of efforts. MLOps also increases the speed of model development and its deployment with the help of monitoring, validation and management systems for machine learning models.

Reproductible workflows

Why do we need it?

- Reduce variability across iterations
- Resiliency to failure
- Smart copy

How do we achieve it?

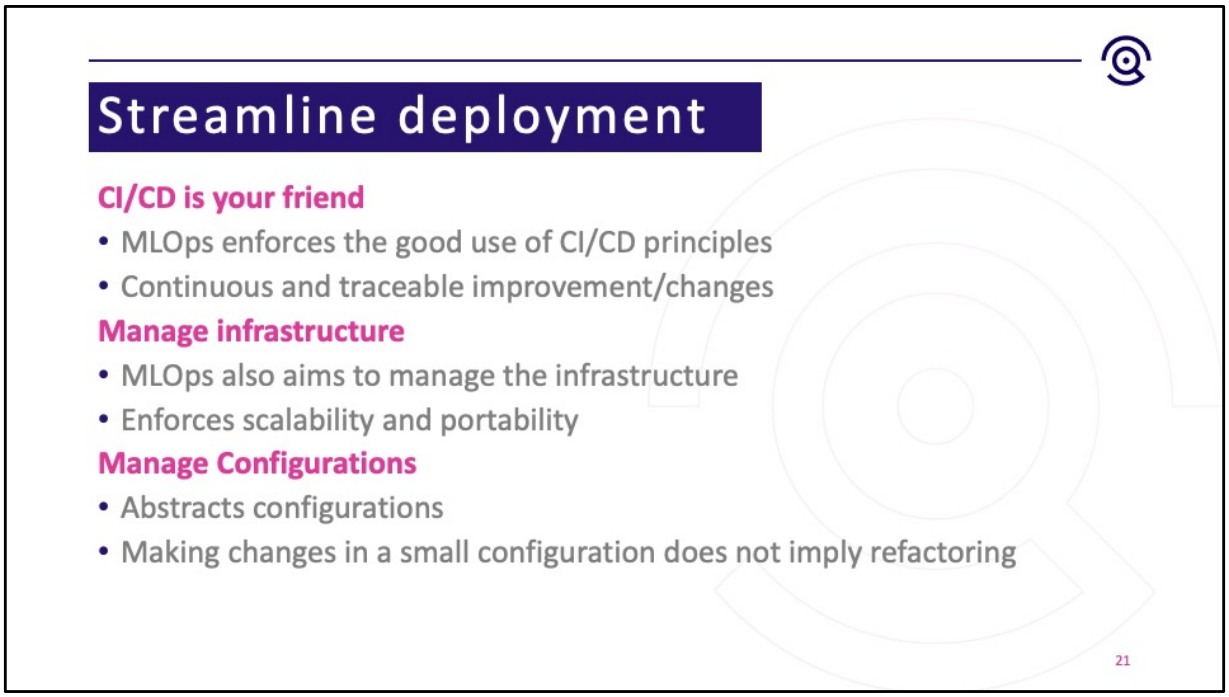
- Focus on traceability
- Enforce the use of abstractions (e.g. pipelines)
- Track resources (data and models)

20

If you ever worked on a data analytics team, in an academic setting or any experimental setting for that matter, you probably have encountered that sometimes it is very difficult to reproduce previous results. This happens the correct measures have not been taken beforehand. Reproducibility is arguably one of the most, if not the most important condition to succeed for any activity that includes experimentation. It is therefore surprising to learn that it is often the most neglected. And make no mistake, experimentation is a very central part of building any good ML system.

But why do we need reproducibility?
In the first place, it reduces variation between model iterations. It is best to make small changes and see what works and what not, instead of trying to succeed in a single trial.
Ensuring that your system is reproducible also allows you to be resilient when real-life unexpected scenarios arise. We won't be able to correct a failure if we are not able to replicate it.
But you also do not need to reinvent the wheel. If you are able to easily reproduce your systems, then you could also smart copy solutions when only small customizations are needed. So being reproducible helps portability.

But how can we achieve reproducibility?
To do this, we should make sure we have the right tools to trace all major components of our systems. This means tracking code, data, model versions, metrics, business KPI's and so on.
More on the technical side, we also need to enforce teams to use abstractions in the workflow. For example, pipelines ease the design and deployment of functionalities while also administering reproducible workflows for consistent model delivery. It does not matter if they are just data pipelines or ML pipelines.
We should also use advanced dataset registries and model registries to track our main resources.



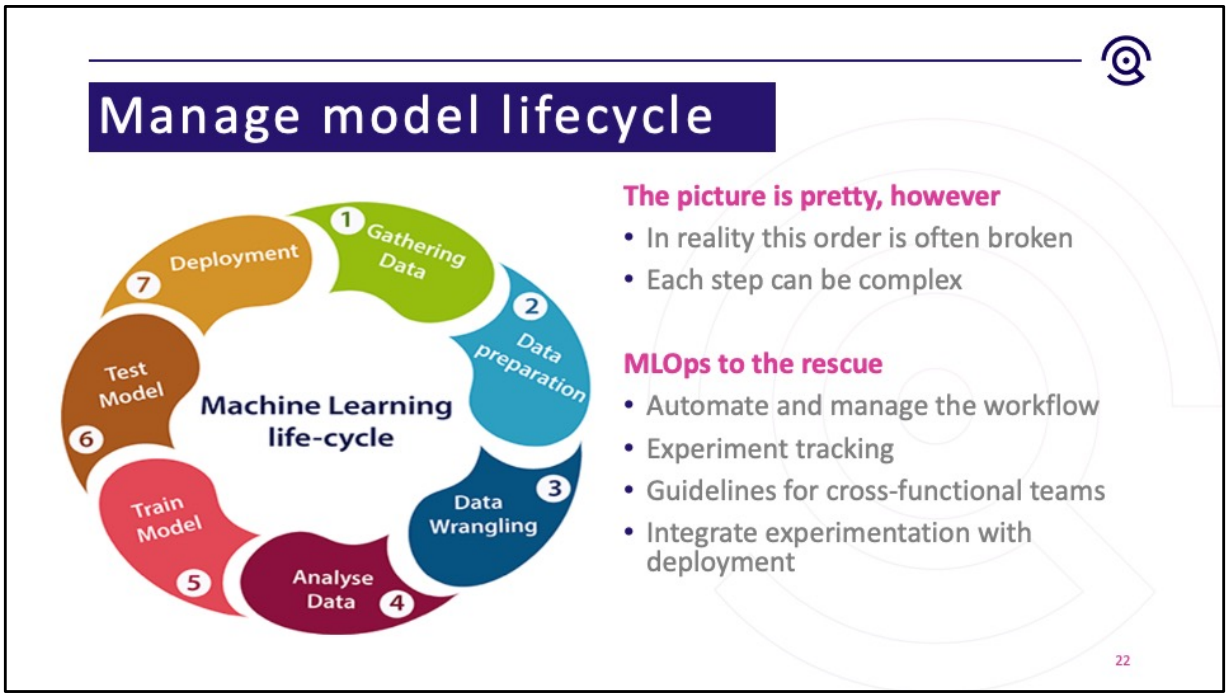
In the previous chapter we learned how we should mind the gap between a PoC and deployment, as it is not always straightforward to move ML systems to production environments. By enforcing CI/CD principles and tools, MLOps aims to smooth this often painful process.

As MLOps is an extension of DevOps, continuous deployment (when possible) is one of its main goals MLOps enforces the good use of CI/CD principles, allowing continuous and incremental improvements that can be traced.

Perhaps a bit more surprisingly, MLOps also helps managing infrastructures. It does not only focus on the ML systems, but also the infrastructure required to host them. It enforces to track how resources are used according to the problem at hand. More importantly, it also makes sure that our solutions are not heavily tool dependant and are able to scale as required while also being as portable as possible.

Last but not least , MLOps heavily focuses on managing configurations. ML systems are dependant on many parameters that have the potential to be finetuned as time goes by. More often than not, this are not properly maintained, leading to production errors that are not easy to solve. Sometimes this means that you can not simply

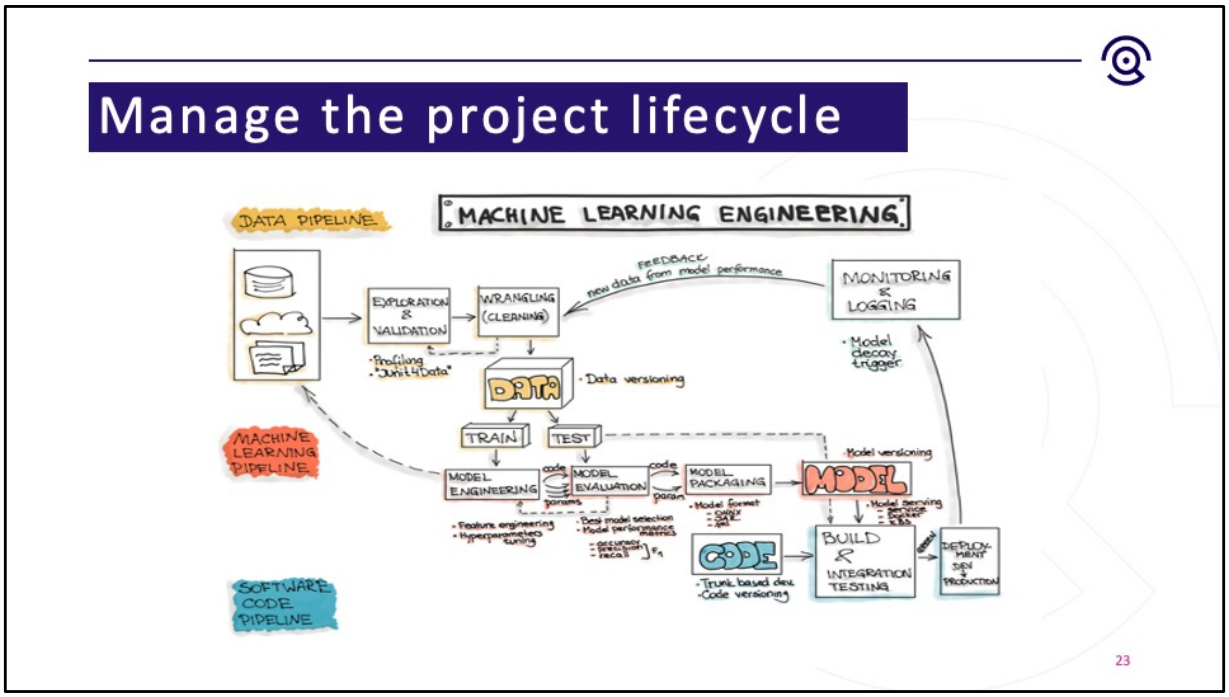
change some parameters and re-deploy, but you should make many modifications because of the hardcoding of parameters. MLOps enforces the use of configuration tools and best practices, therefore easing the deployment process.



In the ideal world, we could think of the ML model lifecycle as gathering, preparing, wrangling and analyzing data, followed by training, testing and deploying the model.
Although we wish the real world was as simple as this picture, reality is way more complicated. The order of these steps is sometimes broken because of feedback loops. Furthermore, each step can be on itself very complex with iterative processes of its own. MLOps seeks to not only streamline the deployment process (like DevOps), but also the iterative experimentation nature of the model creation process. This is done through:
- Tools and processes to automate and manage the whole workflow,
- Tools and processes for experiment tracking,
- Best practices guidelines for cross-functional teams and
- An approach to integrate the experimentation with the deployment.

MLOps mot only sets guidelines about how to go through the lifecycle, it also controls other aspects that are by-products of the processes and external requirements. Some examples are:
- keeping track of version history and model origin to enable auditing;
- evaluating the importance of features and create more advanced models with

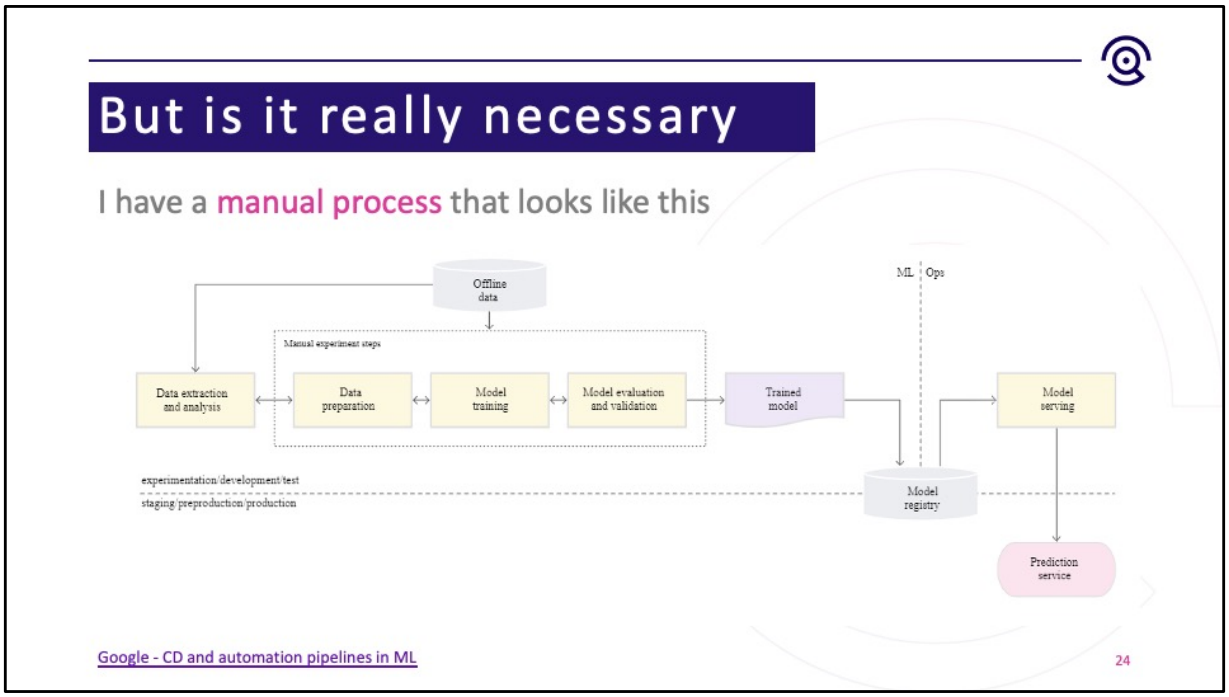
minimal bias using uniform distribution metrics;
- set calculation quotas for resources and enforce policies to ensure compliance with security, privacy, and compliance standards;
- create audit trails to meet regulatory requirements as you mark machine learning resources and automatically trace experiments.



But it is not only about the model's lifecycle, also about the whole project lifecycle.

The project lifecycle of an ML system is very complex and convoluted. It is certainly not linear. Depending on the complexity of the project, many feedback loops can arise across stages or components of the project. In the perfect world, the flow of a project would be sequential, starting from preparing the data, to train a model and then just release it in production. However, we can see in the picture that even just data preparation can be an iterative process that depends on feedback obtained from either cleaning it, training a model or even some feedback obtained after the model has been released and has interacted with end users. Bear in mind that this picture is just an over simplified version of reality. In reality, depending on the project, we could have a significant extra amount of stages and it is possible that we would need to move back and forth between them.

It becomes increasingly evident that not having the right processes and tools to manage this overly complicated lifecycle can easily lead a project astray from the target. The ulterior motive behind MLOps is to improve the way this project lifecycle is managed so no matter how complex an ML system or application is, organizations are always able to deliver valuable solutions.

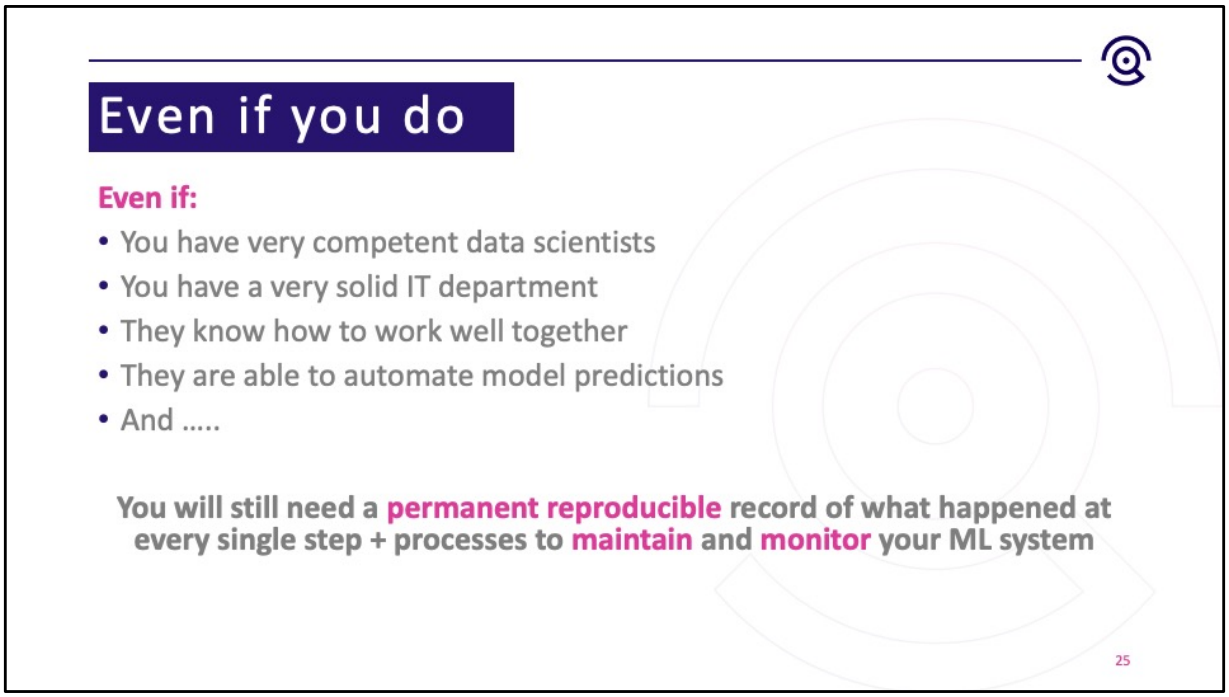


Ok, perhaps by now you are convinced that MLOps is a great framework and is super efficient. But do you really need it? After all, you might haven't been using it and so far things seems to work. Well, imagine that your situation looks like the picture. Data is extracted from a database, analyzed and prepared, after which models are trained and evaluated. Once a best model is chosen, then it is loaded to a model registry so it can be served by an application using an API. This is what Google describes as a MLOps level 0 or manual process. In fact, there are many teams that have ML experts that can build state of the art models, but they still follow this rather manual process. Although this might seem enough, it is worth noticing some issues:

- We are most likely dealing with a script-driven process,
- There is a disconnect between ML and operations,
- Release iterations are most likely infrequent and cumbersome,
- There is no CI/CD,
- Deployment only refers to prediction services, or getting predictions from a trained model, not deploying the whole ML system.
- And there is no active monitoring, so it is difficult to know the real model performance at any given time.

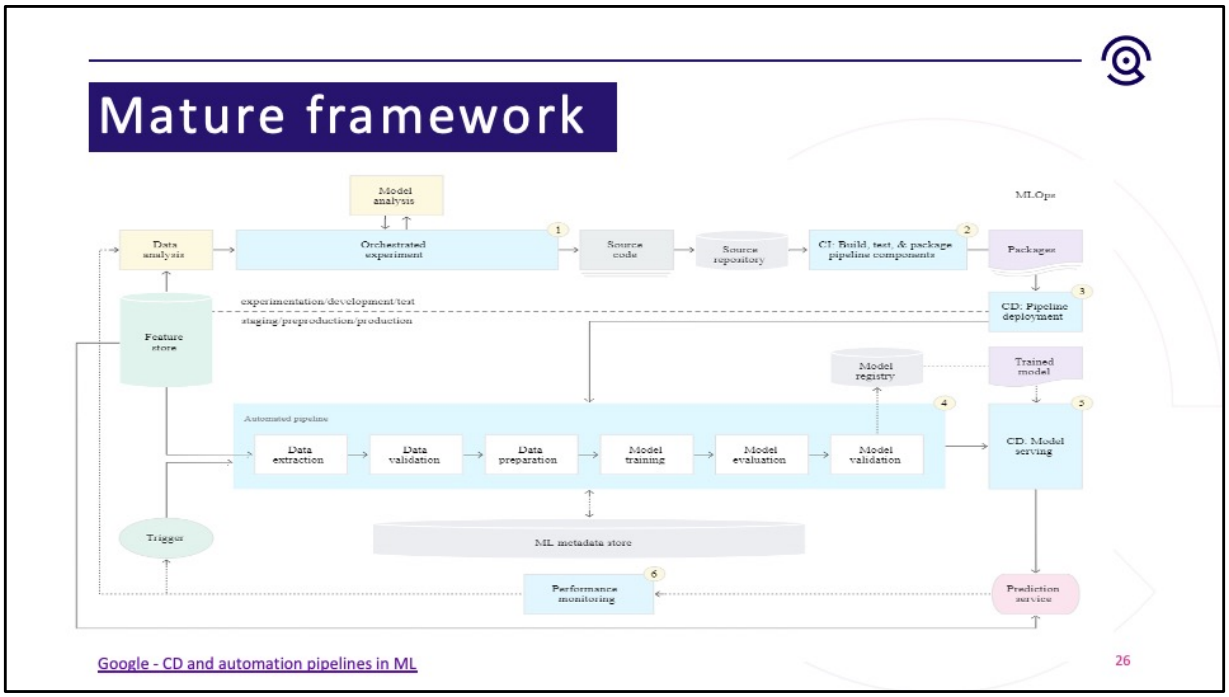
As we have seen before, there are a lot of risks that can arise from any of these

points. It would be beneficial to step up and start actively monitoring the quality of the models in production, frequently retraining the models and being able to continuously experiment with new implementations to produce new models. Even in situations where models only rarely need to be changed or retrained. In practice, models often break when they are deployed in the real world. This happens when the models fail to adapt to changes in the dynamics of the environment or changes in the data that describe the environment.



So, even if:
- You have very competent data scientists
- You have a very solid IT department
- They know how to work well together
- They are able to automate model predictions
- Or there are any other conditions that might justify this manual data scientist driven way of working.

You will still need a permanent reproducible record of what happened at every single step as well as processes to maintain and monitor your ML system. And that is only possible if the right processes and tools are used in conjunction with the models.



In the ideal case scenario, processes are going to be a bit more sophisticated and allow for a real integration between Machine Learning and Operations. In this scenario pipelines are automated and this is called MLOps level 2 by Google. The only difference with MLOps Level 1 is that level 2 also includes CI/CD. The main idea behind this set up is automating the ML pipelines so continuous delivery can be achieved. This way not only the serving of the model is automated, but also other steps like retraining and validation are done without manual intervention. This is a key aspect of MLOps practice for unifying DevOps. This allows for fast experimentation, continuous training of the model, more modular components, continuous delivery and experimental-operational symmetry. This symmetry is very important and requires that the pipeline implementation that is used in the development or experiment environment is also used in the preproduction and production environment. Notice that the data analysis and model creation (in yellow) are still manual steps created by the data scientist, but they are done in such a way that they can easily be automated when possible.

We can roughly split this flow in six components:
1. Development and experimentation: You iteratively try out new ML algorithms and models where the experiment steps are orchestrated. The

output of this stage is the source code of the ML pipeline steps that are then pushed to a source repository. This is basically the automation of the analytical work

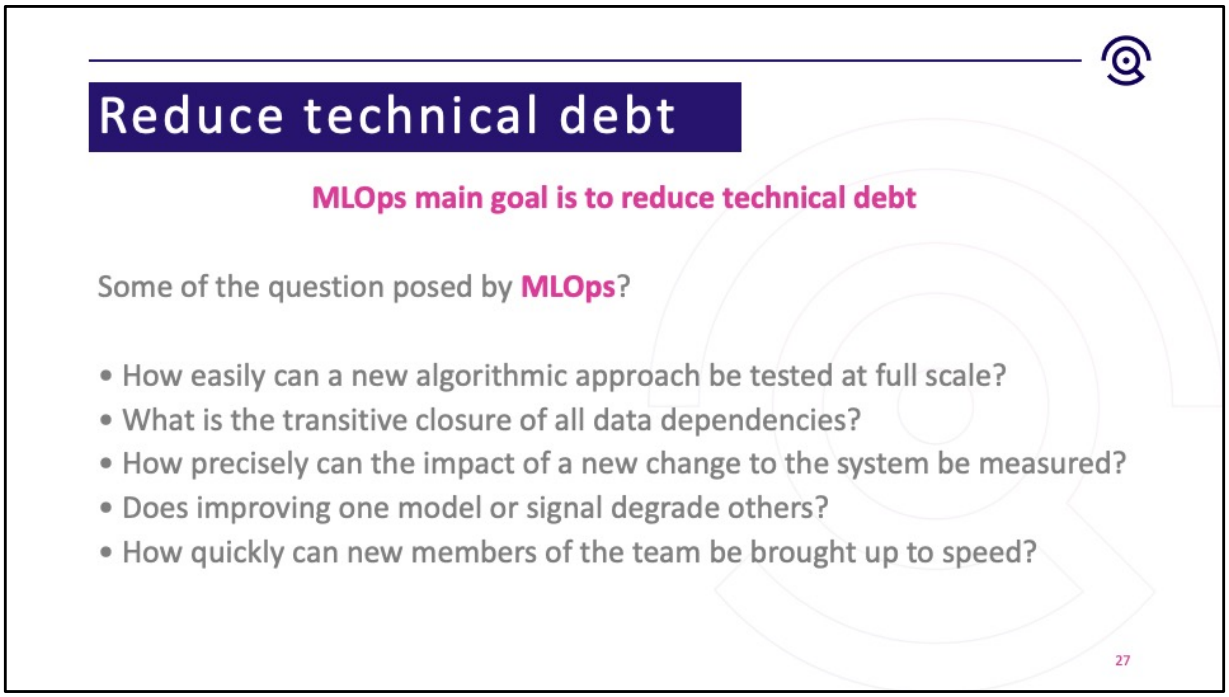
2. Pipeline continuous integration: You build source code and run various tests. The outputs of this stage are pipeline components (packages, executables, and artifacts) to be deployed in a later stage.

3. Pipeline continuous delivery: You deploy the artifacts produced by the CI stage to the target environment. The output of this stage is a deployed pipeline with the new implementation of the model.

4. Automated triggering: The pipeline is automatically executed in production based on a schedule or in response to a trigger. The output of this stage is a trained model that is pushed to the model registry. As we said before, this comes as a result from the automation of the analytical work.

5. Model continuous delivery: You serve the trained model as a prediction service to obtain the model's predictions. The output of this stage is a deployed model prediction service.

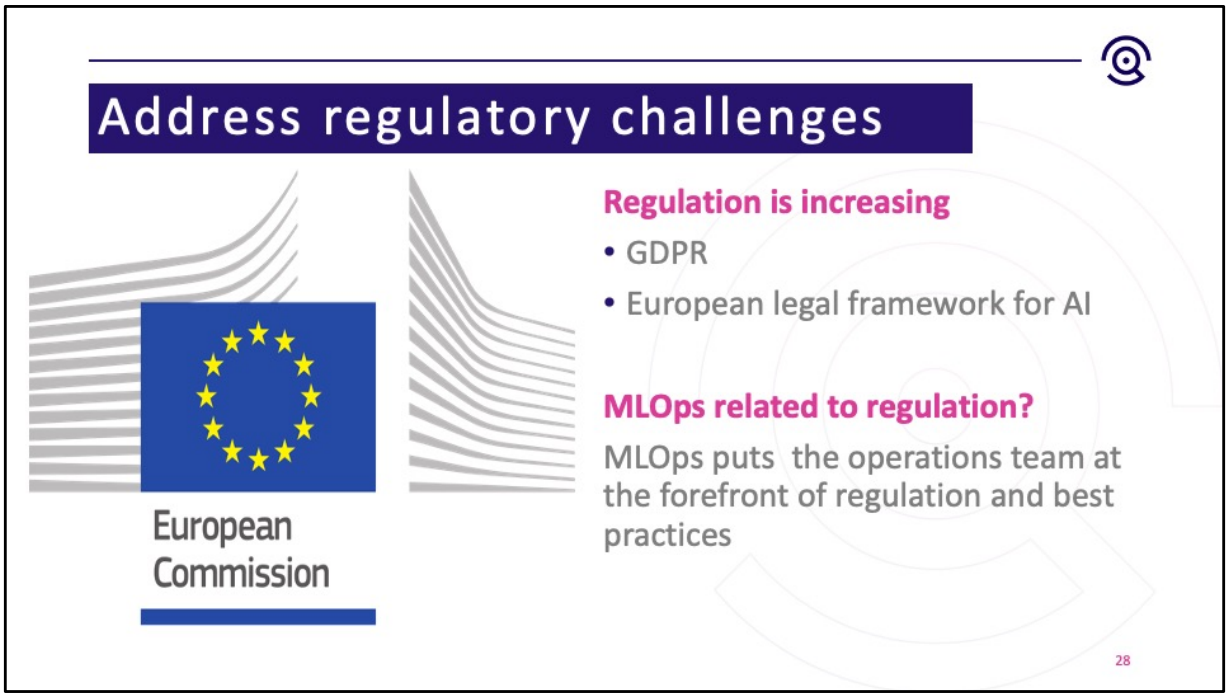
6. Monitoring: You collect statistics on the model performance based on live data. The output of this stage is a trigger to execute the pipeline or to execute a new experiment cycle.



The origins of MLOps go back to 2015 from a paper entitled "Hidden Technical Debt in Machine Learning Systems." In this paper the authors address how the technical debt of ML systems tends to stack fast over time. This is driven by the fact that ML systems tend to be developed without taking the necessary considerations for all the operational challenges that surround them. Basically MLOps was born as a framework to identify and pay technical debt, by trying to give an answer to questions like:

- How easily can an entirely new algorithmic approach be tested at full scale?
- What is the transitive closure of all data dependencies?
- How precisely can the impact of a new change to the system be measured?
- Does improving one model or signal degrade others?
- How quickly can new members of the team be brought up to speed?

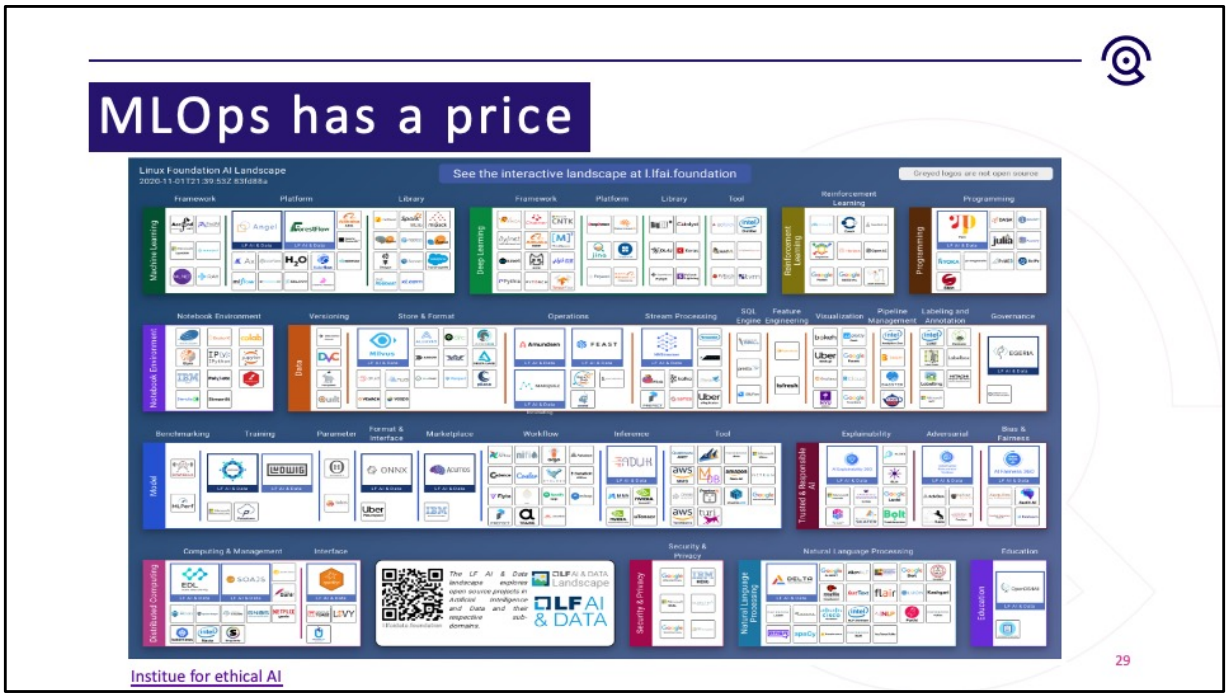
This is immensely valuable as having a low technical debt allows ML systems not only to work smoothly, but also to become more maintainable, extendable and even more accurate. Having the proper MLOps tools and processes in place is the best way to guarantee that the technical debt of our ML systems does not grow out of control and that our models do not end up being useless.



The amount of regulation for AI seems to be progressively increasing. In Europe it is no longer only about privacy and data, which are already covered by GDPR, but also about how AI models affect our day to day life. The European Commission has presented a European legal framework for AI to address fundamental rights and safety risks specific to the AI systems. The Commission aims to address the risks generated by specific uses of AI through a set of complementary, proportionate and flexible rules. These rules will also provide Europe with a leading role in setting the global gold standard, probably similarly to GDPR, which other countries will soon follow with their own initiatives.

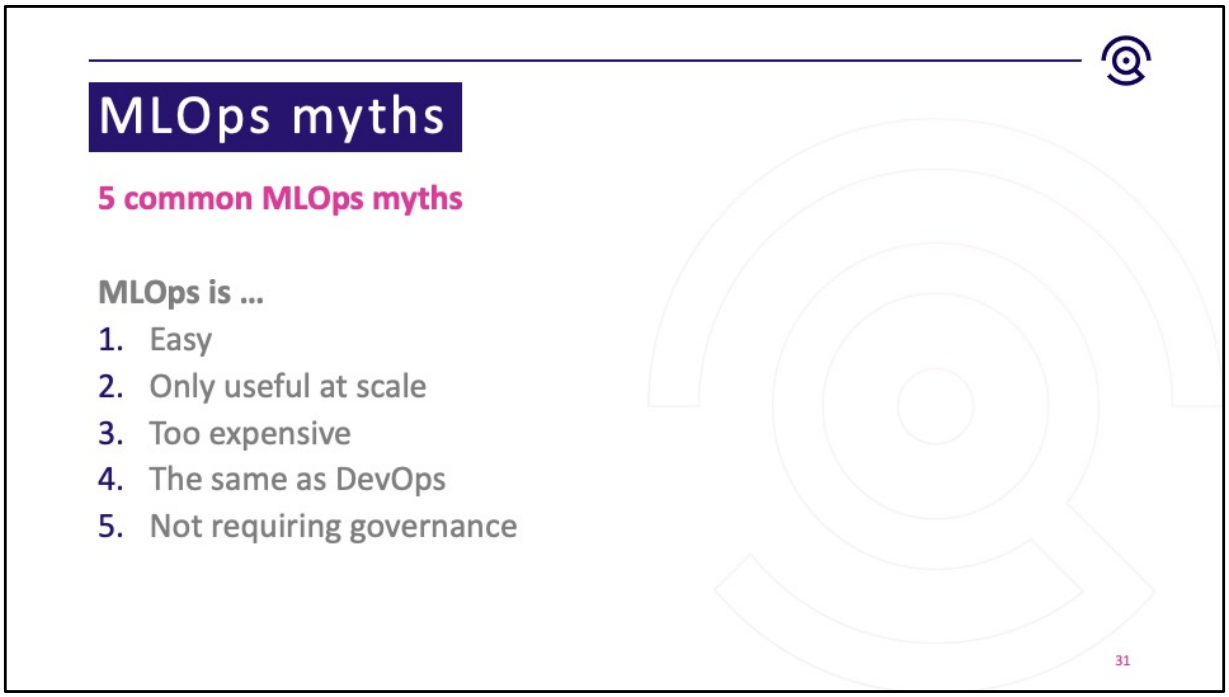
Having this in mind, the more controlled your creative and operational processes are, the more auditable they become. Transparency is a direct by-product of the whole governance that has been placed around the ML system, from experimentation all the way to deployment. It is for this reason that not having a framework to guide the project lifecycle of ML systems can put all the resources that were invested in their creation in jeopardy. By adopting a consistent MLOps strategy, it is possible to get ahead of regulation by making your models more robust and the processes around it more transparent and therefore auditable. MLOps puts the operations team at the

forefront of regulation and best practices.



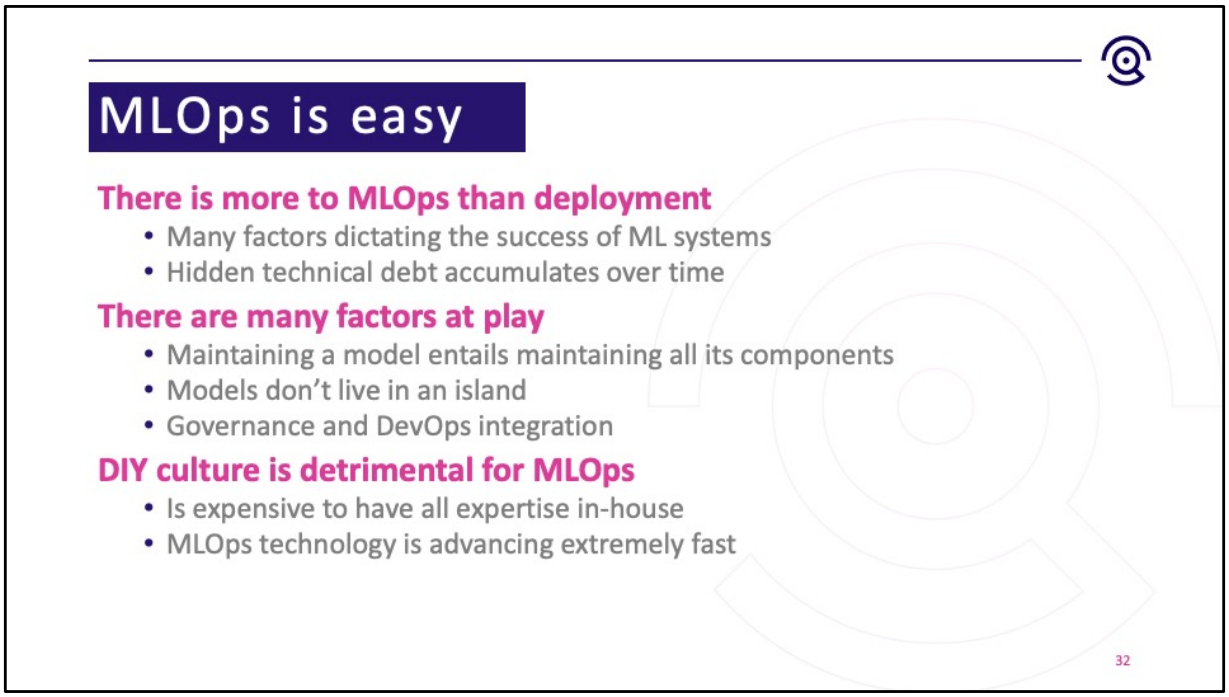
It is true that having a solid MLOps framework in place will increase the quality of your ML systems and allow you to grow your ML applications organically. However, it is also true that it does not come without a cost. Only having a quick look at the picture of a curated list of MLOps and AI tools, put together by the Linux Foundation AI project, we can quickly realize that there are many tools at our disposal and not all of them are as affordable. And even if they are, there is a search cost related to choosing the right tool for the right thing. Moreover, because of the fast pace of growth that the field is experimenting, the landscape changes continuously. The main advice we can give you is to spend some resources building MLOps expertise and that you keep in mind that it is not only about the tooling but also about the processes.

Let’s demystify some of the common MLOps myths.



MLOps is relatively new, so it is natural that there are still some misunderstandings about what it entitles. In this section we are going to review five of the most common myths around MLOps and go a bit deeper about why you should not believe them. These are the five myths:

1. MLOps is easy
2. MLOps is only useful at scale
3. MLOps is too expensive
4. MLOps is the same as DevOps
5. MLOps does not require governance



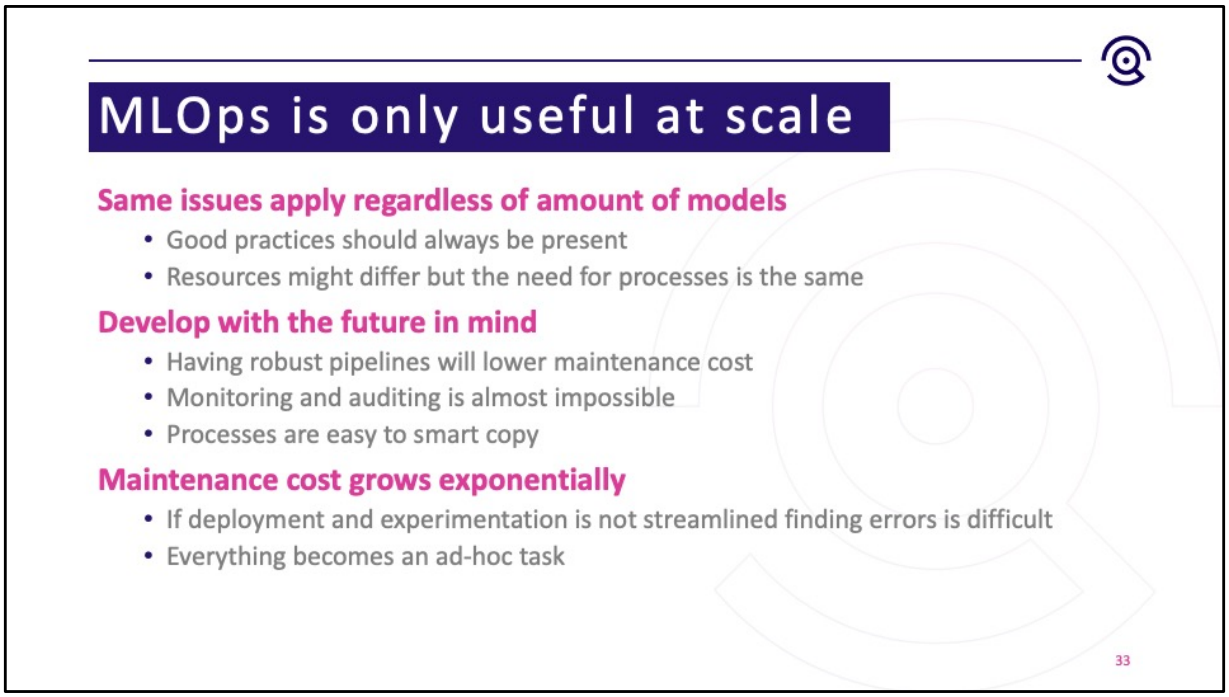
If you hear somebody saying that MLOps is easy, then this is probably because of a lack of understanding on what MLOps really is about and how complex ML systems can get.

There is much more to MLOps than just deployment and there are more factors dictating the success of an ML system than just deployment. When taken altogether, they can quickly become too complicated for most teams to manage on their own. And even if they could manage, it wouldn't be worth the opportunity cost to do so. This would automatically lead to huge hidden technical debt accumulating over time until it becomes unmanageable.

There are many factors at play in ML systems. Maintaining a model entails maintaining all its components, and each of them can be very complex on their own requiring the right expertise that is not always available. Moreover, models don't live on an island. So even if they are state of the art models, the systems that support them also play a huge role. That is without counting that governance and DevOps integration is often neglected, which again, makes future maintenance unfeasible.

Another less obvious dimension to this problem, is that the Do It Yourself or DIY

culture is generally detrimental for MLOps. It is expensive to have all expertise in-house and sometimes it might be cheaper or more efficient to buy than to build. MLOps technology is advancing extremely fast, and integration is becoming less of a problem as flexibility is a big part of the MLOps culture. This might seem paradoxical because it is often the resilient commitment to an entrepreneurial do-it-yourself spirit that launches a business into the initial strata of success. However, there is some evidence that those organizations who outsource their MLOps see the best results on average.



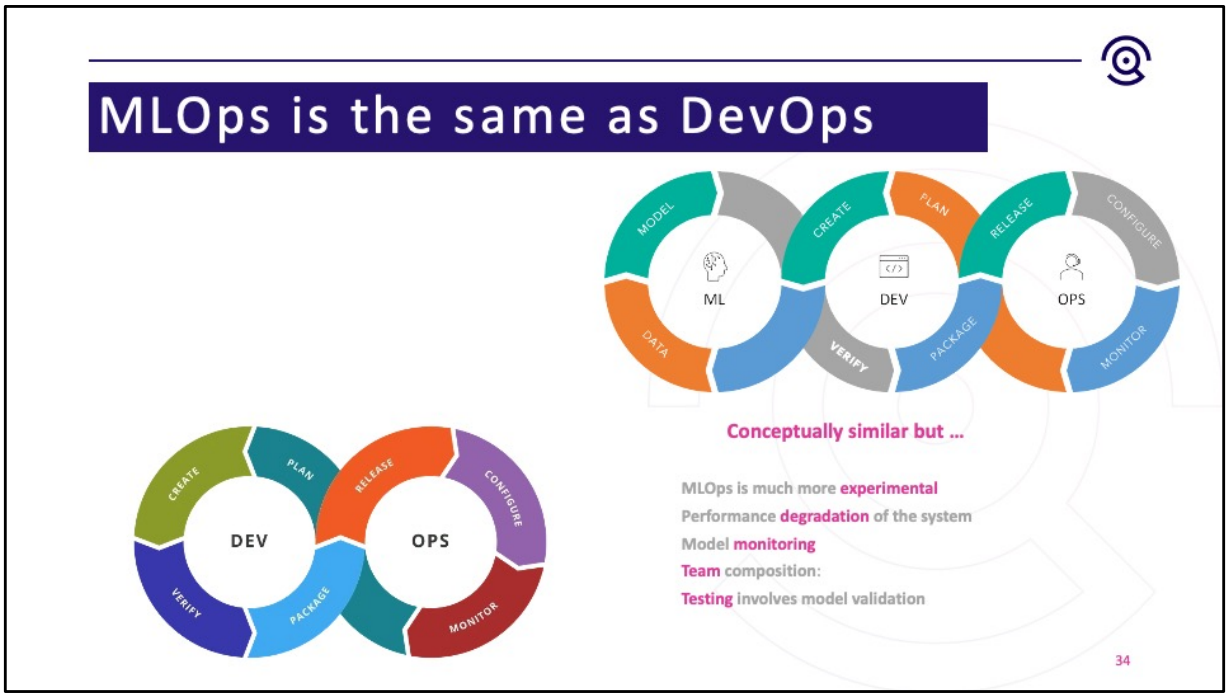
Perhaps the most common of the myths is that MLOps only works, or is only viable, when developing a ton of models. But this is actually very far from the truth.

All the same issues apply regardless of your amount of models. And even then, good practices should always be present if we want solutions to be robust. It is true that the amount of resources required might differ, but the need for well-defined processes remains the same, no matter the scale.

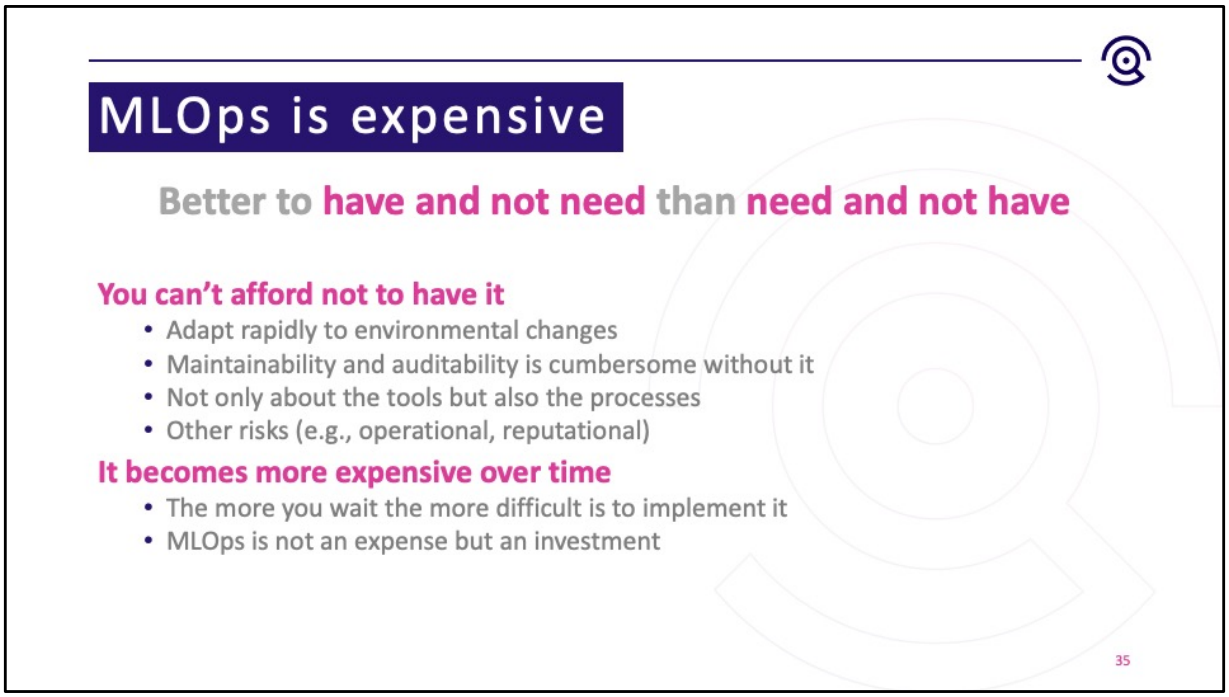
More importantly, you should develop with the future in mind and avoid having a short sight when dealing with ML systems. Having robust pipelines will lower maintenance cost and save you tons of headaches in the future. Even if you don't have an enormous amount of models, monitoring and auditing is almost impossible if there is no systematic approach to developing them. And in case there are opportunities to grow, well-defined processes are easy to smart copy so scaling becomes easier.

Finally, it is good to bear in mind that maintenance cost grow exponentially. If deployment and experimentation are not streamlined, then finding errors is difficult and sometimes you can't afford big downtimes because it can disrupt your operation

or damage your reputation and brand value. Avoid everything from becoming an ad-hoc task, because imagine you would need to spend most of the time of your data scientists in maintaining old applications instead of innovating. That would not be great and result in a lot of missed opportunities.



As we have covered before in this course, although MLOps and DevOps are conceptually similar, the end goal of MLOps has a different and wider scope. Unlike DevOps, MLOps is not only concerned with code, but also with models and data, plus all the extra concerns that this entails. Iteration cycles in MLOps are longer and more chaotic as they need to allow for experimentation with many sources of feedback. Common examples are model parameters, changes in the data or feedback from end users, all modifying original scope of the system. Moreover, ML systems are impacted by changing data profiles. This is not the case in a traditional IT system. Hence, the model might have to be refreshed even if it 'works' currently, which leads to them having to be constantly monitored. Another important difference is that the team needed to build and deploy models in production may not always comprise software engineers. For example, the members working with exploratory data analysis like business experts or data science experts are often not well versed in software engineering. This adds an additional level of complexity as integration tends to be more difficult and MLOps should account for this. Finally, in addition to the software tests such as unit testing or integration tests, testing an ML system also involves model validation, model training and data lineage. All these aspects are not always present in typical software development, requiring MLOps to be more extensive than DevOps.

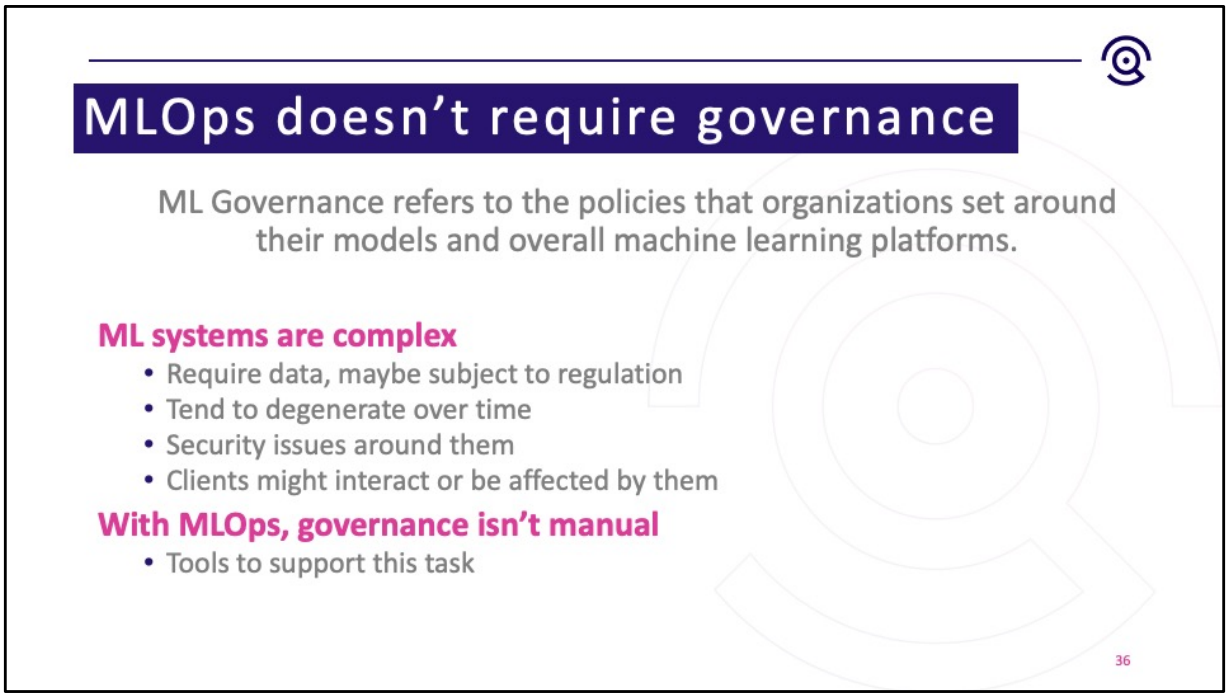


Although there are additional monetary cost and resources necessary to apply MLOps principles and tools, having an MLOps strategy is similar to having an insurance. It is better to have it and not need it, than to need it and not have it. With the small added caveat that, almost certainly, you are going to need it.

In all fairness, you actually can't afford not to have it. Especially when your models are using data that can quickly change because of environmental reasons or the ML systems need to adapt to external conditions timely. And even if this is not the case, because your data is rather static, lacking the necessary processes and tools around your ML systems will most likely translate in big costs when it comes to maintain or audit your models. Remember, MLOps is not about fancy tools but about clean and neat processes enforcing robustness and reproducibility that can be supported by the use of such tools. Perhaps the most important thing to consider is that sometimes the failure of ML systems can translate in different types of risk, from operational to reputational. The more difficult it is for you to mitigate those risks, they higher the probability of realization of those risk becomes.

And if you think MLOps now is expensive, it is almost certain that not having an MLOps strategy will become more expensive over time. The more you wait, the more

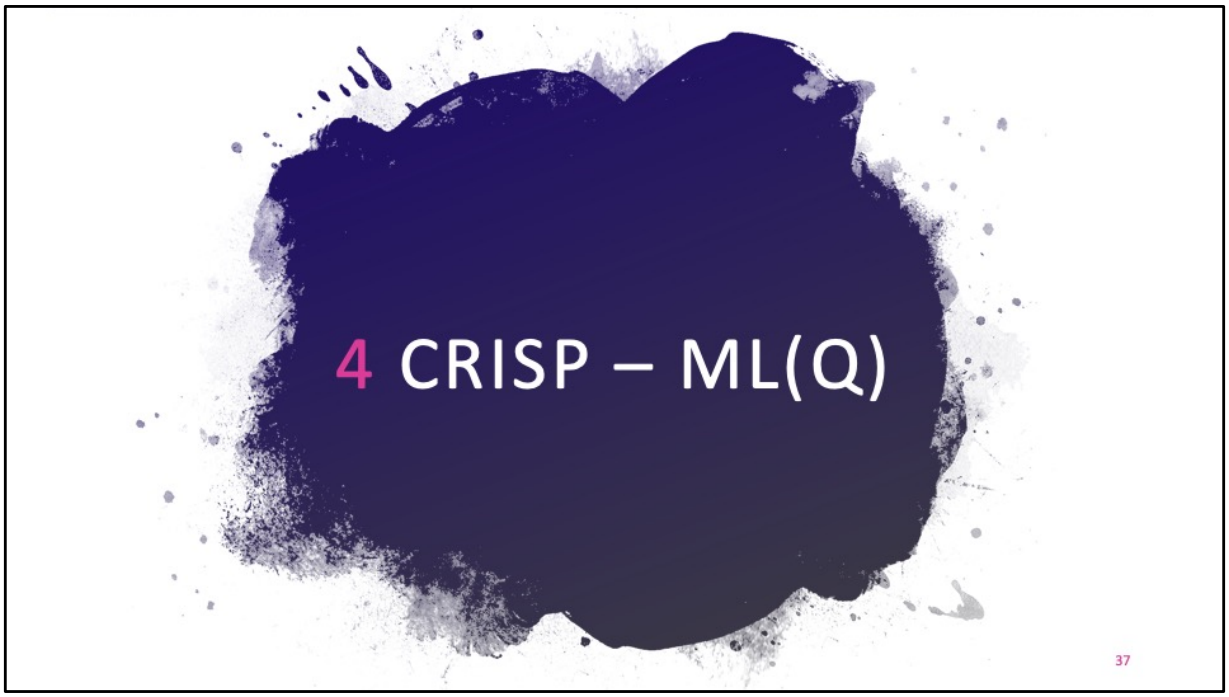
difficult a proper MLOps implementation will become. You should not see MLOps as an expense, but as an investment to ensure the well functioning of your ML systems in production.



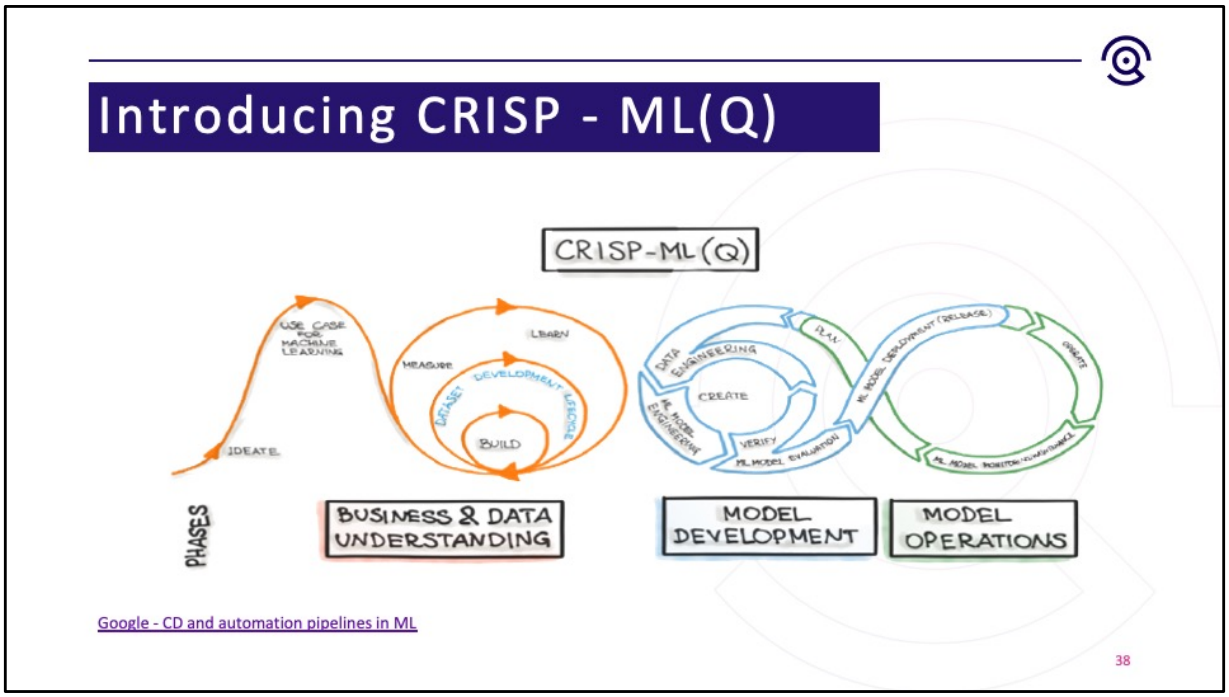
The final myth we are going to discuss is that of MLOps not requiring governance. In terms of ML, governance refers to the policies that organizations set around their models and overall machine learning platforms. But if you have been paying close attention, you would realize by now that this is a shaky claim at best.

ML systems can be extremely complex. They required data and data might be subject to regulation. Not being able to understand how your data is used along the process might lead to regulatory issues and even translate to fines. Also, models tend to degenerate over time, so we should be ready to intervene with the least amount of overhead possible. On top of that, there are security issues surrounding the model which could be related to authentication or regarding confidential data. Without proper governance it is difficult to set an environment in which those security issues are properly addressed. Additionally, clients might interact or be affected by the ML systems. Proper governance ensures that the impact that a failure of an ML system can have on the clients is mitigated and avoided in the future.

Luckily for you, with MLOps governance isn't manual. There are tools to support this task, thereby avoiding the expense of having to continuously dedicate valuable human resources to this task.



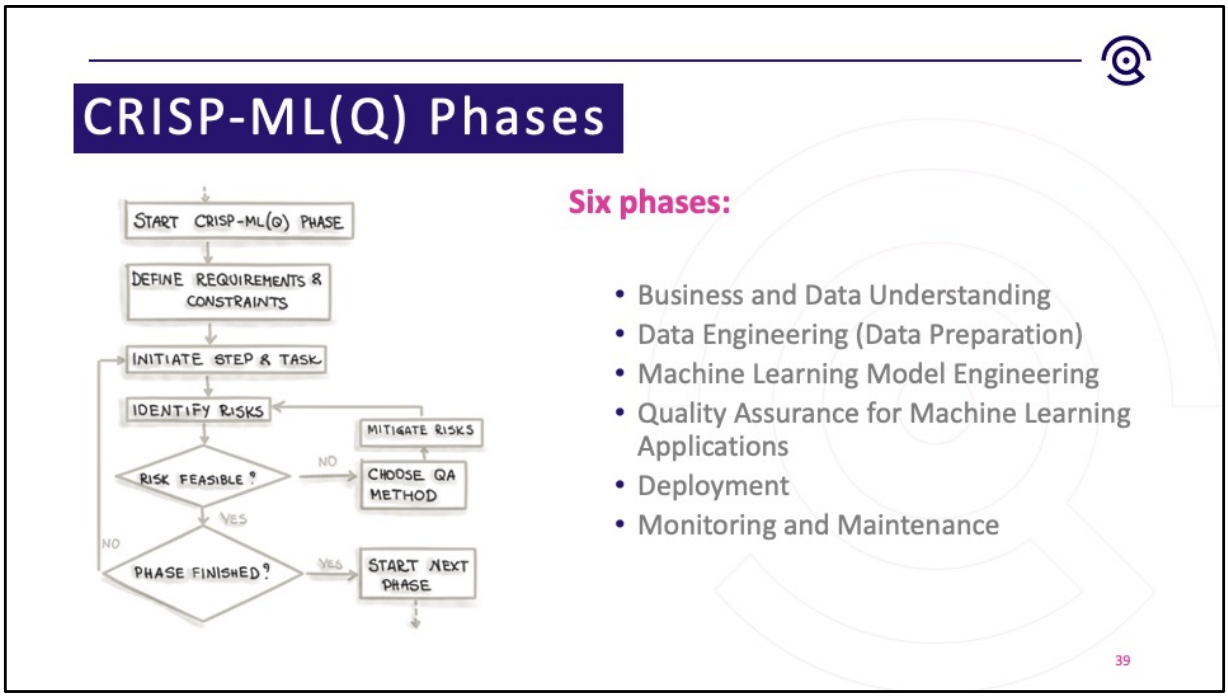
Let’s discuss the process model CRISP ML(Q) for the development of machine learning applications.



Since 1996, CRISP-DM (or the CRoss-Industry Standard Process for Data Mining) has been the open standard model that has been used by most data mining professionals to build analytical solutions. Since there was no similar process available for ML, CRISP-DM has become the standard solution for most teams developing ML systems as well. However, this procedural model is ill-equipped to deal with the complexities arising during the implementation of these ML systems. There are 2 major shortcomings related to this model.

First, CRISP-DM puts focus on data mining and does not cover the application scenario of ML models that are inferring real-time decisions over a long period of time. The ML model has to be adaptable to a changing environment or the model's performance will degrade over time, making a permanent monitoring and maintenance of the ML model a requirement after the deployment. Second, and maybe more worrying, CRISP-DM lacks guidance on a quality assurance methodology. For these reasons, a new process model CRISP-ML(Q) for the development of machine learning applications has recently been proposed in a paper by Studer et al. This new procedural model covers six phases from defining the scope to maintaining the deployed machine learning application. In this section we are going to discuss how adopting the right process model for the construction of ML systems can have a

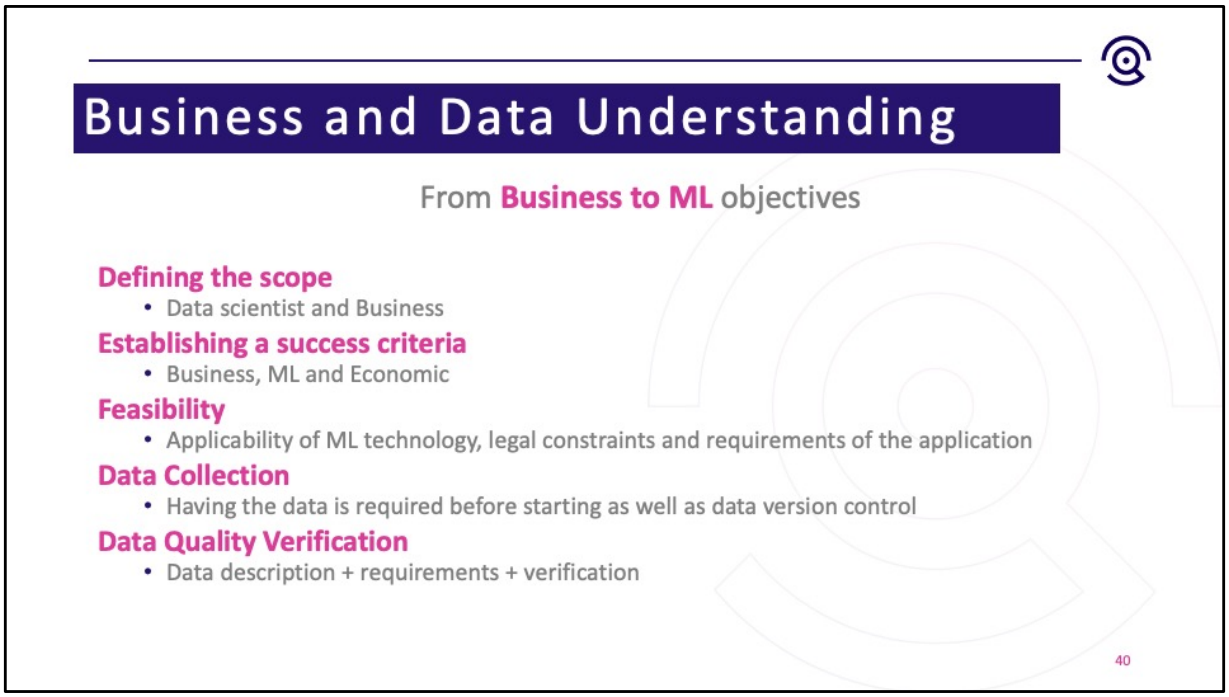
major impact on the their quality and bring your MLOps journey to the next level.



The CRISP-ML(Q) model describes six phases:
- Business and Data Understanding
- Data Engineering (or Data Preparation)
- Machine Learning Model Engineering
- Quality Assurance for Machine Learning Applications
- Deployment
- Monitoring and Maintenance

As shown in the flow diagram, for each phase of the process model, the quality assurance approach in CRISP-ML(Q) requires the definition of the following elements:
- requirements and constraints (e.g., performance, data quality requirements, model robustness, etc.)
- instantiation of the specific tasks (e.g., ML algorithm selection, model training, etc.)
- specification of risks that might negatively impact the efficiency and success of the ML application (e.g., bias, overfitting, lack of reproducibility, etc.)
- quality assurance methods to mitigate risks when these risks need to be diminished (e.g., cross-validation, documenting process and results, etc.).

The idea is to establish a robust quality assurance methodology to determine whether these tasks were performed according to current best practice standards from the industry and academic literature, which have proven to be general and stable and are suitable to mitigate the task-specific risks. Let's dig deeper on each of the six phases.

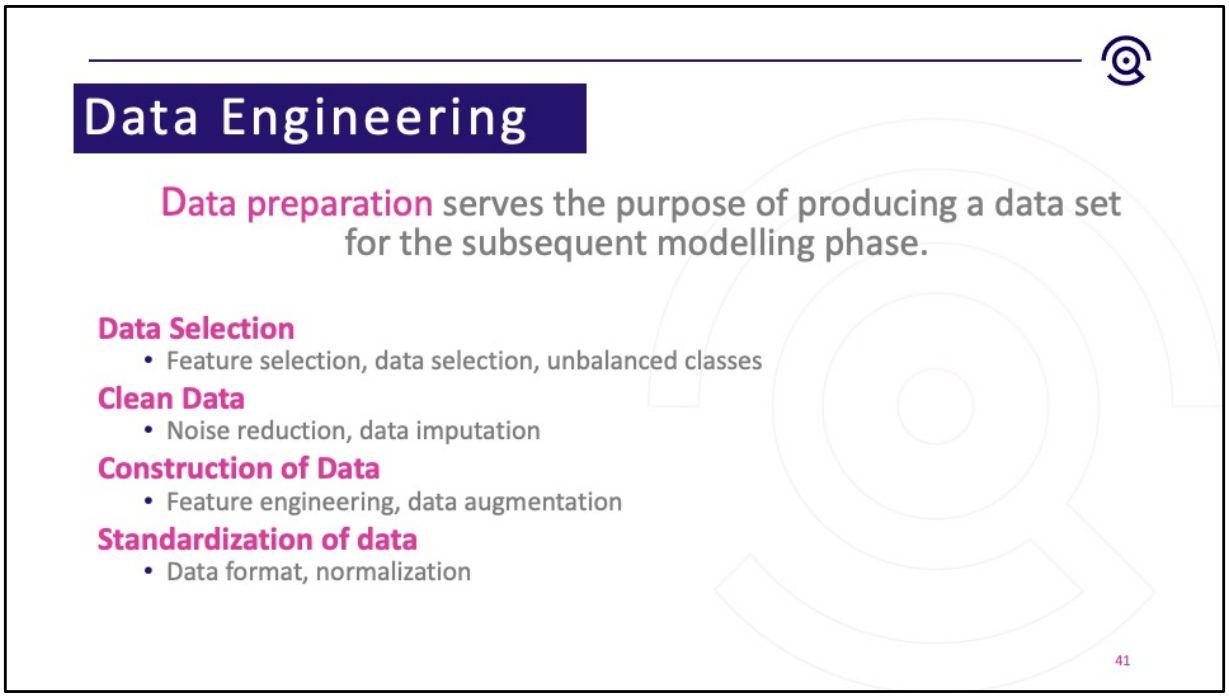


The initial phase is concerned with defining the business objectives and translating them into ML objectives, but also to collect data, verify the data quality and to finally assess the project's feasibility.

- It should start by defining the scope of the project, which is a joint venture between data scientists and business or domain experts. Having only the data scientist doing the scoping carries the risk of not satisfying the business requirements.
- Once the scope is ready, measurable success criteria should be defined at three levels: Business, ML and Economic. These success criteria need to be defined in alignment to each other and with respect to the overall system requirements to prevent contradictory objectives.
- Checking the feasibility before setting up the project is considered best practice for the overall success of the ML approach. This can greatly minimize the risk of premature failures due to false expectations. Among the things to check are the applicability of the ML technology, legal constraints and the requirements of the application. If feasibility is not evaluated positively, stopping or putting the project on hold is the best course of action.
- Costs and time are needed to collect a sufficient amount of consistent data by preparing and merging data from different sources and different formats. An ML

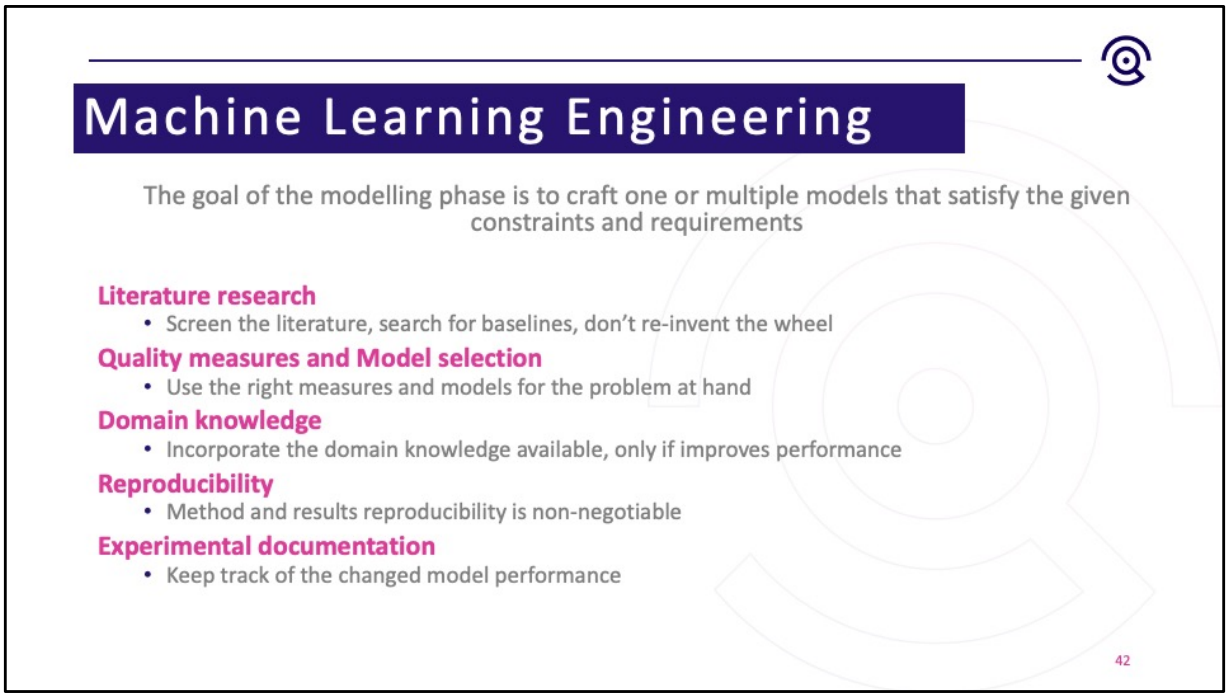
project might be delayed until the data is collected or could even be stopped if it is impossible to collect data of sufficient quality. Moreover, as data is generally not static, all changes on the data should be traced.

- The data quality verification should be done in 3 steps, namely data description, definition of data requirements and verifying that the data complies with the requirements.



The data engineering or data preparation step serves the purpose of producing a data set for the subsequent modelling phase. The most important part of this step is the documentation of all transformations the data goes through as lineage is key in tracking data related errors. For example, when the modelling or the deployment phase reveals erroneous data. To path the way towards the ML lifecycle in a later phase, methods for data preparation that are suitable for automation are preferable. These methods can be grouped in 4 stages:

- Data Selection, e.g., feature selection, data selection and dealing with unbalanced classes.
- Clean Data, e.g., noise reduction and data imputation.
- Construction of Data, e.g., feature engineering and data augmentation.
- Standardization of data, e.g., changing the data format and normalization.

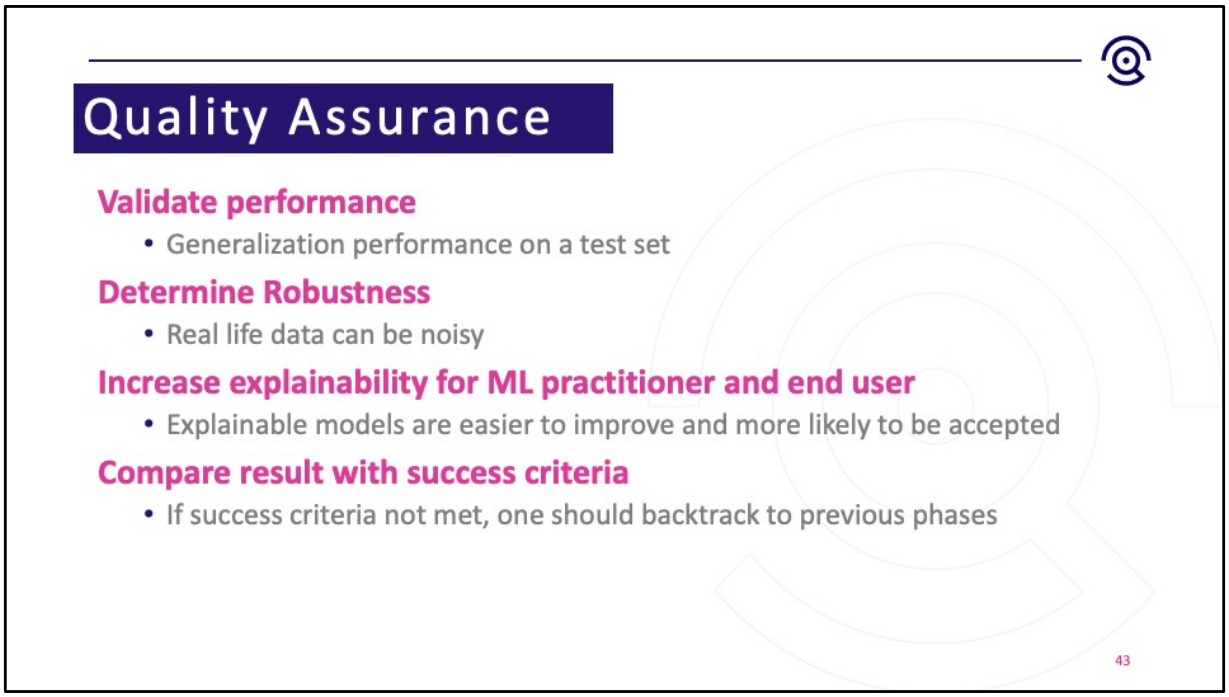


The choice of modelling techniques depends on the ML and the business objectives, the data and the boundary conditions of the project to which the ML application is contributing. The goal of the modelling phase is to craft one or multiple models that satisfy the given constraints and requirements. Some general tips & tricks are the following:

- Make sure to do a literature research on similar problems. It is best practice to screen the literature (e.g., publications, patents or internal reports) for a comprehensive overview on similar ML tasks. ML has become an established tool for a wide number of applications. New models can be based on published insights and previous results can serve as performance baselines. Do not re-invent the wheel.
- Quality metrics should be decided ex-ante. Use the right performance metric for the right business case. Besides a performance metric, soft measures such as robustness, explainability, scalability, resource demand and model complexity have to be evaluated also. The model selection depends on the data and has to be tailored to the problem. There is no such model that performs the best on all problem classes.
- It is desirable to incorporate extra domain knowledge to your models when available. However, it is also best practice to validate the incorporated domain

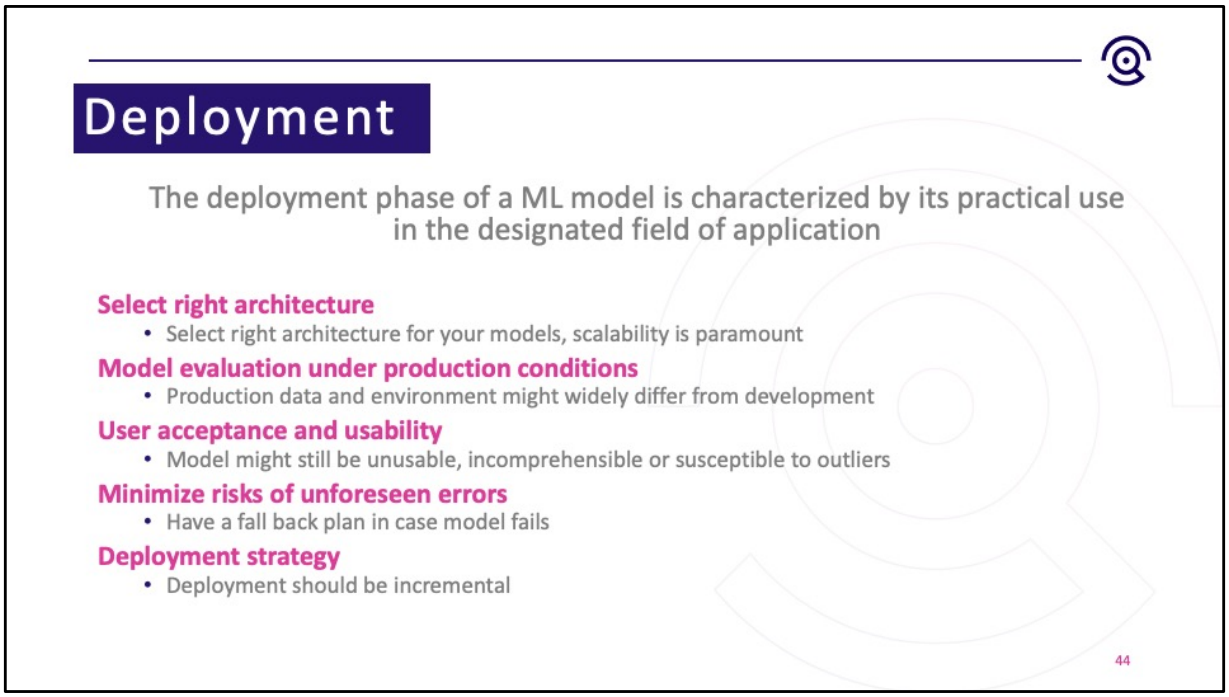
knowledge in isolation against a baseline. Adding domain knowledge should always increase the quality of the model, otherwise, it should be removed to avoid false bias.

- An important focal point of MLOps is reproducibility. Ensuring results can be replicated is fundamental in the construction of a healthy ML system. Both methods and results should always be reproducible, regardless of the context.
- Finally, keeping track of the changes in model's performance and its causes via model modifications is important. This allows model comprehension by addressing which modifications were beneficial and improve the overall model quality.



Making sure the quality of the model is up to par with the business expectations is key. If the model and data are of no quality, then we will be in a garbage in garbage out scenario, rendering the deployment useless. Assuring proper quality involves the following elements:

- As a first step we should validate the performance on a test set from which we are comfortable generalizing to real cases. This validation should be extensive and methodical.
- A major risk occurs when ML applications are not robust to perturbed, e.g. noisy or wrong, or even designed adversarial input data. Models should not be overly sensitive to this noise. Test to determine the robustness of the model should be performed.
- Explainability of a model helps to find errors and simplifies the search for strategies to improve the overall performance. In practice, in case of structured input data with meaningful features, inherently interpretable models are not necessarily inferior to complex models. Moreover, case studies have shown that explainability helps to increase trust and users' acceptance.
- It is best practice to document the results of the evaluation phase and compare them to the business and ML success criteria we have set. If the success criteria are not met, one might backtrack to earlier phases.



The deployment phase of a ML model is characterized by its practical use in the designated field of application. Deployment can be challenging because this is where ML, software engineering and operations are confronted with each other. It is estimated that only around 1 out of every 10 models that are developed are successfully deployed. To ensure success we should take into account the following aspects:

- Start by selecting the right architecture. Many factors come into play here, and sometimes this is not only a matter of choice but also budget, availability and legacy systems. Regardless of these challenges, it is of upmost importance that we try to do our best to optimize our resources in order to select the right architecture for the right problem. For example, the resources required for a simple logistic regression that operates in batch (e.g., once a day) are very unlikely to be the same as those of a deep neural network that needs to be served in real time. Another example is that while cloud services offer scalable computation resources, embedded systems have hard constraints, and the choice to go for one or another is not always possible. It is also important that we think about scalability, as it is a common problem facing ML systems.
- Keep in mind that model evaluation under production conditions might not be identical to the development conditions. For example,

production data and the environment might widely differ from development. We should test

the model under increasingly similar conditions, but do it incrementally.

On each incremental

step, the model has to be calibrated to the deployed hardware and the test environment. This allows identifying wrong assumptions on the deployed environment and the causes of model degradation.

- User acceptance and usability is not always a

given, even after developing a seemingly good model. The model might be incomprehensible and/or does not cover corner cases. It is best practice to build a prototype and run a field test with end users. Make sure that the user understand all

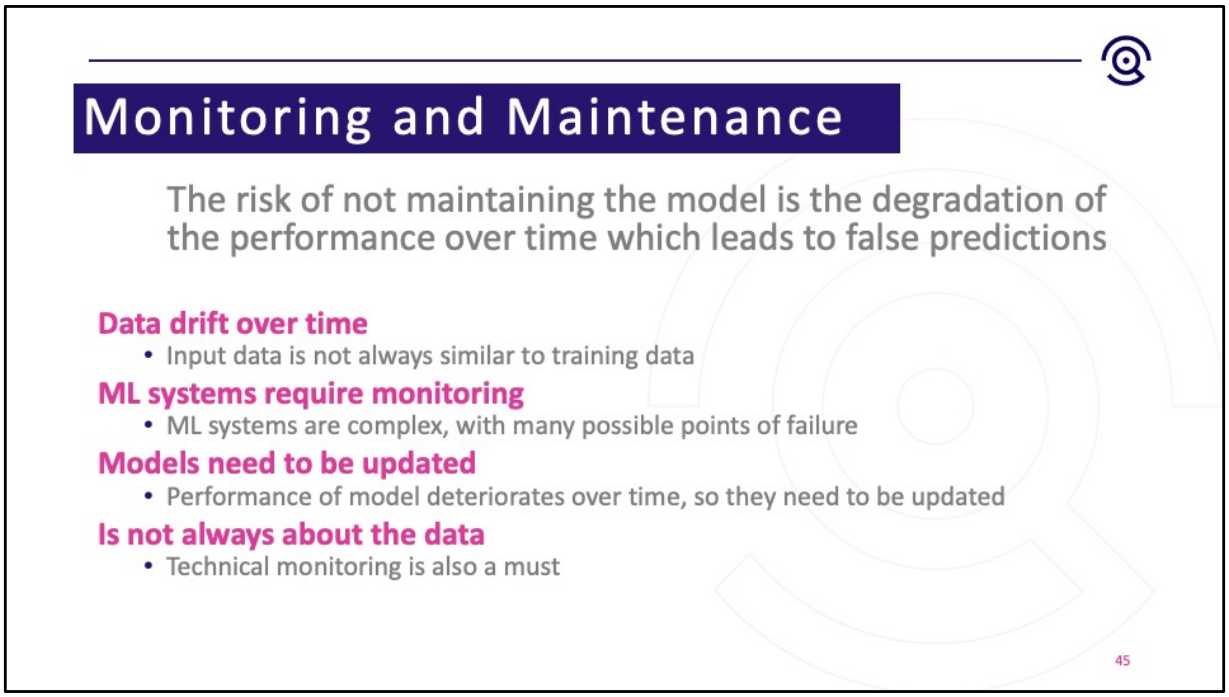
functionalities as well as limitations of the proposed solution.

- It is very important that we minimize the risk of unforeseen errors. But there is always the possibility of the unknown unknowns. A fall-back plan that is activated in case of e.g. erroneous model

updates or detected bugs, can help to tackle the problem. Options are to roll back to a previous version, a pre-defined baseline or a rule-based system.

- Finally, we should try to minimize the impact of such erroneous deployments and the cost of fixing errors. It is

best practice to setup an incremental deployment strategy that includes a pipeline for models and data before rolling out the whole solution.



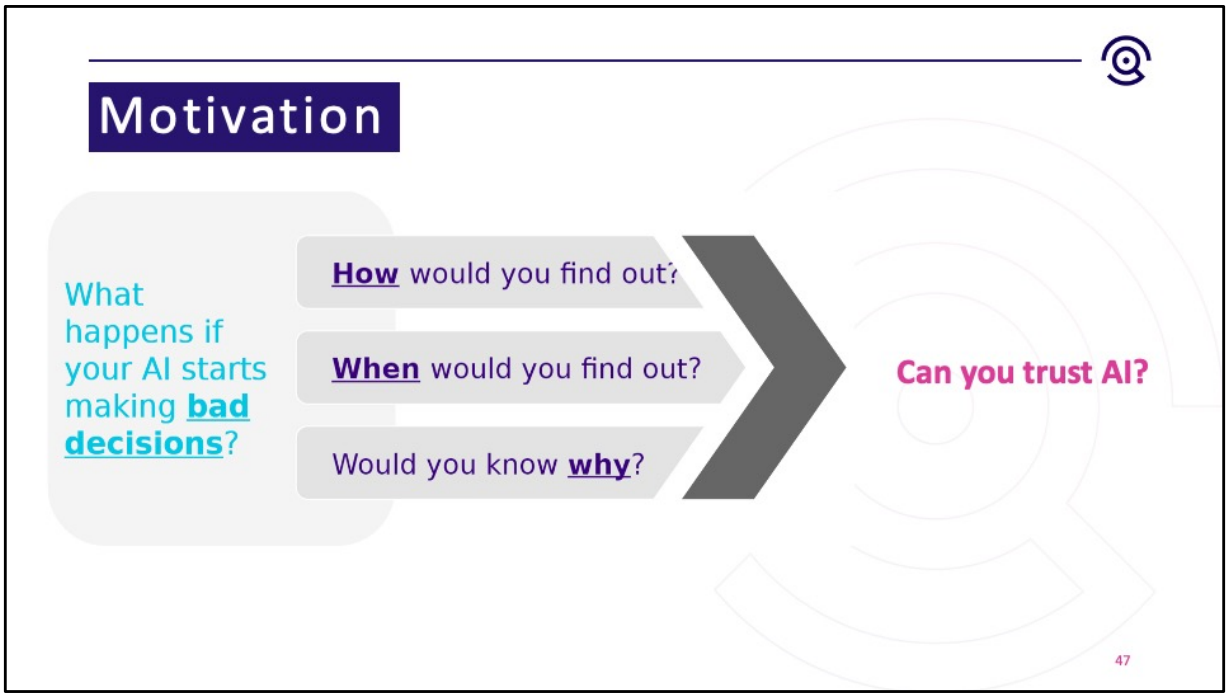
The field of data analytics has evolved from static knowledge discovery and insights generation to data-driven applications able to infer real-time decisions. ML models therefore tend to be used over a long period of time and have a life cycle which has to be managed properly. The risk of not maintaining the model is the degradation of performance over time, which leads to false predictions and in turn this can evolve into operational risk as well as other types of risks. There are many reasons for monitoring and maintaining your models, and we will explore them more in detail in a next section, but let's already have a quick overview:

- The main assumption of ML is that the training data resembles the input data on which a model is eventually tested and applied. However, data is not static and changes over time, which cause the models to decay over time.
- It is for this reason that ML systems should be monitored. On top of that, ML systems are highly complex and as in any software there are many factors that may occasion a production failure. If models are not monitored, these failures can't be addressed.
- To mitigate this risk, models need to be periodically updated. It should be noted that ML systems might influence their own behaviour during updates due to feedback loops, either directly (e.g., by influencing its future training data selection) or indirectly (e.g., via interaction with the world). The risk of feedback

loops causing system instability needs to be addressed, for example by not only monitoring but limiting the actions of the model.

- Although model performance is the most evident concern that comes to mind when models are moved to production, there is also the need for technical monitoring. Degradation of hardware (e.g., failing sensors) and system updates might also negatively affect the well functioning of an ML system.

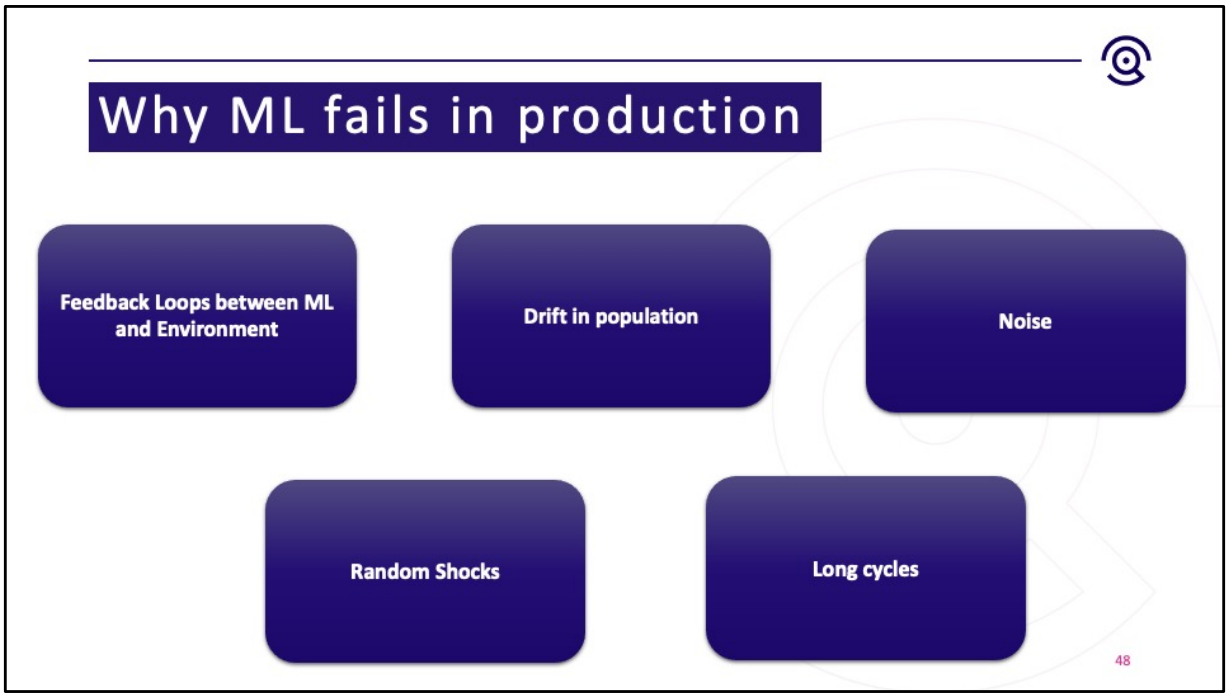
Let’s find out what AI observability entails exactly.



Deployment of an AI system is by far not the closing chapter of the project lifecycle. The question that you should continuously ponder on is the following: What happens if your AI starts making bad decisions?

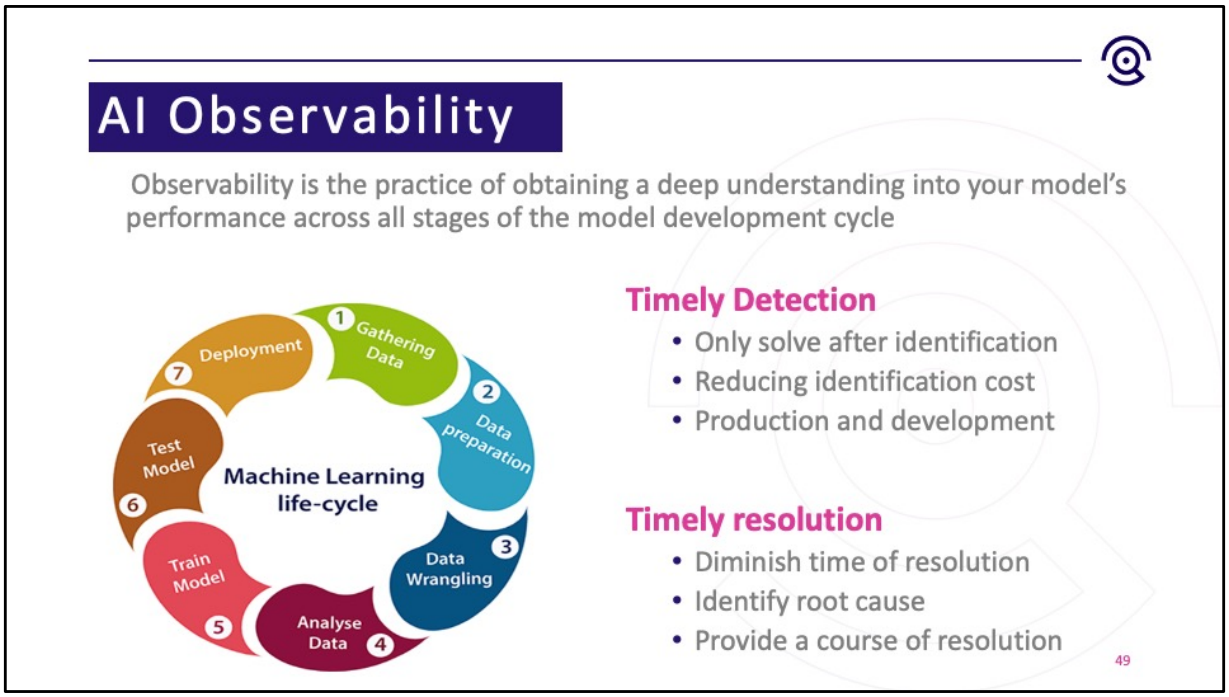
- How would you find out? It is not the same to observe the quality of your model decay gradually over time as watching your whole system crash in flames.
- But also, when would you find out? Are you able to react in a timely manner? If your model only provides insights then time might not be as pressing, but if your model is directly supporting operations and decision-making then finding out too late might come at a big cost.
- And finally, would you know why the AI failed? Would you have the ability to correct the course or would you have to start from scratch and take a wildly different approach?

What really matters in the end is being aware whether or not you can trust your AI solution and that you are able to identify when and why it is not working, as quickly as possible.



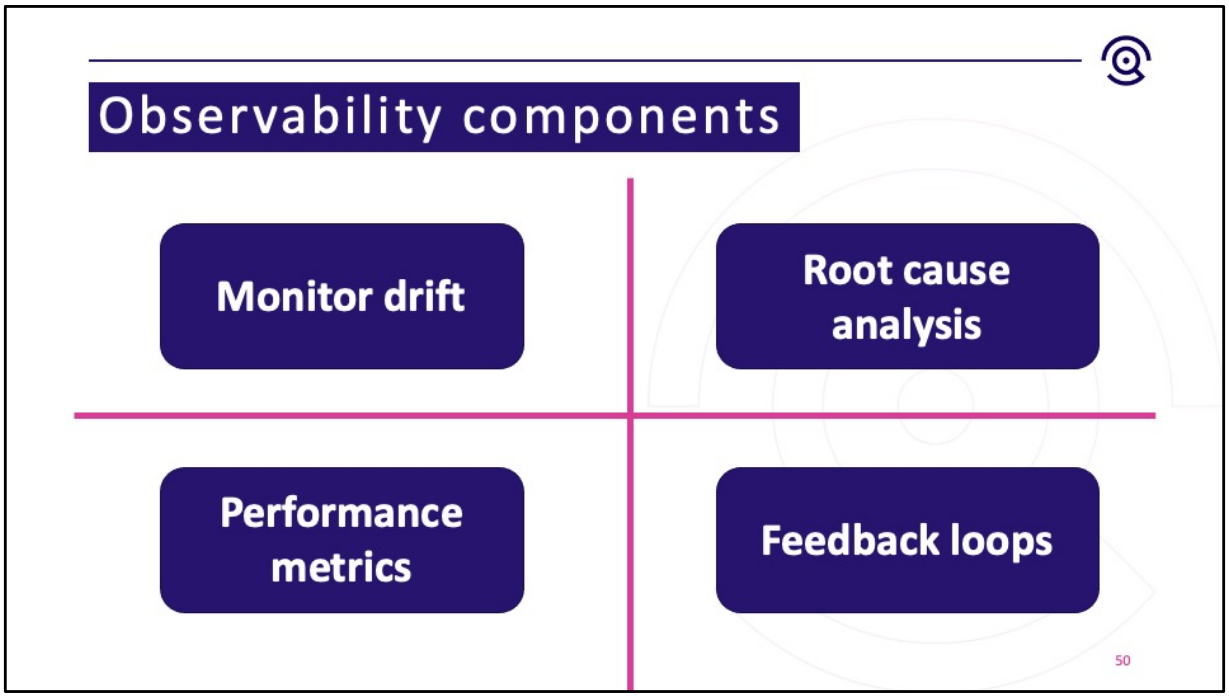
ML systems can fail in production, and in fact, it is normal that they do. It is more a matter of when than a matter of if. For this reason, being prepared for this failure is paramount when building your AI solution. There can be many points of failure, but the majority of them fall in one of following five categories:

- Feedback loops between ML and the environment. For example, an end user consciously changing the information on a loan application to fool the ML model.
- Random shocks. Imagine a new regulation coming into play that directly affects the target of the models or an external event like the Covid-19 pandemic which changes the behaviour of the population.
- Drift in population. When we expand the customer basis to a new market, it might be that the new population has a different behaviour than the one we used to train the model.
- Noise. If you are creating a model for credit scoring, it might be possible that there are bugs on the underlying credit scoring system from which we get our labelled data.
- Long cycles. We could train our model during a boom economy but ending up going to production during a recession economy. These kind of cycles are of course not limited to economics but they can also appear in other places in nature.



To be able to anticipate and react to these failures, we need to understand what drives our ML system's performance. Observability is the practice of obtaining a deep understanding into your model's performance across all stages of the model development cycle. From the first moment that we start development, but also once it's been deployed and long into its life in a production environment. Observability guides us into not flying blind once a model is deployed and it helps teams to iterate and improve their models quickly.

The main two goals of observability are timely detection and timely resolution. On the one hand, a problem can't be solved before it is identified. Therefore, reducing the identification cost is key. Timely detection of problems at both production and development are important. A good observability tool might help a model builder find problems with their model quicker, facilitating a tighter iteration loop. In the context of a production scenario, an ML observability tool might monitor key performance metrics that can notify a model owner when something is going wrong. On the other hand, once problems have been identified, diminishing the time of resolution is also important. For this we should be able to Identify the root cause and provide a clear course of resolution.

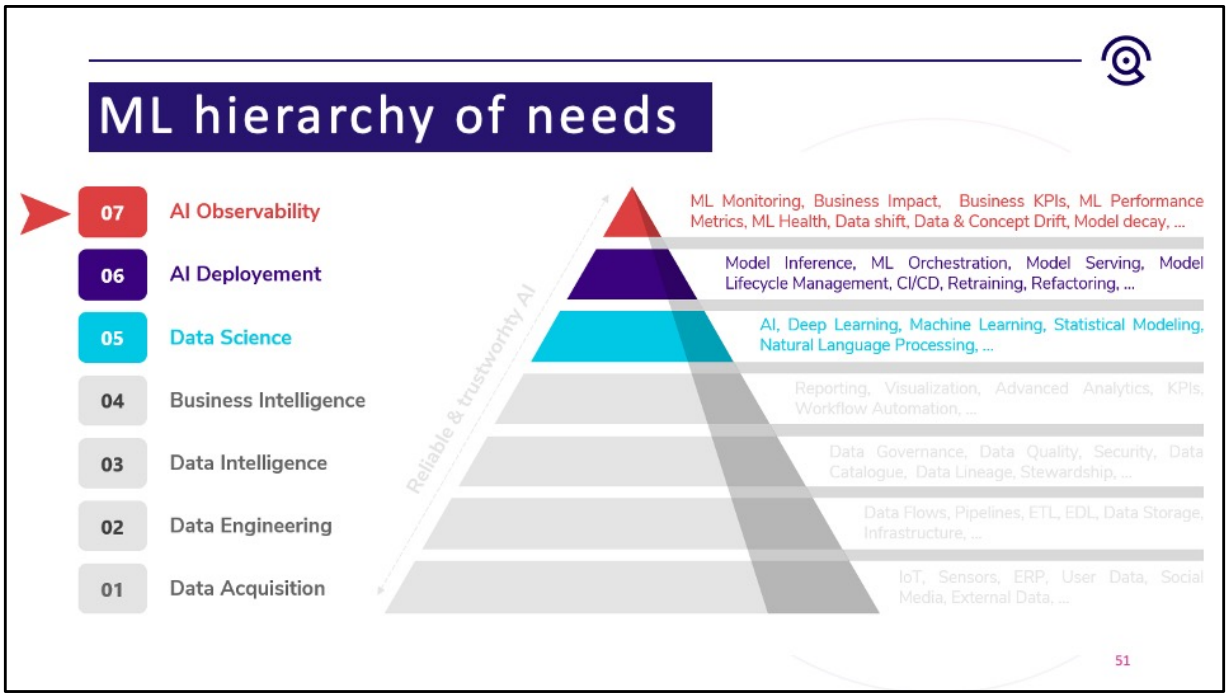


Let's have a deeper look into the main components of a good observability strategy that will allow us to achieve timely detection and resolution of problems. They are four components, namely monitor drift, root cause analysis, analysis of performance metrics and exploiting feedback loops:

- Monitoring drift refers to data quality issues such as data drift or anomalous performance degradations using baselines. Keeping track of the ground truth of our ML system is at the center of good monitoring practices. In some model applications, the ground truth is readily available immediately after the model makes its prediction. However, in other scenarios, the ground truth can be delayed or even missing entirely. So monitoring is not always an easy feat to achieve.
- Root cause analysis will help in finding what the main reasons behind failures are. They might differ depending on the type and complexity of the ML system. Having good metadata for the models is paramount to performing this kind of exercise.
- Analyzing performance metrics should be standard practice when constructing an AI system, regardless of the environment (production, validation or training). These metrics might not only be ML related, but reaching business KPIs, economic criteria and technical KPIs (like resource

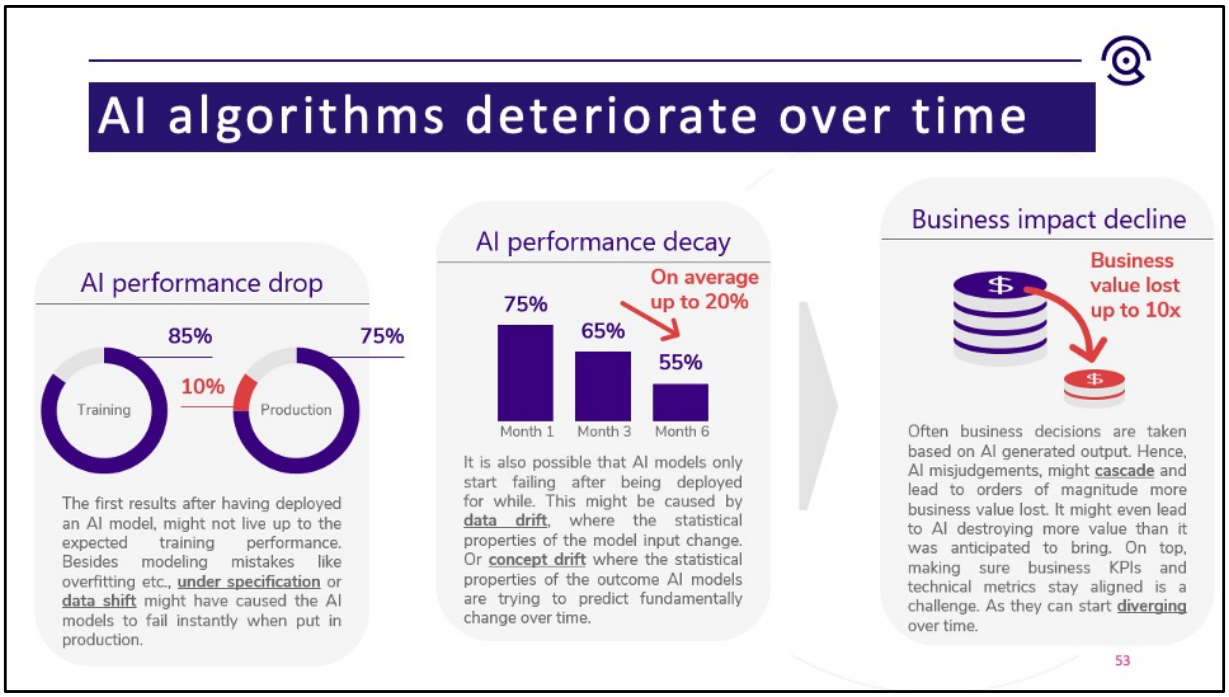
management) are all important when implementing an observability strategy.

- Finally, models can learn continuously from their environment. This enables feedback loops to actively improve the model performance which can take your ML system to the next level.



The general idea of the ML hierarchy of needs is that you should not move up the hierarchy until you've done the basics in the prior stage. We see from the picture that AI observability is at the top of the pyramid. It is true that lacking observability will harm your models performance as well as the development experience. However, you can't have good observability if your building blocks are lacking structure or are underdeveloped. If you are just starting your AI journey, it is important that you keep in mind AI observability as an ulterior goal, but you also have to realize that it requires an extraordinary level of maturity to reach that level of sophistication. This is exactly why having an solid MLOps strategy from the get go is important.

Let’s talk about the monitoring of your AI solutions after they have been deployed into production.



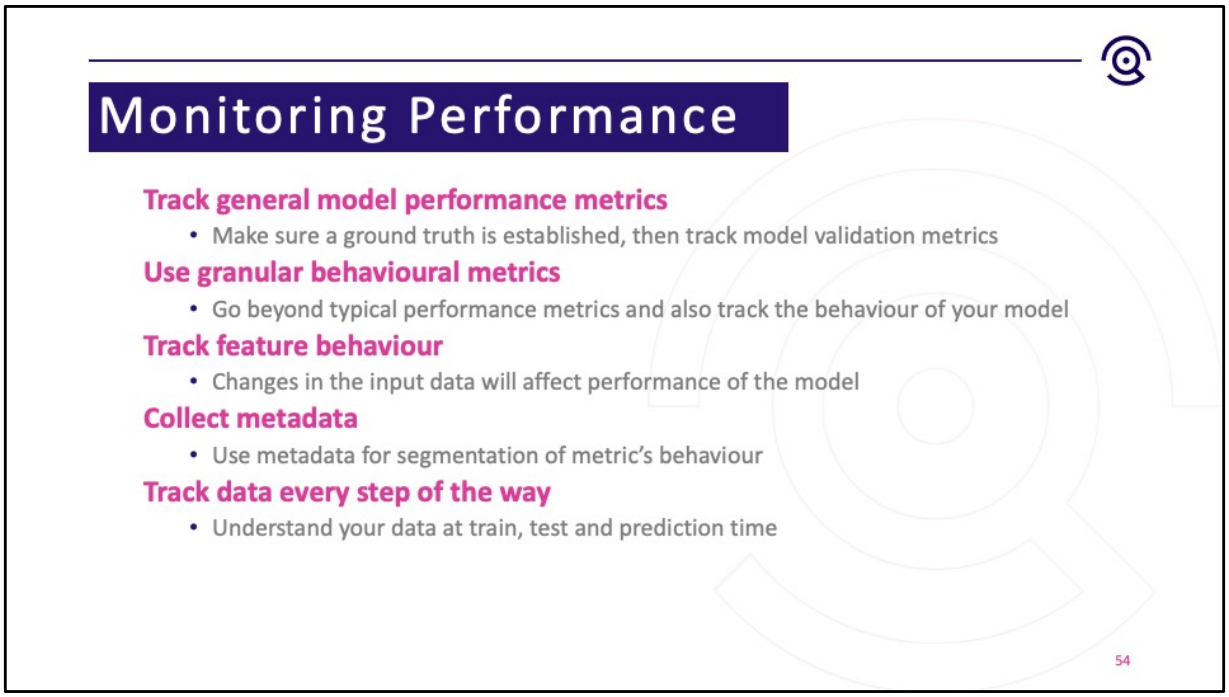
As we have discussed before, it is natural for AI models to deteriorate over time. However, it is not always a visibly gradual process.

AI performance can drop immediately after it is brought to production. The first results we get after having developed a model might not live up to the training performance. Apart from straightforward modelling mistakes like overfitting, other more complicated issues like under specification or data shift can also affect production results.

It is possible that models stop performing after already having been functional for a while. This is mainly driven by changes on the properties of the data or data drift. But also by more fundamental changes in the relationship between the target variable and the other features or concept drift. Although basic monitoring can raise flags when models are decaying, it is important to have other mechanism in place to also identify the cause of degradation.

AI can create business value, but it also has the potential to destroy it. AI misjudgements can lead to bad business decisions which can cascade over time. It is also hard to keep business KPIs and technical metrics aligned as they can start

diverging over time as well.

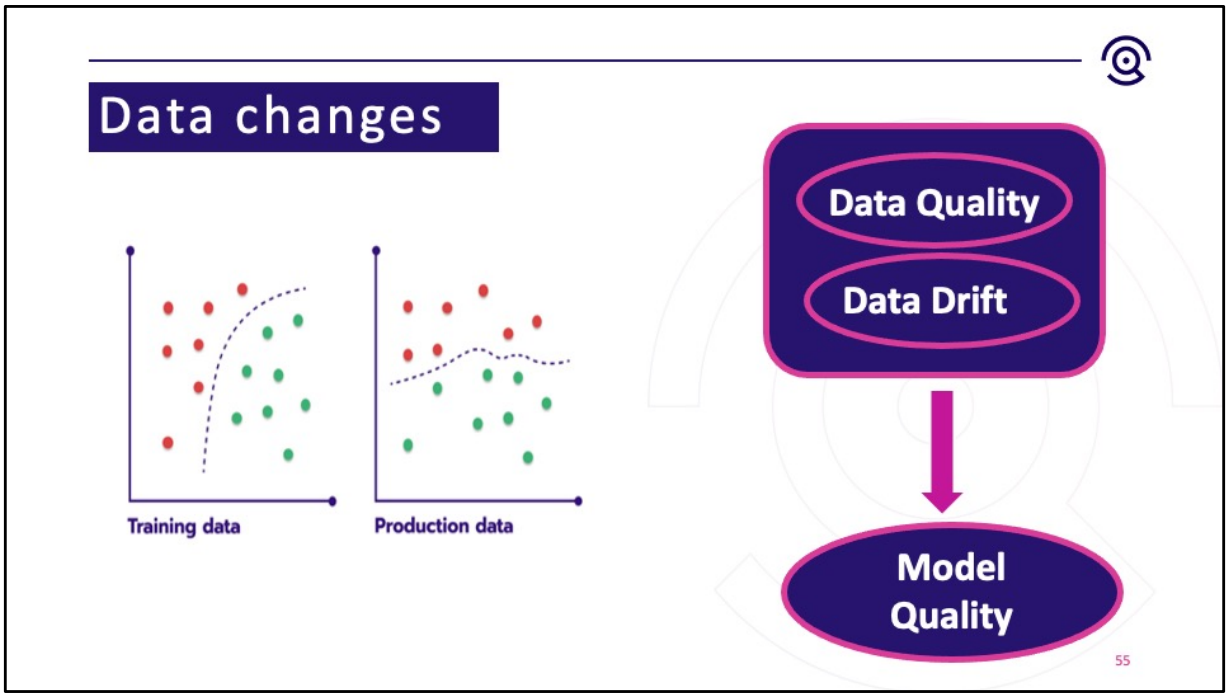


It seems natural that monitoring your AI systems in production must be a given. But like is the case with many AI components, there is no one size fits all approach for monitoring. AI systems are diverse and complex, making organizations struggle on performing this crucial task. Most of the time this leads to simply resorting to manual retraining. Although it might seems expensive and in many cases even futile to monitor your AI systems, every company that has AI/ML models in production must also consider the costs associated with model upkeep. Retrain too early and you will incur added compute and personnel costs. Retrain too late and you incur the business costs of a poorly performing model and, potentially, the reputation costs of inaccurate predictions.

Monitoring the performance of the models is necessary to maintain healthy AI systems in production. The main actions to be taken when monitoring performance are the following:

- Track general model performance metrics. Make sure a ground truth is established and then track model validation metrics. Having quality levels assures that you are able to compute well established key performance metrics like ROC or RMSE, depending on the kind of solution you are implementing.

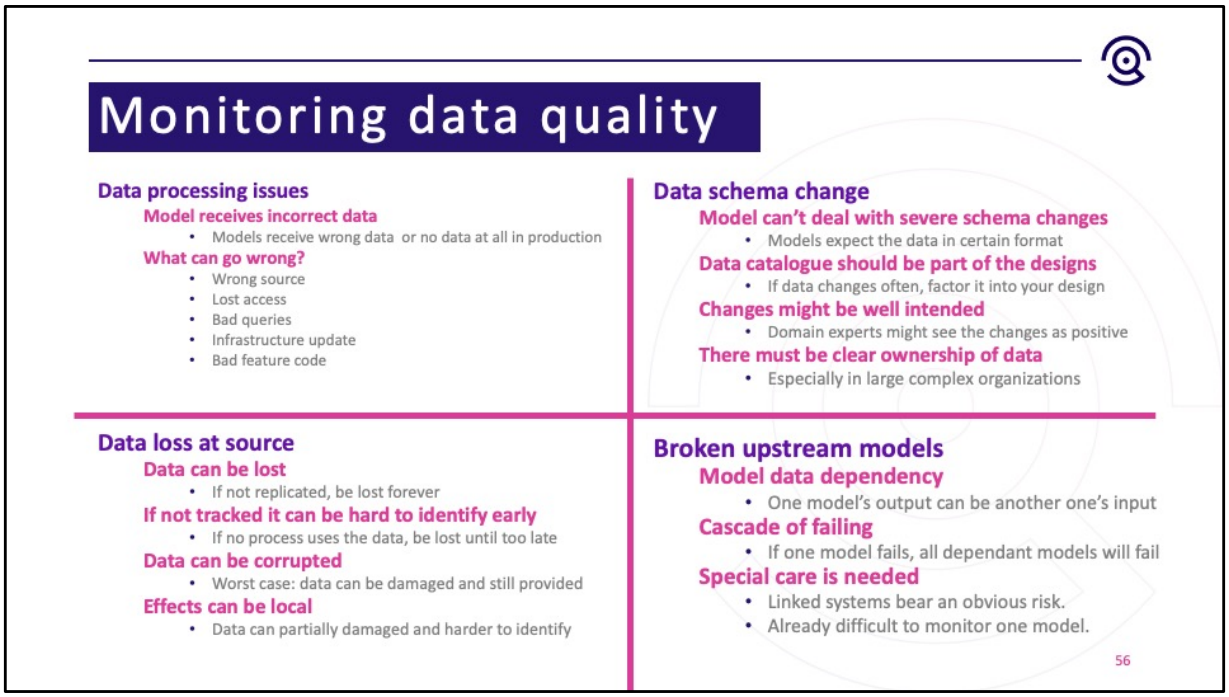
- Use granular behavioural metrics. Go beyond typical performance metrics and also track the behaviour of your model. It is important to understand how your model is changing in time and interacting with the environment. Perform statistical analysis of the model's scores, calculate confidence intervals over time for your scores and so on. Is not only about a metric at one point in time but how it changes over time.
- Track feature behaviour. Changes in the input data will affect performance of the model. Tracking feature behaviour will allow you to better explain changes that were detected in the output of the system. It also allows detecting other issues that may pop up in later stages of the project lifecycle.
- Collect metadata and use this metadata for the segmentation of metric behaviour. Without metadata you'd only be able to track the behaviour of the metrics at a global level, but having rich metadata will allow you to dig deeper on different segments of the population.
- Track data every step of the way. You should understand your data at train, test and prediction time. When models underperform at prediction time, having the ability to compare the features distribution for that segment of data with the corresponding distribution for when the model was trained, can provide the best insights into the root causes of the change in behaviour.



As we have said before, data is not static. It might change over time or even between development and production. Ensuring the quality of your data and monitoring for data drift is of the upmost importance for preserving the quality of your models. In a good set up, the monitoring process that prepares the new raw data for scoring should help ensure the quality of it. Monitoring incoming scoring data to make sure that it is not trending over time or undergoing dramatic shifts helps us to keep data drift in check.

There is a small caveat here, because models are built on complex and complicated patterns. Data quality and data drift may be completely in bounds and yet model quality may erode over time due to small, hard-to-see changes in the incoming data. Such changes can only be caught by assessing the accuracy of models. This requires ground truth, and while in some cases it is easy to obtain this, in other cases it can be completely impossible.

Lets dig a little bit deeper on how to monitor data quality.



Data quality issues arise from a variety of fronts like data processing issues, schema changes, loss at source or even broken upstream models. Each of which needs to be tackled in a very different way.

Data processing issues are probably the most common source of issues when it comes to data quality:

- Models receiving incorrect data or no data at all while in production is something every data professional has faced at some point in time.
- Many things can go wrong and a solution should be designed at problem level. Possible causes are a wrong source, lost access, bad queries or infrastructure updates. But probably one of the most underestimated ones is bad feature code, as many data scientists are not well versed in neither software nor data engineering, so the quality of most of the code they produce is far from production ready.

Data schema changes are another issue that data science professionals are frequently confronted with. Some examples are the following:

- Models can't automatically deal with severe or even simple schema changes. They normally expect the data in a certain format, unless the models are specifically

programmed to adapt to the changes.

- A data catalogue should be part of the designs. This is especially the case if data changes often. Having metadata tacking schema changes will allow AI teams to track and correct errors arising from such schema changes.
- The ironic part of the story is that schema changes might be well intended. Domain experts might perceive the changes as positive, as they genuinely improve the quality of the data. But if there are models in production already expecting for the data to look a certain way, then abrupt schema changes will break them regardless of the good intentions.
- It is for this precise reason that there must be clear ownership of data.

Especially in large complex organizations, were there is huge disconnect between owners of some particular data and the users. A change in the data schema might break many applications that are dependant on the source.

An even more serious problem, is when there is data loss at the source.

- Despite the IT team's best efforts, data can be lost. If not replicated, it can even be lost forever. And when the model does not receive data, it will simply not score.
- A big problem is that if this is not tracked it can be hard to identify early. If no process uses the data, data can be lost until it is too late. For example, data can be available in development but not in production, and once it is time to deploy it can be very difficult to get the data back.
- But even worse, data can be corrupted. A worst case scenario is when data is damaged but still provided like everything is going fine. This directly implies garbage in garbage out.
- All this becomes even more difficult to identify is when the effects are local. Data can be for example partially damaged, which makes it even harder to identify errors.

Finally, for more complex systems we can have broken upstream models.

- In situations where there is model data dependency, then one model's output can be another one's input.
- This can cause a cascade of failing if one model fails, because all dependant models will also fail.
- This requires special care since such linked systems bear an obvious risk. And we know that monitoring a single model is already quite difficult.

Ensuring data integrity

Errors happen, but what to do?

Data schema
- Perform a feature level automated check

Missing data
- Set a combination of monitoring policies to detect and correct

Feature values
- Check if values are abnormal or data shift

Feature processing
- Validate each step of your data's prepossessing journey

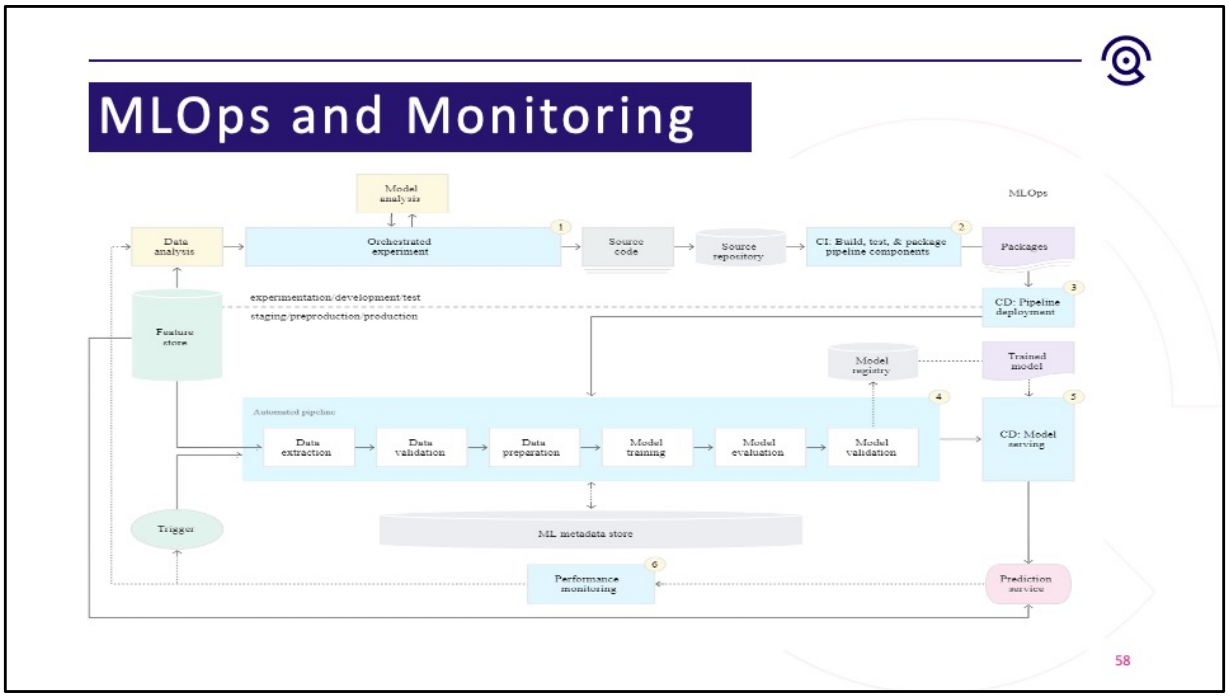
57

Now, an important step is to take the leap between monitoring data quality and ensuring its integrity.

Errors happen, but what to do? Lets constrain ourselves to four of the most straightforward scenarios:

- How to prevent data schema errors? Performing a feature level automated check is the minimum you can do. In the end, you want to be able to have a quick summary view showing that the incoming dataset is shaped as expected.
- How to deal with missing data? Often, there is some acceptable share of the missing values that we tolerate. We do not want to react at each empty entry, but we want to check if the level of missing data stays within the "normal" range. Each case of missing data can have different reasons and a combination of monitoring policies to detect and correct must be in place.
- How to check the validity of feature values? Data can be available, but that does that mean we are dealing with correct data. Always check that values are not abnormal and that there is no data shift. The goal is then to monitor the live dataset for compliance with expectations and validate the data at the input. This way you can catch an error when you detect a range violation, some unusual values, or a shift in statistics.

- How to deal with feature processing? Validate each step of your data's prepossessing journey. This way, we can locate the problem and debug it faster. Data normally goes a long way between its raw form and becoming a usable feature. Having validations at each step allows you to find errors faster.



Finally, lets have a quick look at the overall importance of monitoring on MLOps. In this section we discussed how monitoring performance, data quality and data integrity is key to having a healthy AI/ML system. However, by having a quick look at the picture we realize that there are many other places that require special types of monitoring. Some examples are:

- Monitoring the resources used by the application and making sure it is able to run smoothy.
- Monitoring the CI/CD and make sure other technical applications are working correctly.
- Monitoring and tracking experimentation results.

- Long story short, monitoring is a very big topic for which we have only scratched the surface. It might not be the first priority when you start your AI journey, but you should definitively start the journey with it in the back of your mind.

This is the end of the fifth and final module of our AI4Business. Congratulations, you made it all the way until the end! I hope that this course has brought you an abundance of new AI knowledge and skills. Please feel free to get in touch with us and/or fill out our brief feedback survey on the website. I thank you for your attention and wish you all the best in your future AI endeavors, bye!