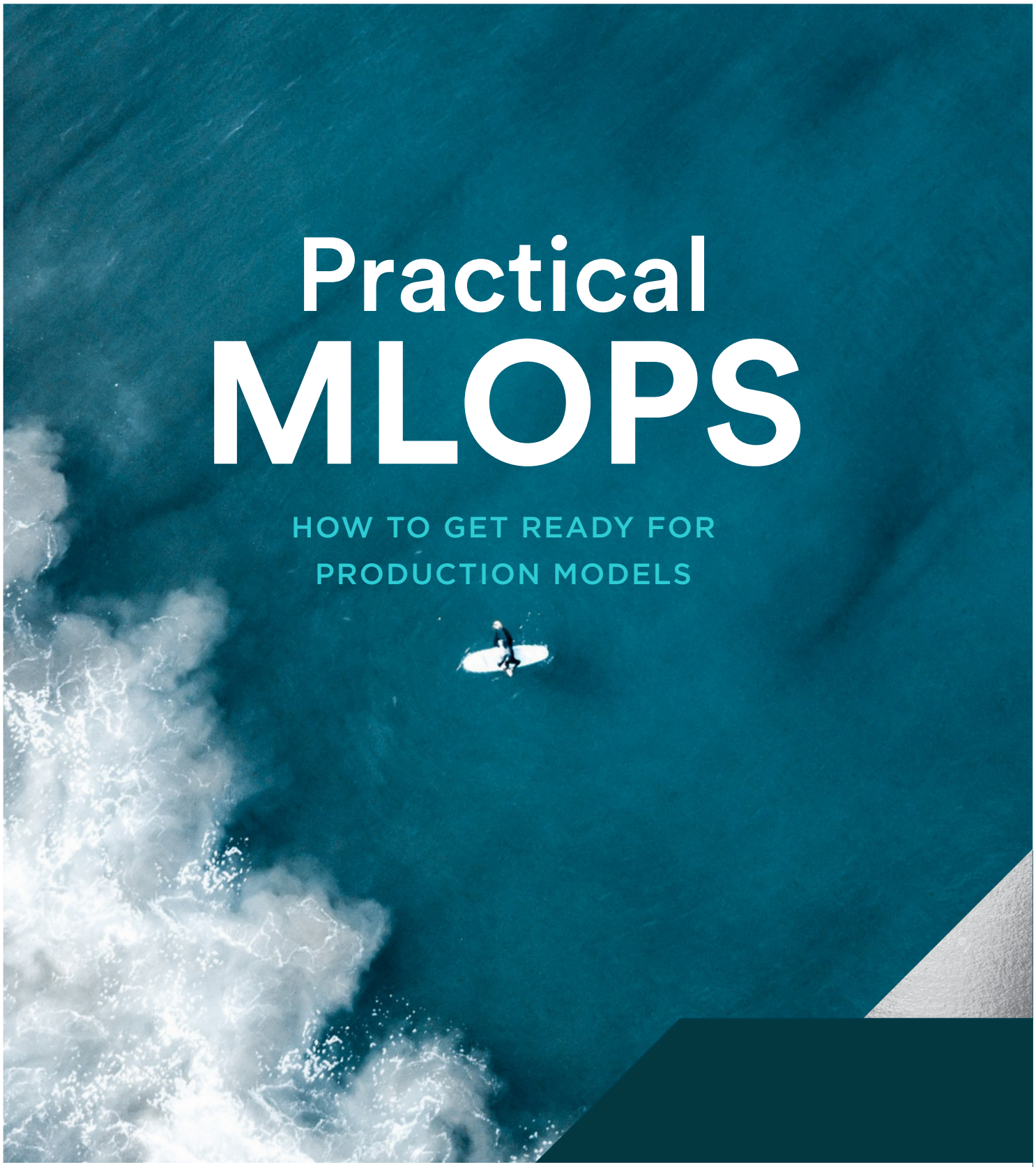
Valohai



Practical MLOPS

HOW TO GET READY FOR PRODUCTION MODELS

WITH CHAPTERS FROM

SIGOPT

tecton

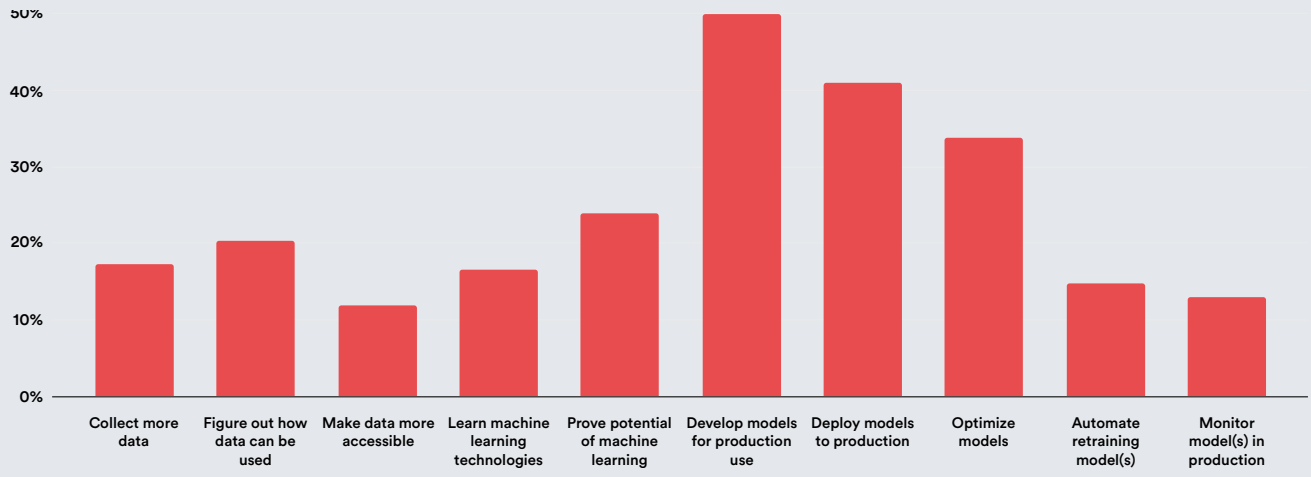
Contents

Why MLOps Matters? (Foreword)

Traditionally, machine learning has been approached from a perspective of individual scientific experiments which are predominantly carried out in isolation by data scientists. However, as machine learning models become part of real-world solutions and critical to business, we will have to shift our perspective, not to depreciate scientific principles but to make them more easily accessible, reproducible and collaborative.

In May 2020, we surveyed 330 data scientists, machine learning engineers and managers in a broad range of companies to ask what they were focused on for the next 3 months and what major obstacles they were faced with. Although 20% of respondents said they are still focusing more on the experimentation and learning phase, half of the respondents said that they were focused on developing models for production use and over 40% said they would be deploying models for production.

Source: State of ML 2020, Valohai
330 respondents



Automating retraining of models and monitoring models in production were still not relevant for most respondents which speaks to machine learning in production being relatively nascent for most. However, the steadily rising interest in MLOps based on search volume and news articles speak to these aspects becoming more and more relevant. Production models raise new challenges, not just for data scientists but the extended team of engineers, product managers and compliance officers, which will need to be solved collaboratively.

In most real-world applications, the underlying data changes constantly and thus models need to be retrained or whole pipelines even need to be rebuilt to tackle feature drift. Business and regulatory requirements can also change rapidly requiring a more frequent release cycle. This is where MLOps comes in to combine operational know-how with machine learning and data science knowledge.

In machine learning, 2020 is the year of production models and we predict that 2021 will be the year of MLOps. We hope that you enjoy our eBook on MLOps and get new, fresh ideas for your use case.

People of Machine Learning

The size and scope of real-world machine learning projects has surely surprised most if not all of us. What seems like a straightforward task of gathering some data, training a model and then using it for profit ends up becoming a deep rabbit hole that spans from business and operations to IT. A single project covers topics such as data storage, data security and access control, resource management, high availability, integrations to existing business applications, testing, retraining and so much more. Many machine learning projects end up being some of the biggest multidisciplinary and cross-organizational development efforts that the companies have ever faced.

To understand the multidisciplinary nature of ML projects, we start by looking at the roles involved in them. These roles come from our unique collective vantage point of seeing around 500 different organizations from startups to Fortune 500 over the last 4 years. While the list of roles below is not entirely exhaustive, it gives an overview of the people involved.

Remember that these roles are not necessarily one per person but rather a single person can - and in smaller organizations often has to - cover multiple roles. For example, many data scientists also handle ML Engineering tasks, and sometimes DevOps and IT are synonymous. The rule of thumb is the larger the organization, the more specialized and siloed individuals are which requires all the more collaboration.

Data Scientist

Data scientists are tasked with finding data-driven solutions to business problems. Data scientists tend to be highly educated with a master's degree or higher in mathematics and statistics, computer science or engineering. Their strengths lie in data wrangling, statistical analysis, data visualization and applying machine learning.

Generally, they are deployed first to analyze existing data and to find

patterns that could be used to solve underlying business problems. For example, they might be looking at user data to find meaningful user segments and building models that can classify those users in segments in order to differentiate the end-user experience and drive more engagement. While the primary purpose of data science is to explore data and build models, often a large portion of time is spent manipulating data to suit the use case.

Data Engineer

Data engineers are responsible for the infrastructure that ensures data is stored and available for data scientists. They build the systems that ingest raw data from various sources and store it in a centralized data lake. Data engineers often also build other data warehouses that derive data from the data lake but transform it so that it is more useful for the end-user; for example, image data might be resized and reformatted so that data scientists can focus on data analysis rather than data manipulation.

Machine Learning Engineer

Machine learning engineer is a rather nascent role that is becoming increasingly relevant as companies push to get real-world value from machine learning. While data scientists explore and prove a theoretical concept, ML engineers take those concepts and build them into a production-scale system. As the title implies, ML engineers understand the theoretical side of machine learning but are more involved with paradigms found in engineering such as continuous integration (CI) and continuous delivery (CD). They tackle automating and scaling model training, testing models before deployment and deploying and monitoring models in production. Often they will rewrite or restructure code written by data scientists; for example, turning notebooks into scripts.

DevOps Engineer

DevOps engineers combine the domains of software engineering and cloud infrastructure. In traditional software projects, DevOps

handles automation around testing and releasing new code, and often also manages testing and production environments. In machine learning projects, DevOps can be the connection between the ML model and the end-user application; for example, releasing a new recommendation engine to a web application. Scalability, stability and responsiveness of the end-user application are often at the forefront of their mind and they'll be involved in ensuring machine learning models don't degrade these elements.

IT

IT operations can be found in most larger organizations. They are generally tasked with resource management, access control and information security. While often seen as red tape, IT operations and the processes involved can help a project run more smoothly, whether it's through requesting that computing resources be provisioned or ensuring that outside vendors are approved.

Business Owner

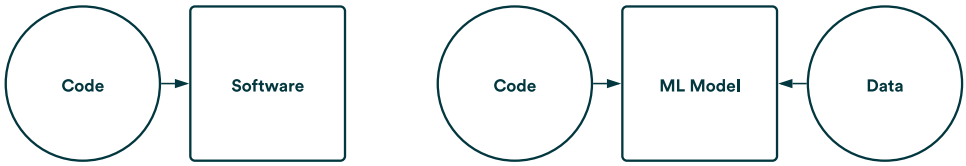
There isn't a specific title for a business owner in a machine learning environment but it's a key role to ensure success. Business owners have the deepest understanding of the end-user and use case, and guide the team to producing ML systems that are not just accurate and well-engineered but also valuable. They'll help define what kind of predictions are useful and help understanding of where risks lie from an operational and regulatory perspective.

Manager

The management role in a team working with machine learning tends to be a little unclear as these teams tend to be relatively new. The key aspects to consider are that machine learning projects are deceptively complex and multidisciplinary. All the required resources to execute a project are often not in a single team and a manager must secure these from different parts of the organization. Another difficult task for a manager is to understand and communicate the ROI of the team's work as costs and benefits are likely not directly linked with the team.

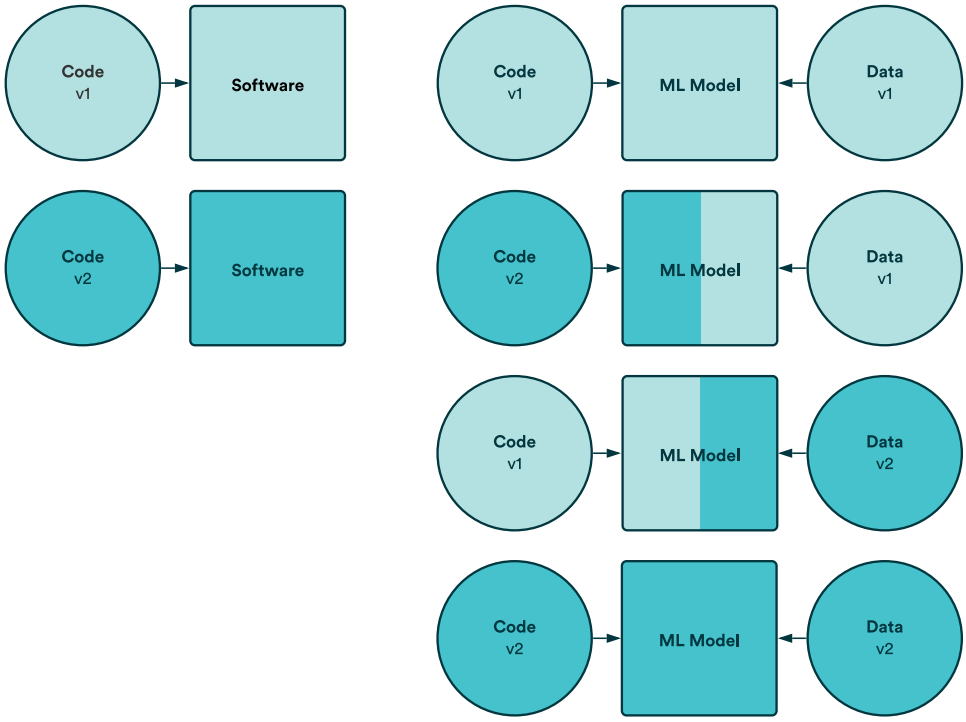
How Is Machine Learning Different from Traditional Software?

We already established that the cross-domain nature of ML causes a new level of discipline in organizations but that's only part of the picture. There is an additional dimension that differentiates ML and classical software — data. It might sound self explanatory and simple but the addition of data, usually big data, brings entirely new challenges to the development process.



While software can generally be developed locally with almost instant feedback on how a code change affects the end-result; in machine learning, to see the effects of a code change requires retraining a model. When working with models that are trained with large datasets, this poses huge infrastructure challenges as either you will be waiting for a long, long time or will need to use remote computing. Just imagine the difference between an application developer just saving a code change and pressing refresh in their browser, and a data scientist saving a similar code change and then spinning up a cluster of GPUs, deploying code, transferring data and training a model before seeing any results.

However, it's not only how the introduction of data changes how code works in the development process but also that data is another aspect that can change. In classical software development, a version of the code produces a version of the software, while in machine learning, a version of the code and a version of the data together produce a version of the ML model.



Data as a new variable in the development process increases complexity drastically, as to reproduce results you need to be able to reproduce the data that you used. While you might not be producing every possible variant of the model, the components must be versioned to ensure reproducibility. The matter of versioning both data and code becomes increasingly complex when you start thinking about whole machine learning systems which have steps for data preparation, augmentation and generation.

The MLOps Workflow

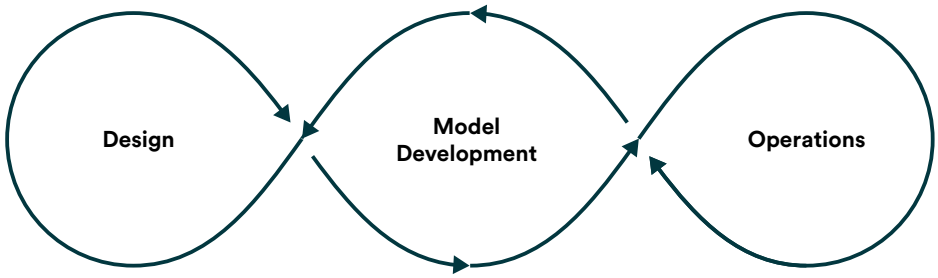
What Is the Point of MLOps?

You might have downloaded this book because you intuitively know that MLOps is essential. This intuition arguably comes from the pursuit of doing things the right way and feeling the pain of constant technical struggle in one's daily work.

However, intuition alone can be hard to argue with when talking to teammates looking for approval for an investment – of time or money.

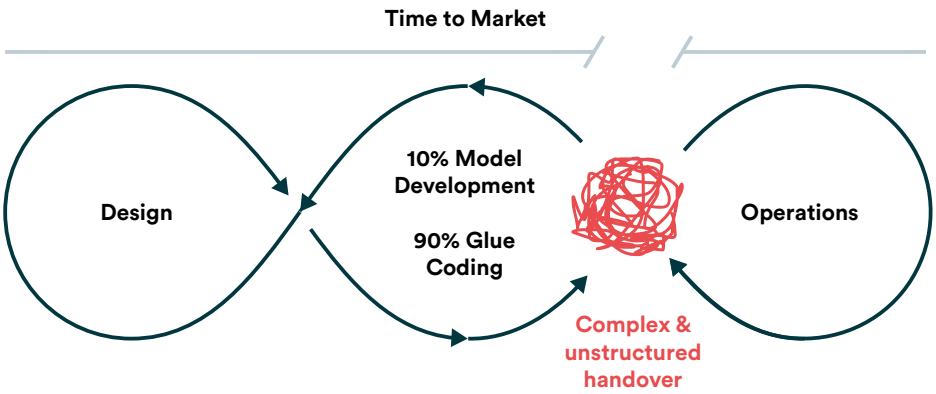
This chapter solidifies the idea of MLOps to help understand why it is worth focusing on and how to justify this. The chapter will also help assess what areas to focus on, and the following chapters will dive deeper into concrete ways to measure and show the ROI.

When we look at how the current literature describes a model lifecycle, we often see a picture like this:



However, in reality, most organizations today still operate in a cycle where hand-overs between model development and operations are – simply put – messy and manual.

As fun as development in a vacuum is (we all have done it), the reality is that only a model running in production can bring value. Models have zero ROI until they can be used. Therefore, time to market should be the number one metric to look at and optimize for any ML project.



> The goal of MLOps is to reduce technical friction to get the model from an idea into production in the shortest possible time to market with as little risk as possible.

Let's break down the statement. It contains two parts, reducing the time to market and reducing the risk involved. These goals help data scientists and business stakeholders to speak the same language and frame MLOps as a business-driven necessity. Albeit MLOps will help automate manual steps and improve the quality of code, but they are not the underlying goals the whole organization will rally around.

Next, we will talk about risk and time to market. The area of risk assessment is a more straightforward concept, so let's start there.

Risk in Machine Learning

The goal of MLOps is to reduce technical friction to get the model from an idea into production in the shortest possible time to market with as little risk as possible.

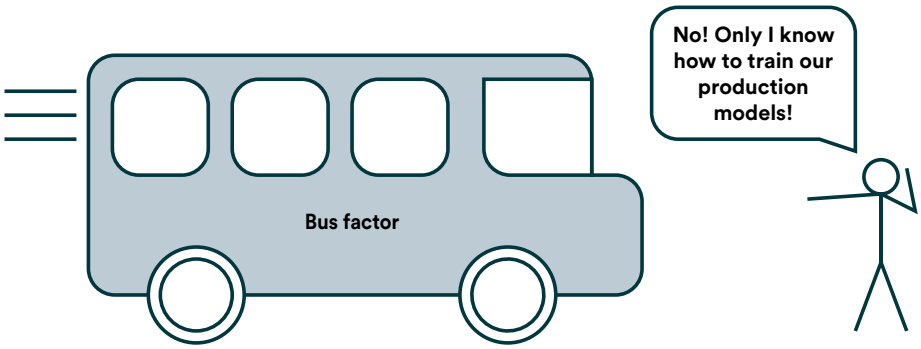
We have identified three primary types of risks the MLOps workflow tackles:

1. Loss of knowledge
2. Failures in production
3. Regulatory and ethical

There is plenty of industry, implementation, and even company-specific detail involved in each of these main topics. However, the most significant risks that machine learning in production introduces fall under one of these categories.

Keep in mind that we are skipping basic IT risk that is already well documented and covered elsewhere but is not exclusive to machine learning in production; for example, data security. It's clear that no-one should be able to steal data from your database, but that is true for all databases – not just ones used for machine learning.

Loss of Knowledge



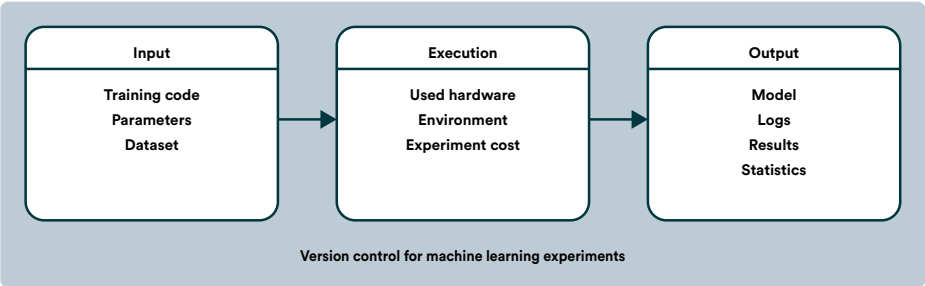
The bus factor is a common term in software engineering describing the risk of a key contributor disappearing unexpectedly from a project – because they get hit by a bus. While it is a well-documented problem for classical software development, machine learning magnifies the bus factor significantly.

In ML, in addition to code that is often somewhat self-documenting, there is data. A single data set can contain vast amounts of hidden information about how it's been collected and then how features are extracted. There may be many versions of notebooks or scripts that produce a single data set.

Loss of knowledge can happen in many different ways. Most obviously by a data scientist leaving the company, and most larger teams have already faced this. But it can also occur without personnel changes. When you work on multiple projects over a more extended period, you

tend to forget details about your own work too. It can be challenging to return to a six-month-old project and retrain a model without breaking things. Many readers may have existing models running that they dread getting back to for fear of breaking something.

For traditional software development, reproducibility is mostly taken care of by proven version control tools like Git. To achieve proper version control and reproducibility for ML, you will need a bit more on top of that.



In ML, a model is a combination of code, data, parameters, and the training environment. To pick up where someone else (or yourself six months ago) left off with a model, you'll need much more than just a Git repository.

You need to know what data the code is using and how that data came to be. There are often multiple scripts bringing together different data sources and doing feature aggregation to build the dataset you need. Putting together the puzzle of several scripts and data sources someone else made leaving behind zero documentation can often be more complicated than just starting from scratch.

To tackle this, we recommend a version control system that can track your whole pipeline from raw data to the model, including your code, environment, configurations, and parameters.

Failures in Production

The next category of risk is failures in production. Everyone who has worked in software development knows that shipping broken code

to production is a prevalent failure point for systems. Teams spend extensive amounts of time building CI/CD pipelines so that developers can be confident that their latest update does not break anything in production.

The same thing applies to ML. From basic syntax errors to changes in the data stream or unexpected model performance.

In a perfect world, not only does your MLOps tooling support various methods of testing your code and data throughout the pipeline, but you've also adopted a practice to start codifying these tests, starting from the design phase of an ML project.

We will cover different parts of the ML workflow and related testing in later chapters of the book, but here are some general pointers to look out for.

1. Ensure that the data used to train a model looks like you expected it to. There may be changes upstream, for example, how the data is collected or stored.
2. Ensure that the model works not only in training but in a real-world environment. Overfitting is a real risk that should be addressed by your pipeline and process.
3. Ensure that the infrastructure works consistently. Your ML pipeline should produce identical results if the inputs stay the same, and you should always be able to roll back.

Regulatory and Ethical

Machine learning algorithms come under severe regulatory and ethical scrutiny due to not being explicitly defined by a person. When realized, consequences can be financial but maybe even more prevalent is the loss of trust and reputation.

Therefore a data scientist might be looking at frequent governance audits, making many people nervous. Enforcing MLOps best

practices makes reproducibility a priority, and introduces version control for every component of a machine learning model. Think of this as bookkeeping, and when books are in order, there is nothing to worry about. Most importantly, if a prediction a production model has served comes into question, it can always be traced back to how the model was trained.

Most MLOps tooling will introduce version control to your workflow, but that's only half the battle in minimizing regulatory risk. The second half is ensuring that production models are unbiased, and that is generally much trickier as tooling can only provide a framework, but subject matter expertise is required to enforce the right rules.

A prejudiced model likely won't fail from a technical perspective, but it might from a regulatory perspective. Therefore building on the technical tests by including tests that combat bias and other ethical concerns need to be considered in an ML pipeline These concerns vary wildly depending on the application; for example, a healthcare application will have very different ethical concerns to a financial application.

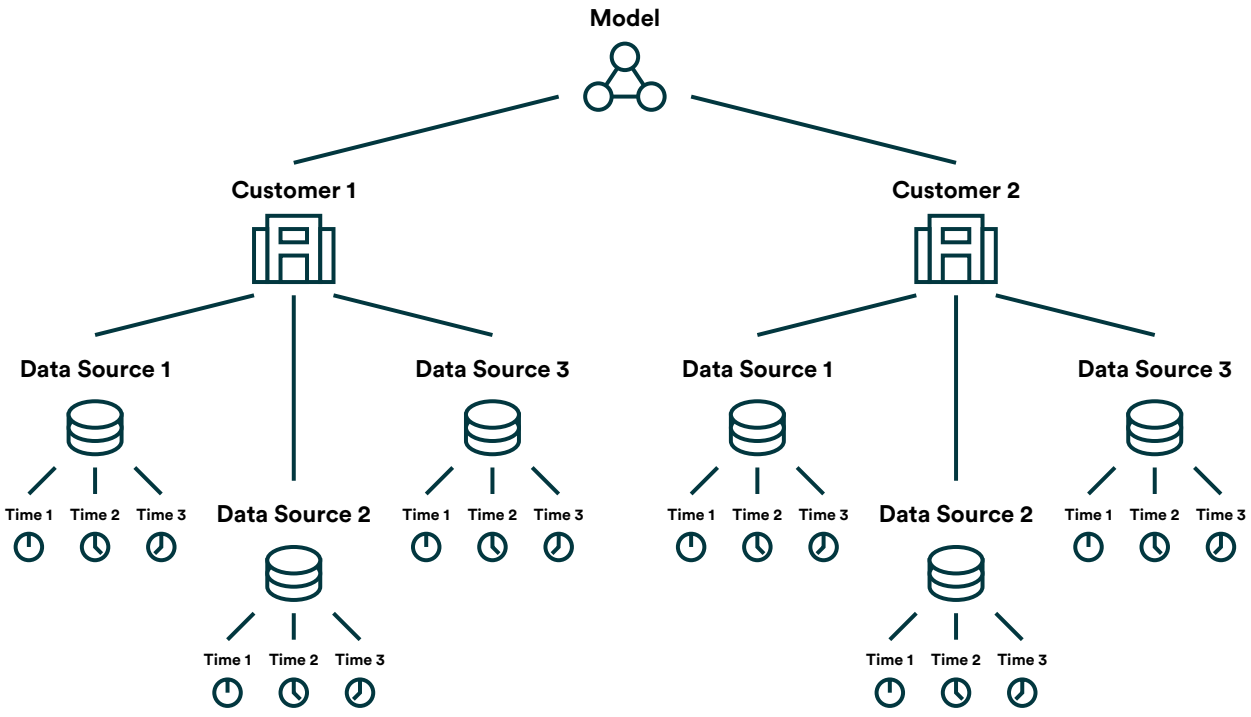
While MLOps can't fix model biases, it comes with the notion that these concerns should be addressed by codifying tests for them into a machine learning pipeline. Checkpoints and safeguards for how the data used for training should look and what the expected predictions should be will make governance audits much less intimidating. All of the production models will be produced in compliance with stable and reliable rules.

Time to Market

It's hard to beat an ad-hoc workflow on speed to market when first developing a machine learning model. For many data scientists and engineers, that is the reason they are hesitant to adopt rigor into their process. However, machine learning in production is rarely about a single version of a single model getting into production but rather a long-term commitment to building machine learning capabilities that

are continuously evolving.

In most cases, you have a lot of dimensionality in model building. First of all, models rarely work over long periods and tend to require retraining somewhat regularly. Secondly, there are often several other dimensions to think about. It is common for models to be, for instance, retrained per customer based on their specific data or by geography, etc. The same model architecture can be running in production as multiple versions of multiple datasets are being re-trained periodically over time. This quickly results in exponential retraining, and ad-hoc workflows will quickly result in chaos, customer dissatisfaction, and poor results. To get a truly scalable business model around ML usually requires that you step back and think of your product in terms of the pipeline instead of a single instance of a trained model. Here's where the most successful companies in ML are today.



18 Trained Models

Broadly speaking, there are two ways that MLOps helps accelerate the time to market of machine learning:

1. It creates a shared language among the extended team.
2. It automates manual tasks.

Shared Language

We have seen a drastic uptake in software development delivery speed and rate over the years. Modern tooling and shared work methods (CI/CD, version control, microservices) have enabled companies to scale their throughput in software development exponentially.

A shared language enables new team members to hit the ground running when you need to accelerate your development. There is very little to suggest that the same requirements for exponential delivery growth would not hold for machine learning too.

Secondly, as discussed earlier, MLOps revolves around turning the process of creating a model into a machine learning pipeline. To achieve this, teams have to codify and componentize their work. While requiring extra effort compared to a single notebook and some manual actions, splitting work into components allows work to scale in a completely different way. This is very similar to what micro services have accomplished in larger development projects.

A single data scientist doesn't need to create the whole model, but rather a team can work on separate pieces such as data preprocessing, model training, and testing. As the complexity of model creation increases, so do the benefits of working around a shared pipeline.

Also, a pipeline consisting of various code pieces is inherently self-documenting – to some extent – and can be more easily handed to the next person than manual actions.

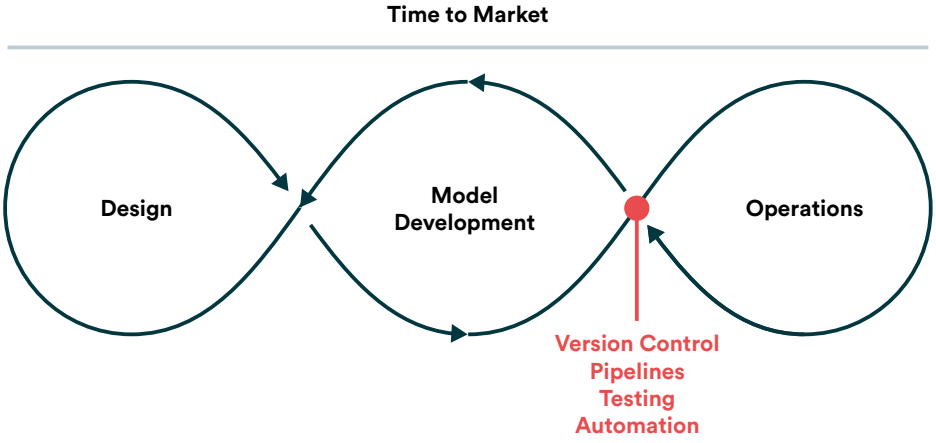
Automation

DevOps has allowed software developers to move from a monthly

or quarterly release cycle to a daily or weekly cycle. Despite the complexities we've explained above with machine learning containing more variables than just code, we should still ensure optimal progress.

Building a CI/CD pipeline for machine learning is more challenging than for traditional software, but the same benefits apply. Automation of collecting data, training, and evaluating models allows the most scarce resources, i.e. data scientists, to focus their efforts on further development. These pipelines are also not throw-away as you can re-use parts when developing a model for a different purpose.

Again the importance varies by application as the benefit of a quicker release cycle is not the same across different use cases. However, in the most extreme cases, your model needs to be updated all the time to keep up with the pace at which the underlying data is changing. This is one reason why some trailblazing companies such as Facebook and Uber had built their own MLOps infrastructure before the term was even coined.



The MLOps Workflow Enforces Best Practices

MLOps ties model development and operations together into a continuous loop through a set of best practices. These best practices can be enforced in many ways, most frequently through a shared platform. To quickly summarize, we identified four best practices that will help you reduce technical friction to get the model from an idea

into production in the shortest possible time to market, with as little risk as possible.

- Version control everything, including models, code, data, parameters, and environment. Enable anyone to trace how a model was produced.
- Componentize the steps of the model creation process and build them into a pipeline. A single notebook is not a pipeline.
- Codify testing. With checkpoints and safeguards in place, there is a standard that models have to adhere to.
- Automate work to increase how much time can be spent on future development.

How to Quantify Success in an MLOps Project?

Quantifying success is not just about the success of the business outcome. In this stage of MLOps maturity, it's also about the success of the process, including multi-disciplinary collaboration and documentation of learning.

> "Only a model that is running in production can bring value."

As we move from projects that experiment with machine learning to projects that are aimed to bring real business value in production systems, we grow the complexity of the project. With it, the uncertainty of success increases. Here are three steps to get started:

1. Define your objective and clear, concrete, and measurable metrics with all the relevant stakeholders
    - Ensure everyone understands the "why" behind the project and the expected benefits it will bring, and educate everyone on the basics of how the ML project will be evaluated. Everyone should know the basics of how a model's accuracy can be measured and what is over-/under-fitting.
    - Document and share how the work is currently done and benchmarked within the team and what it will take to replace the current way of working.
    - Uncover assumptions that different team members have by hosting a session to discover pre-conceptions about the problem and come up with possible solutions. Assumptions could be related to, for example, what data sources should be used, how the data should be processed, or how the model should be published. Try to uncover these assumptions and scope out small experiments to test the hypotheses out, before taking them as facts.

2. Measure the success of different stages of the project - not just the end outcome
    - It will be hard to estimate the total time it will take to deliver on your MLOps project as a whole, and often it's hard to justify the costs of a project that might take 6 or 12 months before we can determine the success of it.
    - Make sure you split your project into smaller pieces to validate your assumptions, track your progress, and gather feedback. The age-old "Push to 'production' often and iterate with feedback" applies here as well. We're not saying to skip long-term engagement, but instead that you should show progress, for example, every six weeks or so.
    - As you go through different stages of the project, make sure you reflect on your progress and whether you're heading in the right direction to solve the original issue. Don't be afraid to go back to defining the problem. Document the lessons you've learned and place them in an internal Wiki for other teams to learn from. These lessons might be about internal data access policy, internal processes that you didn't identify in the beginning, or limitations of your infrastructure.
3. Have a multi-disciplinary team with clear roles and responsibilities
    - As part of your success metrics, you should measure how well your team is working together. Having a multi-disciplinary virtual team for the MLOps project will allow you to effectively tackle surprising blockers that will arise from different parts of your organization.
    - Make sure everyone understands each other's roles and responsibilities in this project. Having this clarity will help the team collaborate and move forward - no one is the "red tape" on purpose.
    - Involve IT early on - they've been around for a while and have a mature infrastructure that you can rely on.
        - IT will want your model to be automated (e.g. automated

pre-processing, training, build, and deployment)

- IT is usually well versed around data compliance and access.

Speak the Same Language

A data scientist's workflow might consist of pulling data from an internal database, exploring with that data, developing a model, and saving it to cloud storage from where others can leverage it. The model's success criteria might primarily consist of model evaluation metrics (e.g., accuracy, precision, recall) based on the business owner's initial specification.

Taking this into production most likely won't happen by copying the model files over to the production server. In most organizations, the path to production will be managed by the IT (or Ops) department, which has been around for a while and has a mature infrastructure with established practices. They will assume that you can automate most (if not all) of your ML pipeline steps, from re-training the model with new data, to comparing it to existing models, and publishing it to production.

Use Every Project as an Opportunity to Educate Your Organization about Machine Learning

Whatever your preference is: company-wide meetings, town halls, or lean coffees, make sure you share learning from your machine learning projects across the organization. These can be either high-level topics or nitty-gritty details on how your organization teams need to collaborate to enable effective collaboration in MLOps projects.

MLOps projects will continue to require contributions from a wider variety of professionals in your organization. Don't wait until the last minute to educate your staff on the MLOps process.

Define Clear Shared Objective and Metrics

After you've uncovered assumptions, make sure you write down and

share the project's objective and the individual metrics used to track its success within the virtual team.

Assumptions

Why does this project exist?
What triggered this project to be started as a machine learning project? What is the expectation in business?

How is performance measured today?
What are we benchmarking against? Is our goal to improve an existing process, or augment or automate a manual process?

How will the deployed model be consumed?
Is the model going to be publicly accessible as an HTTP endpoint? Or is access going to be limited and the model consumed only by internal services?

Who owns the data?
Where is the data stored, and how will it be accessed? How often is the data updated?

Success criteria

Each organization and each project will have it's own custom metrics.

You might focus on improving existing processes, or augmenting a manual process, or fully automating it.

Whatever you're doing, make sure you've listed the outcomes (not outputs) your virtual team should track and what numbers should be reached.

When defining the metrics for your success criteria, think about:

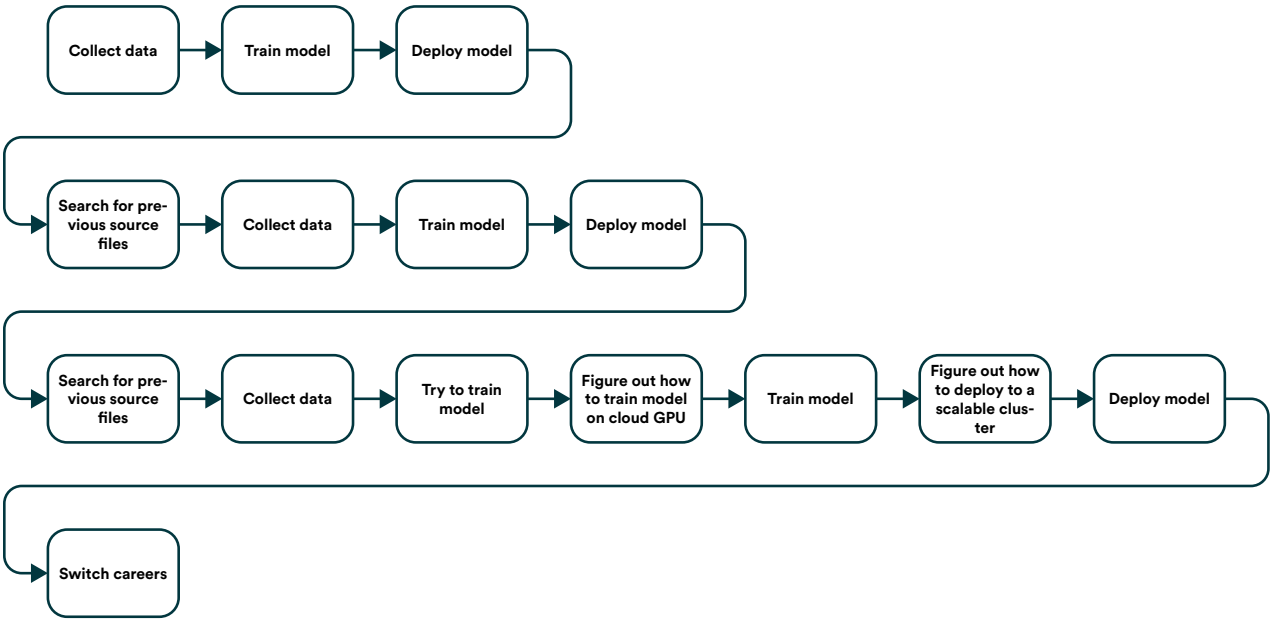
- **What is the key metric that will determine if the model can be taken to production?**
- **How will the model be taken to production?**
- **How often will the model need to be updated?**
- **Model metrics (e.g. accuracy, precision, recall)**
- **Cost to maintain and update the model (and pipeline)**
- **Ethical considerations**

Success criteria	
What's the minimum viable product? Define the bare minimum that will be needed to test the model. This will help you split the project into smaller pieces and "get to a number" faster.	• **Model accuracy x to be able to test in a sandboxed environment** • **Manually train, build and deploy the pipeline.**
MLOps metrics Here you'll find some additional metrics you might want to consider for your MLOps project.	• **Model update frequency met (and able to identify stale models)** • **Time to re-train and deploy a new model and push to production** • **Performance of model endpoint for online predictions (e.g. response time in ms)** • **# of calls / % of failed calls to the model endpoint** • **Collaboration between the multi-disciplinary team (e.g. data scientists, engineers, IT-Ops, legal, business)** • **Attendance of key stakeholders to regular project updates (for example, every six weeks)** • **Infrastructure scales to the machine learning teams without manual work from IT.**

Real-World Example - The Story of Two Companies

The examples presented are fictional, but they illustrate patterns that we've seen repeated over and over in the past five years.

Company 1



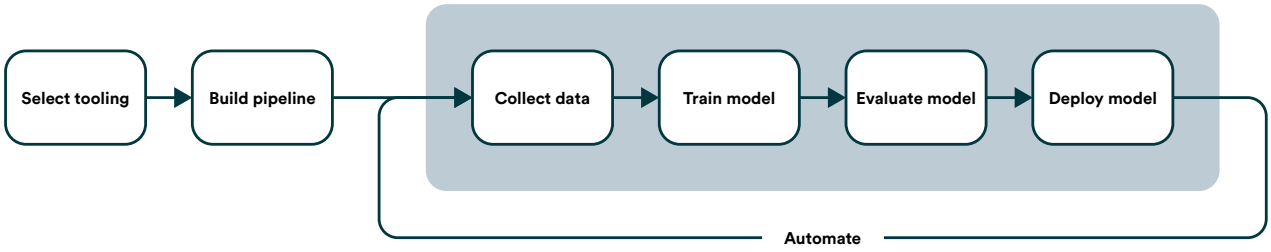
Company 1 jumps right into the nitty-gritty and starts gathering and analyzing data. Their lead data scientist develops a model in Jupyter notebook on their laptop. The model is subsequently deployed to production by a DevOps engineer from another team. From idea to production, the delivery time is blazingly fast.

But trouble starts as time passes; user feedback shows that the predictions the model produces are getting worse. Perhaps the underlying data has changed, and the model needs to be retrained. However, the first time around was poorly documented and done mostly on a single computer. It's hard to reproduce all the steps that it took to train the model – but it gets done.

The company runs through the same firefighting exercise multiple times with different variations. The lead data scientist has left the company, and much of the work on their computer was lost. The model training became too slow to finish on a local machine, and DevOps people had to jump in to solve how to train the model on cloud GPU machines.

Finally, the company decides to put in place tooling to automate retraining and deployment, and at this point, everything is rebuilt.

Company 2



Company 2 acknowledges that machine learning models will be core assets going forward and decides to adopt ML focused tooling before going past the first successful proof-of-concept. The team spends a month upfront in selecting technologies, replacing manual steps (like data collection) with scripts, and building an end-to-end pipeline.

While Company 2 is slower to initial delivery, the benefits start to compound quickly. Everything is automatically reproducible, and data scientists can concentrate on further development rather than firefighting.

Learnings

These two stories illustrate why we don't recommend considering ML workflow as an afterthought. It is a fallacy to think, "We just aren't there yet." The right time to consider tooling and workflow is when the first data scientist is hired, not when the first model goes to production or the prediction serving breaks for the nth time.

Building up an ML workflow does require a strategic commitment and investment from the company. Still, we don't think a management team can afford not to support data scientists with forethought in the current landscape. Machine learning expertise is in high demand and comes with a hefty price tag, and it would be irresponsible to waste any of it.

The MLOps Toolchain

In this chapter, we'll explore what we call the MLOps toolchain. The purpose is to examine areas of MLOps you are likely to need tooling for and how you should work with them. There are many ways to create your own MLOps toolchain, whether you purchase a platform that covers most, if not all, of your needs, use several more specialized tools in combination, or build something custom using open-source tools.

While this eBook doesn't give direct recommendations for tools, we try to help you to be more informed to make the right choices and, more importantly, make the most of the tools you select.

Model and Data Exploration

Exploration is the first concrete, hands-on step of any machine learning project. Thus, it is the most familiar step and one that dominates education, competitions, and other hobby projects. For many data scientists, the exploration step is analogous to a machine learning project. This often leads to confusion when a project advances to productization and deployment as they require different tools and mindset.

Data Exploration

Without data, there can be no model, so data exploration precedes model exploration quite naturally.

The purpose of data exploration is to understand the data. Computers naturally hold all data in a tabular format, which is not natural for us. A raw table of two million rows is not the optimal interface for human understanding. We are great at reading the world in terms of shapes, dimensions, and colors instead. To understand the traits of a single variable or its relationships to other variables, one needs to have tools to analyze the data visually. Understanding the data also leads to transforming and processing the data into higher-level features, which

become valuable in the model exploration step.

Data exploration has the lowest requirements from a technical and DevOps point of view. Fast iterations and visual feedback are the critical components for the data exploration work. Notebooks are a natural fit due to their ability to combine code, visualization, and fast iteration cycles like no other tool out there. Many of the notebooks for this step end up as disposable throwaways. The value is in the insights and discoveries that they have provided, not preserving the environment that produced them. Reproducibility, library dependencies, and version control can bring some value but are often not a hard requirement.

Model Exploration

Model exploration can overlap with the data exploration, but it can be thought of as a separate step. During the model exploration step, the data scientist explores the viability of different models like regression, decision tree, or random forest to the problem at hand. Choosing the model often requires the data scientist to try out and look for optimal parameters (also known as hyperparameters) within the model. Tools exist to find the optimal models and parameters automatically (AutoML). Still, they tend not to work well for more complicated projects and are often not viable at all for the deep learning cases.

Model exploration has higher requirements from the technical and DevOps point of view than data exploration. Single notebooks are still often used, but the dependency with 3rd party libraries, experiment reproducibility, scalable compute infrastructure, and version control becomes much more valuable. Data exploration can usually be done on a local laptop; the model exploration step has much higher computational demands. Testing the viability of a model with a single set of parameters can sometimes take hours, if not days. An auto-scalable cloud environment soon becomes a hard requirement for efficient work. Model exploration costs both time and money, so version control and reproducibility are paramount for all the experiments.

Exploration and MLOps

Considering exploration as a separate throwaway step and not a part of the MLOps pipeline is dangerous. It creates a gap between your data scientists and engineers. Data scientists will simply explore and produce huge, messy proof-of-concept notebooks and expect engineers to do the rest, basically reinventing the wheel. Having a unified pipeline for data scientists and engineers to move from the first experiment to the final productized model is essential. A unified platform for all the project steps creates a common language and understanding from start to finish.

Key Takeaways

1. Data exploration is a lightweight, offline task for understanding data using visualization
2. Model exploration has higher requirements for computing power and version control
3. Disconnecting exploration and MLOps is disconnecting data scientists and engineers

Metrics and Model Optimization

In the previous section, we discussed the ability to visualize and explore data and associated models. Here, we consider how to value different models and efficiently identify which models are the best for production. In particular, how should the hyperparameters of the model be chosen?

Metrics are the quantities which define the success or failure of a model. How many fraudulent credit card transactions will be identified before being approved? How many viewers will respond to a given advertisement? How much will it cost to purchase steel in six months?

Defining metrics is only half the battle, though. Before putting a model into production, the data scientist must find a model which satisfies necessary performance expectations on the key metrics. A model's performance is strongly impacted by its hyperparameters (also called free parameters) such as learning rate, nodes per layer, or maximum number of estimators.

Optimization Vocabulary

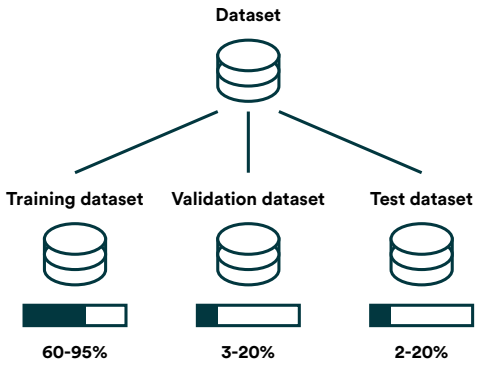
In software engineering (algorithms), optimizing a function means rewriting it to be faster or smaller, which is an entirely different thing and sometimes causes confusion.

In data science (mathematical), optimizing a function means changing the input to minimize (or maximize) the output. A loss function is the formula for the model's error, and when someone says they are optimizing the loss function, they are actually minimizing the model's error.

The act of optimizing the loss function is often called training. In training, you repeatedly feed the model training data, adjusting it in tiny steps, working toward smaller and smaller error. The error during training is called training loss.

Training/Validation/Test Datasets

In a production setting, it is common to divide all the available data into three different datasets: training, validation and test datasets.



These datasets are usually sampled in some stratified sense, to consistently represent the entire population.

- Training dataset - Between 60-95% of the available data. Used to compute the training loss for a model – minimizing the training loss finds the parameters of a model (e.g., random forest splits or neural network weights).
- Validation dataset - Between 3-20% of the available data. Used to define validation metrics for a model – optimizing the validation metrics finds the hyperparameters of a model (e.g. SGD momentum/dropout, RF minimum split fraction, SVM box constraint).
- Test dataset - Between 2-20% of the available data. Used to compute the validation metrics on a separate dataset. After model tuning, provides an unbiased estimate of how the validation metrics will perform in production.

Validation metrics are the key tool in systematically identifying the best hyperparameter choices. If hyperparameters are tuned by minimizing the training loss, the model will likely be overfit to the training data. Models will be better able to generalize to unseen data when a separate validation set is used to choose the hyperparameters.

Validation Metrics for Tuning and Production

Validation metrics are so-called because they help validate the production performance of the model. These metrics should represent the success of the model while it is being used. They can mirror those found in an ML classroom, or they can be entirely unique to a specific business case. Below, we give some examples.

Consider an image classification model for identifying vehicles as either bicycles, cars, trucks or something else. A simple metric might be the fraction of correct classifications (i.e. validation accuracy). If this is meant to be used in a vehicle to guide the driver, an important metric might be the fraction of misclassifications of bicycles (for rider safety). If this model is deployed in an automated toll, maybe correct classification of trucks would be much more important than other vehicles. If the model is used for opening or closing a security gate, maybe the inference time of the model would be very important.

Optimizing Metrics and Decision Making

Production ML pipelines require hyperparameter optimization, sometimes called model tuning, to be conducted efficiently so that models can be deployed in a timely fashion. Unfortunately, each choice of hyperparameters requires a new model training, which can make model tuning very costly. Furthermore, validation metrics almost always lack gradient information, meaning that the standard tools used for minimizing the training loss cannot be used for optimizing the validation metrics.

Bayesian optimization is a popular tool for model tuning: it requires no gradient information, can optimize noisy metrics and can work with categorical parameters (such as choosing between Adam, Adagrad and RMSProp). The core structure involves an iteration where hyperparameters are suggested, validation metrics are computed, those values are reported to the optimization loop, and new hyperparameters are suggested based on these new results (to continue the iteration).

In most production settings, multiple metrics define success. In the computer vision example earlier, the model may need to balance high accuracy (a larger, more complicated model) with low inference time (a smaller, simpler model). In financial trading situations, models may need to balance high return against low risk and high liquidity (however those may be defined). A fraud detection model may be interested in maximizing sales (allow more transactions to proceed) and minimizing loss (decline more transactions).

Multimetric optimization explores the tradeoff between competing metrics. This allows modelers to understand the balance of the key metrics and choose hyperparameters which meet their needs. Additionally, metric thresholds, or metric constraints, can be applied when certain metrics must meet minimum performance thresholds of viability – maybe the vision model should maximize accuracy but only for models with less than 100ms inference time.

After completing a model optimization with one or more metrics, the test dataset should be used to reevaluate the metrics on any hyperparameters being considered for production. This will confirm the generalizability of the model before it is used in the real world.

Key Takeaways

1. Split available data into training, validation and test datasets.
2. Define and study metrics which represent success in production, not just during training.
3. Identify the best hyperparameters for your metrics with as little tuning cost as possible.

This chapter was authored by SigOpt. Organizations across a wide range of industries trust SigOpt to solve their toughest optimization challenges.

Productionalization - End-to-End Pipelines

In a typical setting, the data scientist starts with a laptop, a static dataset, and a problem to solve.

> Is the client going to churn? Is there a nuclear missile silo in the satellite photo? What word is going to be typed next?

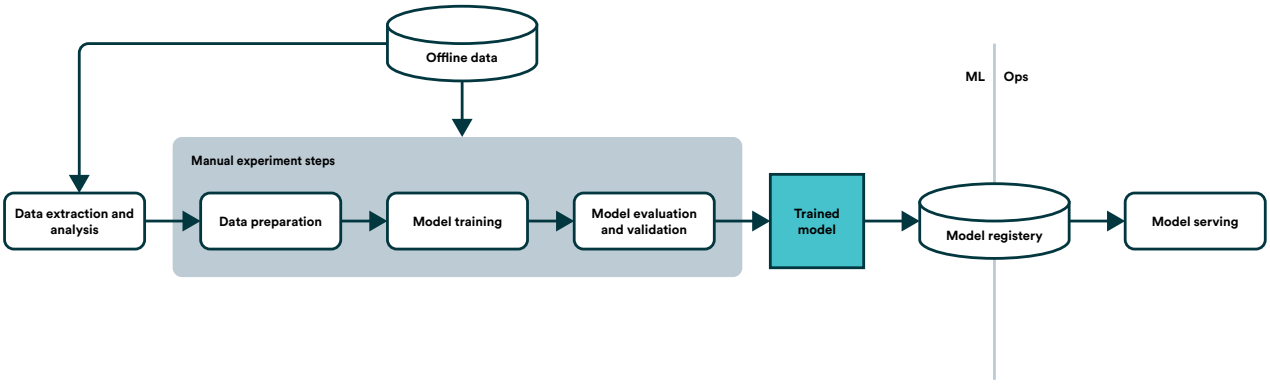
To solve the problem, the data scientist takes a notebook, starts to sanitize the dataset, transforms it into features, installs a dozen libraries, writes code, deletes code, tries different models, changes hyperparameters, drinks a lot of coffee, and finally comes up with a giant notebook that finally solves the problem. On their laptop. Today.

Productionalization for ML is taking that problem-solving capability - only existing in the data scientist's laptop today - and refining it into an accessible and scalable system. One that provides value at scale and can be nurtured to keep updating and improving for years to come.

The Manual Cycle

In the manual workflow, where no real infrastructure exists, the data scientist hands over the huge notebook to the ML engineer.

Adapted from Google's article, MLOps: Continuous delivery and automation pipelines in machine learning



The engineer manually cleans up the code, and refactors it for performance and integration with the production environment. The engineer then figures out the required libraries and environment and where and how to get the live data, sets up the deployment endpoint, and finally pushes the model. In this workflow, the model is the product.

Time goes by, and somebody alerts the team that the model might be misbehaving. The original problem-solving capability seems to be declining based on a gut feeling. Nobody knows for sure. The data scientist manually grabs a new batch of offline data, starts the coffee machine, and the entire manual cycle is ready to start all over again.

Characteristics of a manual ML pipeline:

- The model is the product
- Manual or script-driven process
- A disconnect between the data scientist and the engineer
- Slow iteration cycle
- No automated testing or performance monitoring
- No version control

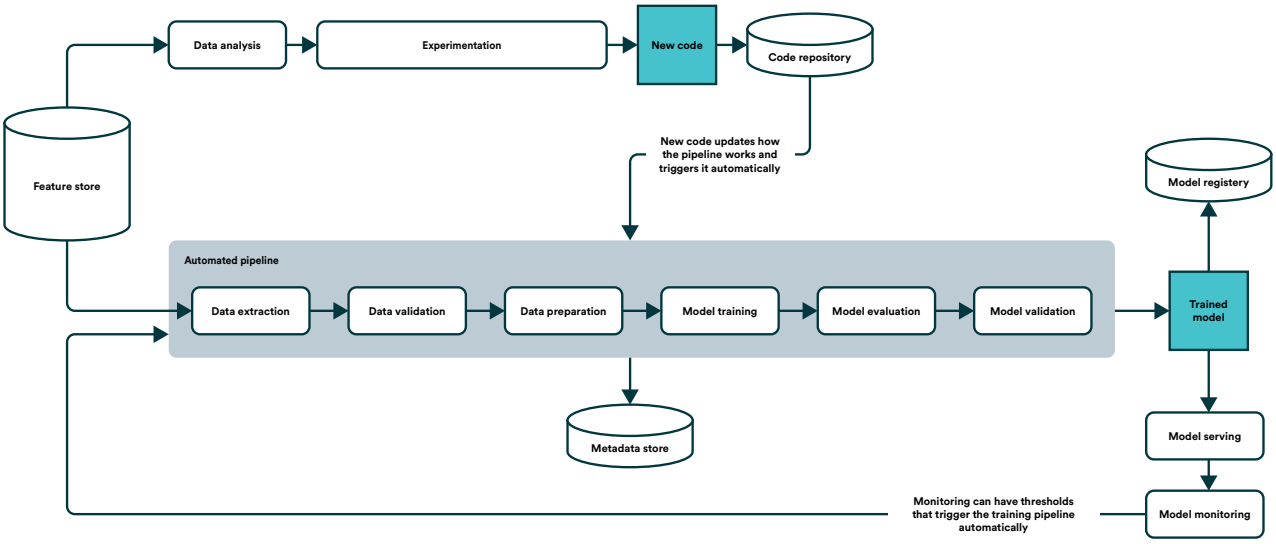
The Automated Pipeline

In the automated pipeline workflow, you don't build and maintain a model. You build and maintain a pipeline. The pipeline is the product.

An automated pipeline consists of components and a blueprint for how those are coupled to produce and update the most crucial component – the model.

In the automated workflow, the data scientist doesn't aim for a gigantic notebook on her laptop with offline data. While she may initially start with a single notebook, she will eventually split the problem-solving into more manageable components.

Adapted from Google's article, MLOps: Continuous delivery and automation pipelines in machine learning



Examples of different components:

- Data validation
- Data cleanup
- Training
- Model evaluation
- Model validation
- Re-training trigger

In addition, the pipeline also has static components like:

- Feature store
- Deployment endpoint
- Metadata store
- Source code version control

The system offers the ability to execute, iterate, and monitor a single component in the context of the entire pipeline with the same ease and rapid iteration as running a local notebook cell on a laptop. It also lets you define the required inputs and outputs, library dependencies, and monitored metrics.

This ability to split the problem solving into reproducible, predefined and executable components forces the team to adhere to a joined

process. A joined process, in turn, creates a well-defined language between the data scientists and the engineers and also eventually leads to an automated setup that is the ML equivalent of continuous integration (CI) – a product capable of auto-updating itself.

Characteristics of an automated ML pipeline:

- The pipeline is the product
- Fully automated process
- Co-operation between the data scientist and the engineer
- Fast iteration cycle
- Automated testing and performance monitoring
- Version-controlled

Key Takeaways

1. The pipeline is the product, not the model. Do not deploy the model; deploy the pipeline.
2. To build a pipeline, split the system down into small well-defined components.
3. Model accuracy will eventually degrade as the world changes. Prepare for it.

Productionalization - Feature Stores

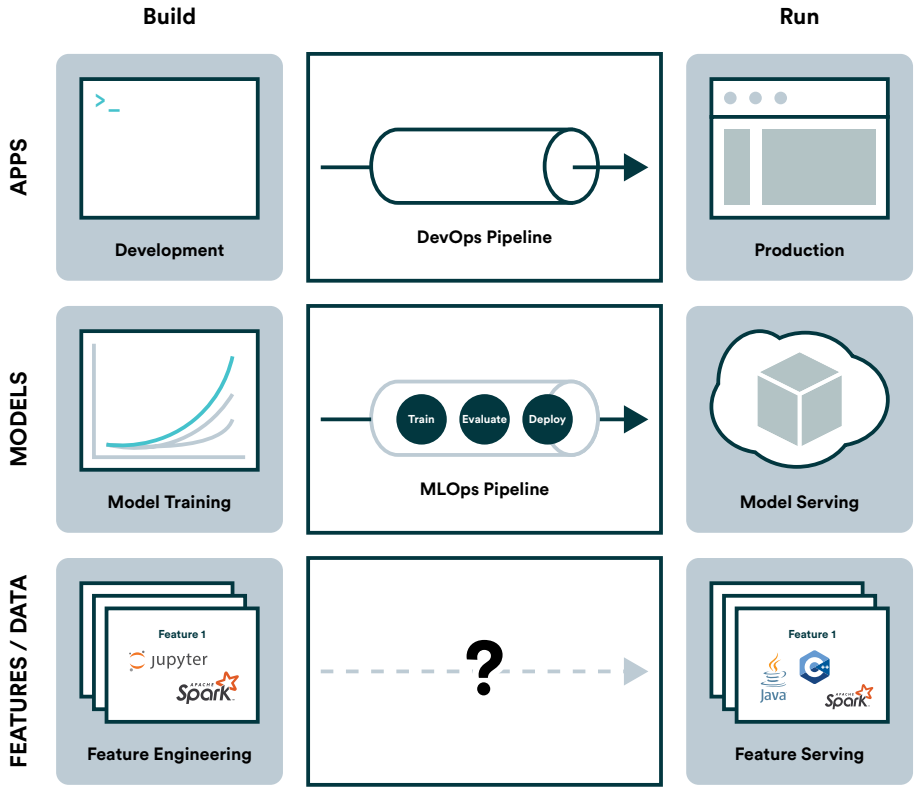
In the previous section, we talked about machine learning pipelines with a focus on the model. Features are, however, another critical piece for ML in production and are similarly complex to manage as models.

When building traditional applications, developers really only need to get code to production. And we have mature DevOps tooling and processes to do that quickly and efficiently. Developers can now push code out to production multiple times a day.

But when it comes to building ML-powered applications, we now also have to get models and features to production. And that creates additional headaches. Data scientists are not software engineers. They build models and features in their individual notebooks. How do these models and features then make it to production?

For models, there are tools for building machine learning pipelines and managing the model lifecycle. MLOps platforms allow data scientists to train models, run experiments, validate models, and finally deploy them to production.

For features, the tool offering has been much less mature. Getting features to production is particularly hard because we need to get feature transformations (or pipelines) to production, and curate the feature values to serve consistent data for training and online inference. Data scientists typically pass their feature transformations to data engineers to re-implement the feature pipelines with production-hardened code. That's a complicated process that typically introduces weeks or months of lead time.



Tooling for managing features is almost non-existent

The Feature Store

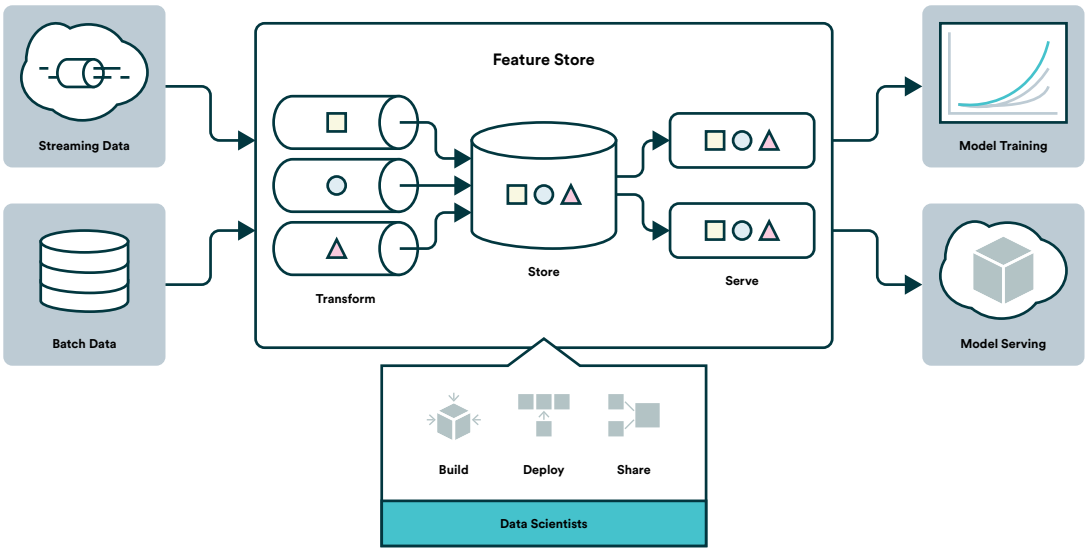
This is where feature stores come in. Feature stores are central hubs for features. They transform raw data into feature values, store the values, and serve them for model training and online predictions. By automating these steps, feature stores allow data scientists to build and deploy features within hours instead of months. They enable data scientists to fully control their features from development to production, bringing DevOps-like tooling to the feature engineering process.

Feature stores allow data scientists to:

- Build a library of great features collaboratively. Instead of managing feature transformations in a local notebook, data scientists create standard feature definitions that are managed in a Git-backed repository. These feature definitions are then applied to the feature

store. This brings consistency to feature definitions, and the ability to collaborate on new features.

- Deploy features to production instantly. Once feature definitions are applied to the feature store, it automates the feature transformations and curates the feature values. Those values can be used to create training datasets or can be served online for real-time inference.
- Share, discover, and re-use features. Features and their metadata, transformation logic, and values, are all managed in a central feature registry and are searchable. Data scientists can easily discover existing features and re-use them across models.



Feature Store: Interface Between Models and Data

Feature stores were first introduced by the Uber Michelangelo team. Since then, many companies like Airbnb and Netflix have built their own internal feature stores to solve this problem. But feature stores are also complicated to build, and have to a large extent remained inaccessible to less advanced organizations.

In the past year, however, we've seen the introduction of several open source and commercial feature stores. They integrate with existing data lakes, data warehouses, event streaming platforms, processing

engines, pipeline orchestrators, and ML platforms to augment the infrastructure with feature management capabilities.

Key takeaways

1. Building features and getting them to production is one of the hardest parts of productionizing ML.
2. Feature stores allow data scientists to build, deploy, and share features quickly and easily.
3. Feature stores complement existing ML infrastructure to bring DevOps-like capabilities to the feature lifecycle.

tecton

This chapter was authored by Tecton. They provide an enterprise-ready feature store to make world-class machine learning accessible to every company.

Testing

Traditional software is built by writing fixed rules against well-defined static assumptions of the surrounding world. It is relatively straightforward to test every single rule (unit test) or group (integration tests) when everything is defined in advance.

Machine Learning, by definition, is about dynamically finding the rules, based on constantly changing data, which makes testing much more difficult. Testing in ML is like trying to hit a moving target. The system's behavior depends on the data's dynamic qualities and the various model configuration choices.

Testing is tightly linked with exploration as it should inform you what the criteria are; for example, in regards to statistical distributions.

Data Testing

For an ML project, data is as (if not more) important than the code. Like the unit tests for your code define and test your assumptions about the inputs, your data validation tests should do the same for training and inference input data. You should test for null values, abnormal statistical distributions within a feature, and the relationships between features.

For example, if your input is expected to be random English, one way to test it is to calculate that 'the' is the most commonly occurring word, as that is a known assumption. If some other word appears more often than 'the', it probably means that your data has some unexpected bias, or perhaps some of it is accidentally in Spanish. Another example of validating your assumptions could be that you make sure your input data has an even split between male and female data if that is a valid assumption for your case.

You also need to test for relationships between features. If two or more features are highly correlated, it can have a degrading effect on the model's performance and accuracy. For example, if your data

is about products and has two similar columns for the price: taxes included, and taxes excluded, it should trigger an alarm.

Model Testing

Once the assumptions about data are tested, we can move on to test the models and their training. You shouldn't just blindly deploy a model that shows promising accuracy for your offline data and hope for the best.
In the training phase, you can test the impact of each hyperparameter. Running a grid-search or more advanced search strategy can uncover reliability issues and also improve predictive performance. Using an additional test set, disjoined from training and validation sets should also be used whenever possible.

When deploying the model, test the relationship between your offline metrics and the actual impact of the model in the real world. Correlation between your offline accuracy and the real click-through rate on your website can be measured in a small scale A/B experiment.

Another viable smoke test can be testing your new shiny model against a simple baseline model. Trickle small amounts of live production data to be handled by your new model as a canary test before you fully commit.

Infrastructure Testing

Your ML pipeline should be as reproducible as possible from one day to the next. Perfect determinism is hard, but you should work towards it.

Test the reproducibility of your training pipeline by training two or more models side-by-side with the same data and measure any discrepancies between metrics. Also, test things like the ability to continue training predictably from a mid-crash checkpoint. Don't forget to create integration smoke tests for your entire pipeline, all the way from first data validation down to model deployment. These sorts of tests should run continuously and during the deployment of a

new model version.

Rolling back to a stable model can also be vital in unexpected circumstances or when human error occurs. You should continuously test your rollback infrastructure, as it is your last line of defense when all other tests have failed you.

Key Takeaways

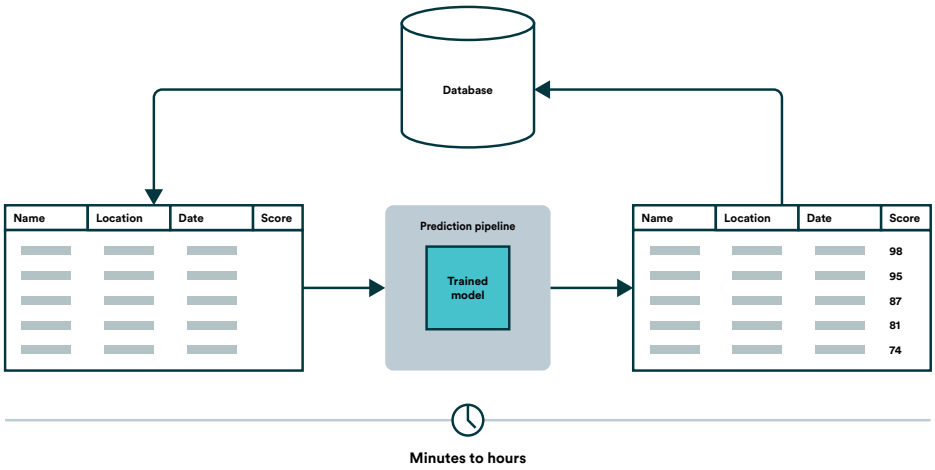
1. Due to the dynamic nature of ML, testing is even more critical.
2. Testing code is good; testing data is paramount.
3. Reproducibility of the pipeline is the key to safe deployment.

Deployment and Inference

Deployment is the act of serving an ML model to the rest of the world via API, application, or otherwise. Inference is what the model does, once it is deployed. Whether it is making predictions, classifying input, or clustering data, it is always referred to as inference.

The question one should always ask first is whether you aim for batch inference or online inference; followed by whether to do classic centralized cloud inference or distribute the compute requirements to your customer's hardware (edge inference).

Batch Inference

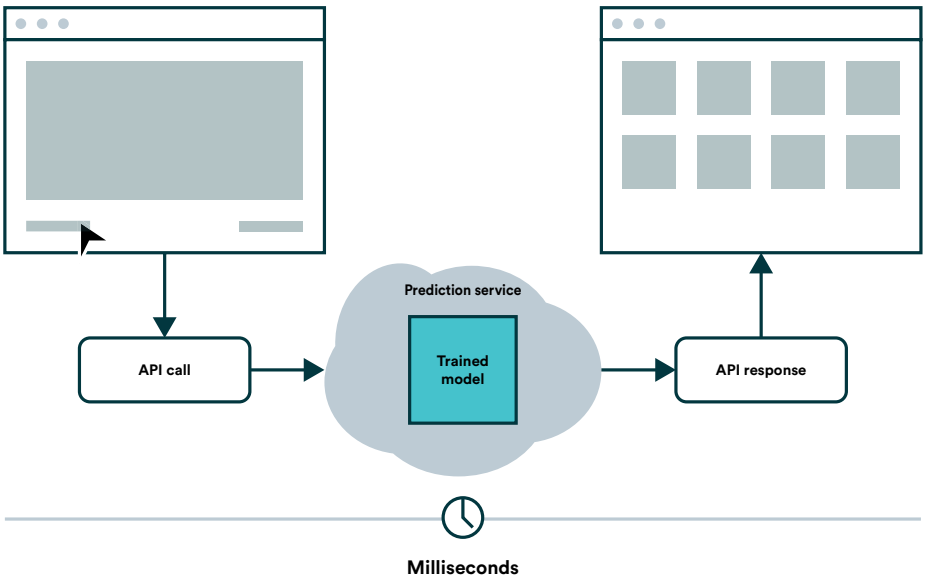


The batch inference is the process of making inference for a batch of requests. Instead of providing value instantly in real-time for each request, batch inference provides answers to a set of questions later. The batch is often processed after a fixed interval, for example, once a day. Batch inference, at its core, is close to a classic caching strategy in software development.

Batch inference suits any scenario where latency is not an issue. For example, if your model needs to score all the new leads to the sales team, it probably doesn't matter if there is a 24-hour latency in scoring.

In the MLOps context, the value of inference in fixed intervals allows engineers to parallelize more efficiently and use a considerable amount of computing power more predictably. The infrastructure exposed to the outside world is much simpler to maintain and monitor, as there is no need to connect your complex model directly into a live online API with unpredictable traffic patterns. You also get some time to act on the problems and fix them before any are shown to the customer.

Online Inference



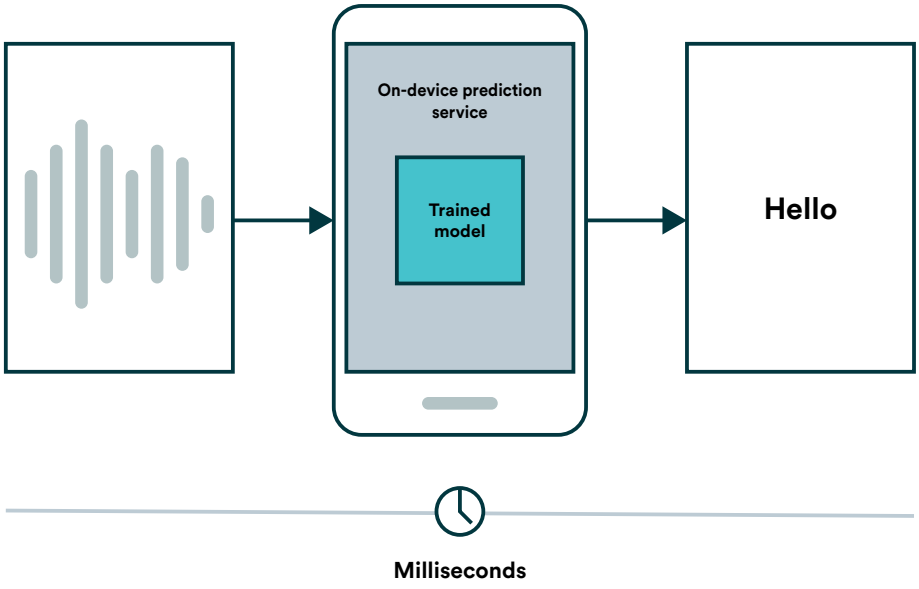
The online inference is the process of making inference in real-time. Every request is handled by the model right away.

Online inference suits any situation where the value provided by the model is needed right away. For example, if the model is supposed to predict the best route for an ambulance to take or the stock market's volatility for the next ten minutes, every second counts. The prediction has no value if it's late.

In the MLOps context, online inference is much more demanding. As the requests are handled right away, and the model is directly connected to live and unpredictable online traffic, things can go wrong, and they can go wrong fast. Any error or bias in the prediction

leaks back to the customer right away. The system also needs to be automatically scalable to accommodate the peaks in traffic. Typically, scaling is handled by using an auto-scalable cluster like Kubernetes. Requirements for monitoring are much higher, and reaction time for any intervention needs to be close to zero.

Edge Inference



The classic option for deployment and inference is to use a centralized system. Typically, you configure a Kubernetes cluster on one of the cloud providers to handle your model's requests. Another modern option is to harness the computing power of your customer's hardware. This is called the edge inference.

Instead of having an app consume a central cloud API, you deploy the model as part of your application directly in the user's device or browser. A good example would be the speech-to-text model running on a mobile phone. Audio isn't sent back to a server (high bandwidth and latency requirements), but rather, the model on the phone can transform the audio as text right away. The transcribed text can be used for a more straightforward text request (low bandwidth and latency) back to a server, where another model is explicitly trained to make sense of the transcribed text input.

Edge inference has an obvious upside: Perfect scaling comes for free as the more requests you get, the more edge devices you have at your disposal. Often, the cloud service doesn't even need to know that inference has happened as the model on the edge device can be fully autonomous in this sense. The downside is that it might become harder to maintain all the different versions out there. Depending on your update mechanism, you might need to prepare for a wide variety of versions of your model running in production simultaneously.

You also need to face the problem of different devices and environments. A thousand mobile phones equal a thousand slightly different environments. Depending on how well you encapsulate your edge inference from the surrounding environment, you might need to invest in a complex testing infrastructure when dealing with edge devices.

Another potential downside is security. While you are not sending all your data to external devices, you are sending a model that is trained with that data. While it is usually very hard to exploit, it is still a security concern to consider.

Key Takeaways

1. Use batch inference where possible. Online inference should be a last resort.
2. Use edge inference where possible as it means perfect scaling for free.
3. Complex setups need robust monitoring.

Conclusion

Investing in machine learning will enable you to solve business cases that were previously impossible to solve, for example, automatically categorizing images by their contents. Contrary to ML, MLOps doesn't come with a promise to solve any business problems directly. Instead, it comes with the promise to accelerate how your investments in ML return value.

To make an analogy to a more traditional industry: machine learning is shipping goods while MLOps is containerization. And much like containerization of global shipping, MLOps is process and infrastructure in equal parts. MLOps won't yield immediate results, and it'll likely require commitment from a broad range of stakeholders. Still, the process will deliver increasing benefits with the scale of your machine learning efforts.

For some, the MLOps practice is nothing new, especially for data scientists with a strong software engineering background. In collaborative efforts, however, it's often essential to emphasize that we work by best practices.

The four critical best practices we suggest you adopt are:

- Versioning to ensure reproducibility of models
- Pipelines to build better systems collaboratively
- Testing to set standards for your production models
- Automation to save time and build towards self-healing systems

There are many different choices for MLOps tooling out on the market and many different strategies to setting up your whole MLOps toolchain, from building your own to buying a managed end-to-end platform. In this eBook, we've tried to highlight the areas you should consider and give some key takeaways when deciding what you need.

Ultimately, the goal of MLOps is to reduce technical friction to get the model from an idea into production in the shortest possible time, and then to market with as little risk as possible, and you should judge tooling decisions against that goal.

About the Authors

Train, Evaluate, Deploy, Repeat. Valohai is the only MLOps platform that automates everything from data extraction to model deployment. Valohai is used by everyone from startups to enterprises, including PARC, LEGO Group, Twitter and JFrog.

For more, visit www.valohai.com.

SigOpt offers the most complete MLOps platform for experiment management and model optimization to scale the AI model development process.

For more, visit www.sigopt.com.

Built by the creators of Uber Michelangelo, Tecton provides the first enterprise-ready feature store that manages the complete lifecycle of features — from engineering new features to serving them online for real-time predictions.

For more, visit www.tecton.ai.