Engineering MLOps

Rapidly build, test, and manage production-ready machine learning life cycles at scale

Emmanuel Raj

Engineering MLOps

Group Product Manager: Kunal Parikh
Publishing Product Manager: Aditi Gour
Senior Editor: Mohammed Yusuf Imaratwale
Content Development Editor: Nazia Shaikh
Technical Editor: Arjun Varma
Copy Editor: Safis Editing
Project Coordinator: Aishwarya Mohan
Proofreader: Safis Editing
Indexer: Priyanka Dhadke
Production Designer: Joshua Misquitta

First published: April 2021
Production reference: 1160421

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-80056-288-2
`www.packt.com`

Table of Contents

5
Model Evaluation and Packaging

12
Model Serving and Monitoring

13
Governing the ML System for Continual Learning

Preface

MLOps is a systematic approach to building, deploying, and monitoring machine learning (ML) solutions. It is an engineering discipline that can be applied to various industries and use cases. This book presents comprehensive insights into MLOps coupled with real-world examples to help you to write programs, train robust and scalable ML models, and build ML pipelines to train and deploy models securely in production.

You will begin by familiarizing yourself with MLOps workflow and start writing programs to train ML models. You'll then move on to explore options for serializing and packaging ML models post-training to deploy them in production to facilitate machine learning inference. Next, you will learn about monitoring ML models and system performance using an explainable monitoring framework. Finally, you'll apply the knowledge you've gained to build real-world projects.

By the end of this ML book, you'll have a 360-degree view of MLOps and be ready to implement MLOps in your organization.

Who this book is for

This MLOps book is for data scientists, software engineers, DevOps engineers, machine learning engineers, and business and technology leaders who want to build, deploy, and maintain ML systems in production using MLOps principles and techniques. Basic knowledge of machine learning is necessary to get started with this book.

What this book covers

Chapter 1, *Fundamentals of MLOps Workflow*, gives an overview of the changing software development landscape by highlighting how traditional software development is changing to facilitate machine learning. We will highlight some daily problems within organizations with the traditional approach, showcasing why a change in thinking and implementation is needed. Proceeding that an introduction to the importance of systematic machine learning will be given, followed by some concepts of machine learning and DevOps and fusing them into MLOps. The chapter ends with a proposal for a generic workflow to approach almost any machine learning problem.

Chapter 2, *Characterizing Your Machine Learning Problem*, offers you a broad perspective on possible types of ML solutions for production. You will learn how to categorize solutions, create a roadmap for developing and deploying a solution, and procure the necessary data, tools, or infrastructure to get started with developing an ML solution taking a systematic approach.

Chapter 3, *Code Meets Data*, starts the implementation of our hands-on business use case of developing a machine learning solution. We discuss effective methods of source code management for machine learning, data processing for the business use case, and formulate a data governance strategy and pipeline for machine learning training and deployment.

Chapter 4, *Machine Learning Pipelines*, takes a deep dive into building machine learning pipelines for solutions. We look into key aspects of feature engineering, algorithm selection, hyperparameter optimization, and other aspects of a robust machine learning pipeline.

Chapter 5, *Model Evaluation and Packaging*, takes a deep dive into options for serializing and packaging machine learning models post-training to deploy them at runtime to facilitate machine learning inference, model interoperability, and end-to-end model traceability. You'll get a broad perspective on the options available and state-of-the-art developments to package and serve machine learning models to production for efficient, robust, and scalable services.

Chapter 6, *Key Principles for Deploying Your ML System*, introduces the concepts of continuous integration and deployment in production for various settings. You will learn how to choose the right options, tools, and infrastructure to facilitate the deployment of a machine learning solution. You will get insights into machine learning inference options and deployment targets, and get an introduction to CI/CD pipelines for machine learning.

Chapter 7, *Building Robust CI and CD Pipelines*, covers different CI/CD pipeline components such as triggers, releases, jobs, and so on. It will also equip you with knowledge on curating your own custom CI/CD pipelines for ML solutions. We will build a CI/CD pipeline for an ML solution for a business use case. The pipelines we build will be traceable end to end as they will serve as middleware for model deployment and monitoring.

Chapter 8, *APIs and Microservice Management*, goes into the principles of API and microservice design for ML inference. A learn by doing approach will be encouraged. We will go through a hands-on implementation of designing and developing an API and microservice for an ML model using tools such as FastAPI and Docker. You will learn key principles, challenges, and tips to designing a robust and scalable microservice and API for test and production environments.

Chapter 9, *Testing and Securing Your ML Solution*, introduces the core principles of performing tests in the test environment to test the robustness and scalability of the microservice or API we have previously developed. We will perform hands-on load testing for a deployed ML solution. This chapter provides a checklist of tests to be done before taking the microservice to production release.

Chapter 10, *Essentials of Production Release*, explains how to deploy ML services to production with a robust and scalable approach using the CI/CD pipelines designed earlier. We will focus on deploying, monitoring, and managing the service in production. Key learnings will be deployment in serverless and server environments using tools such as Python, Docker, and Kubernetes.

Chapter 11, *Key Principles for Monitoring Your ML System*, looks at key principles and aspects of monitoring ML systems in production for robust, secure, and scalable performance. As a key takeaway, readers will get a concrete explainable monitoring framework and checklist to set up and configure a monitoring framework for their ML solution in production.

Chapter 12, *Model Serving and Monitoring*, explains serving models to users and defining metrics for an ML solution, especially in the aspects of algorithm efficiency, accuracy, and production performance. We will deep dive into hands-on implementation and real-life examples on monitoring data drift, model drift, and application performance.

Chapter 13, *Governing the ML System for Continual Learning*, reflects on the need for continual learning in machine learning solutions. We will look into what is needed to successfully govern an ML system for business efficacy. Using the Explainable Monitoring framework, we will devise a strategy to govern and we will delve into the hands-on implementation for error handling and configuring alerts and actions. This chapter will equip you with critical skills to automate and govern your MLOps.

To get the most out of this book

You should have access to a Microsoft Azure subscription and basic DevOps-based software used to build CI/CD pipelines. A personal computer or laptop with a Linux or macOS is a plus.

Software/hardware covered in the book	**OS requirements**
Python	Linux, macOS, or Windows
Git	Linux, macOS, or Windows
Docker	Linux, macOS, or Windows
Kubernetes	Linux, macOS, or Windows
Microsoft Azure	Linux, macOS, or Windows
Azure ML Service	Linux, macOS, or Windows
MLFlow	Linux, macOS, or Windows
Azure DevOps	Linux, macOS, or Windows
Fast API	Linux, macOS, or Windows
Locust.io	Linux, macOS, or Windows

If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/EngineeringMLOps`. In case there's an update to the code, it will be updated on the existing GitHub repository. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781800562882_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "The preprocessed dataset is imported using the .get_by_name() function."

A block of code is set as follows:

```
uri = workspace.get_mlflow_tracking_uri( )
mlflow.set_tracking_uri(uri)
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
# Importing pre-processed dataset
dataset = Dataset.get_by_name (workspace, name='processed_
weather_data_portofTurku')
```

Any command-line input or output is written as follows:

```
python3 test_inference.py
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Go to the **Compute** option and click the **Create** button to explore compute options available on the cloud."

Tips or important notes

Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

Section 1: Framework for Building Machine Learning Models

This part will equip readers with the foundation of MLOps and workflows to characterize their ML problems to provide a clear roadmap for building robust and scalable ML pipelines. This will be done in a learn-by-doing approach via practical implementation using proposed methods and tools (Azure Machine Learning services or MLflow).

This section comprises the following chapters:

1
Fundamentals of an MLOps Workflow

Machine learning (**ML**) is maturing from research to applied business solutions. However, the grim reality is that only 2% of companies using ML have successfully deployed a model in production to enhance their business processes, reported by DeepLearning.AI (`https://info.deeplearning.ai/the-batch-companies-slipping-on-ai-goals-self-training-for-better-vision-muppets-and-models-china-vs-us-only-the-best-examples-proliferating-patents`). What makes it so hard? And what do we need to do to improve the situation?

To get a solid understanding of this problem and its solution, in this chapter, we will delve into the evolution and intersection of software development and ML. We'll begin by reflecting on some of the trends in traditional software development, starting from the waterfall model to agile to DevOps practices, and how these are evolving to industrialize ML-centric applications. You will be introduced to a systematic approach to operationalizing AI using **Machine Learning Operations** (**MLOps**). By the end of this chapter, you will have a solid understanding of MLOps and you will be equipped to implement a generic MLOps workflow that can be used to build, deploy, and monitor a wide range of ML applications.

In this chapter, we're going to cover the following main topics:

- The evolution of infrastructure and software development
- Traditional software development challenges
- Trends of ML adoption in software development
- Understanding MLOps
- Concepts and workflow of MLOps

The evolution of infrastructure and software development

With the genesis of the modern internet age (around 1995), we witnessed a rise in software applications, ranging from operating systems such as Windows 95 to the Linux operating system and websites such as Google and Amazon, which have been serving the world (online) for over two decades. This has resulted in a culture of continuously improving services by collecting, storing, and processing a massive amount of data from user interactions. Such developments have been shaping the evolution of IT infrastructure and software development.

Transformation in IT infrastructure has picked up pace since the start of this millennium. Since then, businesses have increasingly adopted cloud computing as it opens up new possibilities for businesses to outsource IT infrastructure maintenance while provisioning necessary IT resources such as storage and computation resources and services required to run and scale their operations.

Cloud computing offers on-demand provisioning and the availability of IT resources such as data storage and computing resources without the need for active management by the user of the IT resources. For example, businesses provisioning computation and storage resources do not have to manage these resources directly and are not responsible for keeping them running – the maintenance is outsourced to the cloud service provider.

Businesses using cloud computing can reap benefits as there's no need to buy and maintain IT resources; it enables them to have less in-house expertise for IT resource maintenance and this allows businesses to optimize costs and resources. Cloud computing enables scaling on demand and users pay as per the usage of resources. As a result, we have seen companies adopting cloud computing as part of their businesses and IT infrastructures.

Cloud computing became popular in the industry from 2006 onward when Sun Microsystems launched Sun Grid in March 2006. It is a hardware and data resource sharing service. This service was acquired by Oracle and was later named Sun Cloud. Parallelly, in the same year (2006), another cloud computing service was launched by Amazon called Elastic Compute Cloud. This enabled new possibilities for businesses to provision computation, storage, and scaling capabilities on demand. Since then, the transformation across industries has been organic toward adopting cloud computing.

In the last decade, many companies on a global and regional scale have catalyzed the cloud transformation, with companies such as Google, IBM, Microsoft, UpCloud, Alibaba, and others heavily investing in the research and development of cloud services. As a result, a shift from localized computing (companies having their own servers and data centers) to on-demand computing has taken place due to the availability of robust and scalable cloud services. Now businesses and organizations are able to provision resources on-demand on the cloud to satisfy their data processing needs.

With these developments, we have witnessed **Moore's law** in operation, which states that the number of transistors on a microchip doubles every 2 years – though the cost of computers has halved, this has been true so far. Subsequently, some trends are developing as follows.

The rise of machine learning and deep learning

Over the last decade, we have witnessed the adoption of ML in everyday life applications. Not only for esoteric applications such as **Dota** or **AlphaGo**, but ML has also made its way to pretty standard applications such as machine translation, image processing, and voice recognition.

This adoption is powered by developments in infrastructure, especially in terms of the utilization of computation power. It has unlocked the potential of deep learning and ML.. We can observe deep learning breakthroughs correlated with computation developments in *Figure 1.1* (sourced from OpenAI: `https://openai.com/blog/ai-and-compute`):

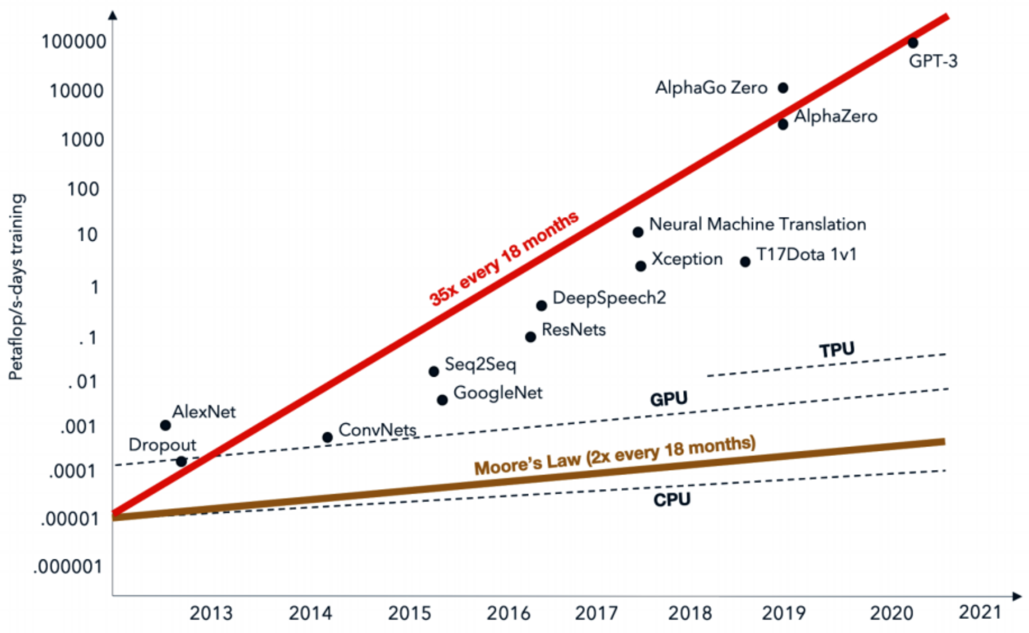

Figure 1.1 – Demand for deep learning over time supported by computation

These breakthroughs in deep learning are enabled by the exponential growth in computing, which increases around 35 times every 18 months. Looking ahead in time, with such demands we may hit roadblocks in terms of scaling up central computing for CPUs, GPUs, or TPUs. This has forced us to look at alternatives such as **distributed learning** where computation for data processing is distributed across multiple computation nodes. We have seen some breakthroughs in distributed learning, such as federated learning and edge computing approaches. Distributed learning has shown promise to serve the growing demands of deep learning.

The end of Moore's law

Prior to 2012, AI results closely tracked Moore's law, with compute doubling every 2 years. Post-2012, compute has been doubling every 3.4 months (sourced from AI Index 2019 – `https://hai.stanford.edu/research/ai-index-2019`). We can observe from *Figure 1.1* that demand for deep learning and **high-performance computing** (**HPC**) has been increasing exponentially with around 35x growth in computing every 18 months whereas Moore's law is seen to be outpaced (2x every 18 months). Moore's law is still applicable to the case of CPUs (single-core performance) but not to new hardware architectures such as GPUs and TPUs. This makes Moore's law obsolete and outpaced in contrast to current demands and trends.

AI-centric applications

Applications are becoming AI-centric – we see that across multiple industries. Virtually every application is starting to use AI, and these applications are running separately on distributed workloads such as **HPC**, **microservices**, and **big data**, as shown in *Figure 1.2*:

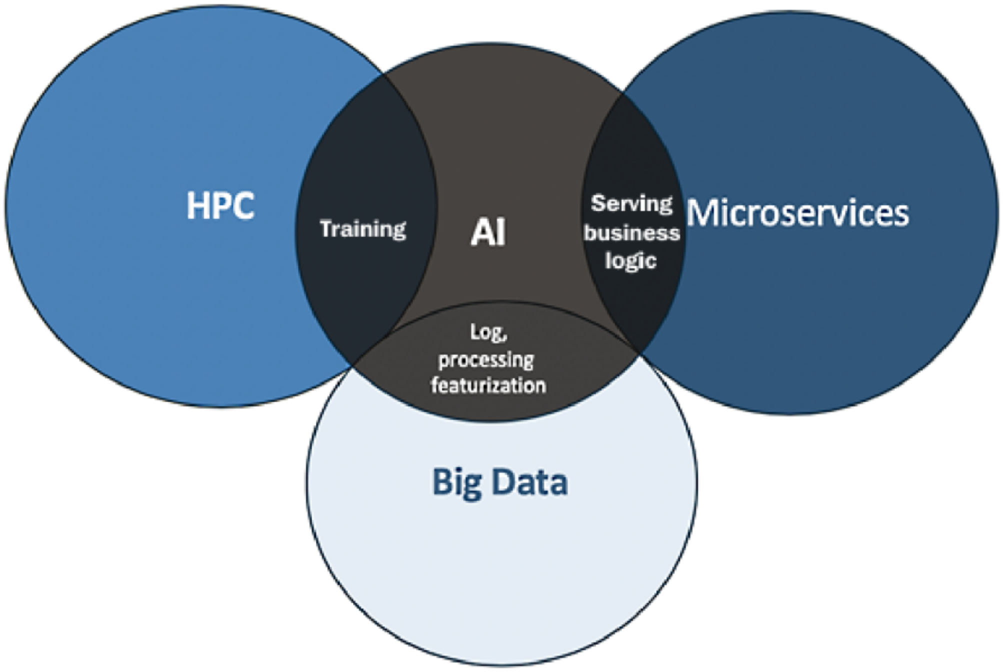

Figure 1.2 – Applications running on distributed workloads

By combining HPC and AI, we can enable the benefits of computation needed to train deep learning and ML models. With the overlapping of big data and AI, we can leverage extracting required data at scale for AI model training, and with the overlap of microservices and AI we can serve the AI models for inference to enhance business operations and impact. This way, distributed applications have become the new norm. Developing AI-centric applications at scale requires a synergy of distributed applications (HPC, microservices, and big data) and for this, a new way of developing software is required.

Software development evolution

Software development has evolved hand in hand with infrastructural developments to facilitate the efficient development of applications using the infrastructure. Traditionally, software development started with the waterfall method of development where development is done linearly by gathering requirements to design and develop. The waterfall model has many limitations, which led to the evolution of software development over the years in the form of Agile methodologies and the DevOps method, as shown in *Figure 1.3*:

Figure 1.3 – Software development evolution

The waterfall method

The **waterfall method** was used to develop software from the onset of the internet age (~1995). It is a non-iterative way of developing software. It is delivered in a unidirectional way. Every stage is pre-organized and executed one after another, starting from requirements gathering to software design, development, and testing. The waterfall method is feasible and suitable when requirements are well-defined, specific, and do not change over time. Hence this is not suitable for dynamic projects where requirements change and evolve as per user demands. In such cases, where there is continuous modification, the waterfall method cannot be used to develop software. These are the major disadvantages of waterfall development methods:

- The entire set of requirements has to be given before starting the development; modifying them during or after the project development is not possible.

- There are fewer chances to create or implement reusable components.
- Testing can only be done after the development is finished. Testing is not intended to be iterable; it is not possible to go back and fix anything once it is done. Moreover, customer acceptance tests often introduced changes, resulting in a delay in delivery and high costs. This way of development and testing can have a negative impact on the project delivery timeline and costs.
- Most of the time, users of the system are provisioned with a system based on the developer's understanding, which is not user-centric and can come short of meeting their needs.

The Agile method

The **Agile method** facilitates an iterative and progressive approach to software development. Unlike the waterfall method, Agile approaches are precise and user-centric. The method is bidirectional and often involves end users or customers in the development and testing process so they have the opportunity to test, give feedback, and suggest improvements throughout the project development process and phases. Agile has several advantages over the waterfall method:

- Requirements are defined before starting the development, but they can be modified at any time.
- It is possible to create or implement reusable components.
- The solution or project can be modular by segregating the project into different modules that are delivered periodically.
- The users or customers can co-create by testing and evaluating developed solution modules periodically to ensure the business needs are satisfied. Such a user-centric process ensures quality outcomes focused on meeting customer and business needs.

The following diagram shows the difference between **Waterfall** and **Agile** methodologies:

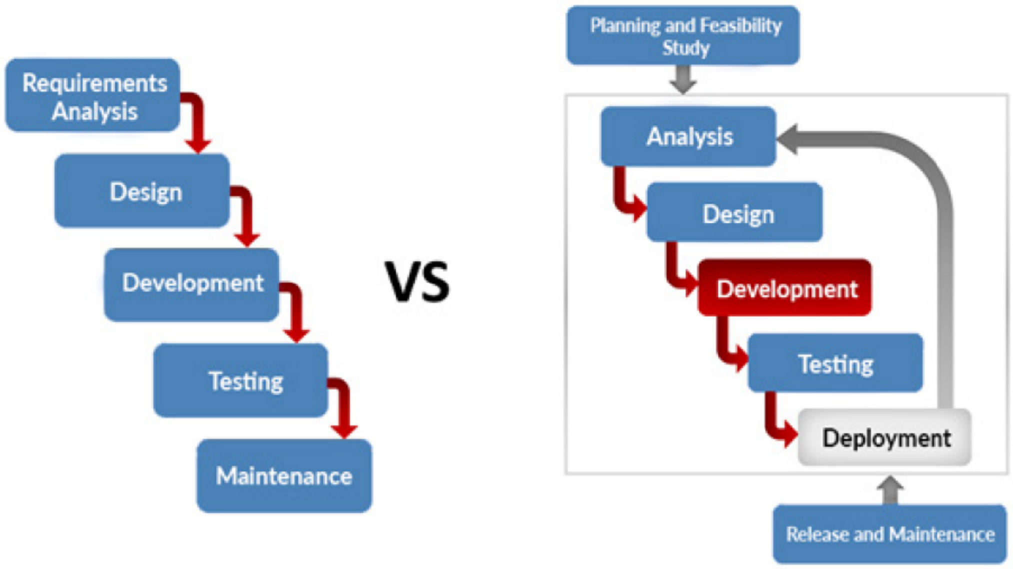

Figure 1.4 – Difference between waterfall and agile methods

The DevOps method

The **DevOps method** extends agile development practices by further streamlining the movement of software change through the *build*, *test*, *deploy*, and *delivery* stages. DevOps empowers cross-functional teams with the autonomy to execute their software applications driven by continuous integration, continuous deployment, and continuous delivery. It encourages collaboration, integration, and automation among software developers and IT operators to improve the efficiency, speed, and quality of delivering customer-centric software. DevOps provides a streamlined software development framework for designing, testing, deploying, and monitoring systems in production. DevOps has made it possible to ship software to production in minutes and to keep it running reliably.

Traditional software development challenges

In the previous section, we observed the shift in traditional software development from the waterfall model to agile and DevOps practices. Agile and DevOps practices have enabled companies to ship software reliably. DevOps has made it possible to ship software to production in minutes and to keep it running reliably. This approach has been so successful that many companies are already adopting it, so why can't we keep doing the same thing for ML applications?

The leading cause is that there's a fundamental difference between ML development and traditional software development: *Machine learning is not just code; it is code plus data*. A ML model is created by applying an algorithm (via code) to fit the data to result in a ML model, as shown in *Figure 1.5*:

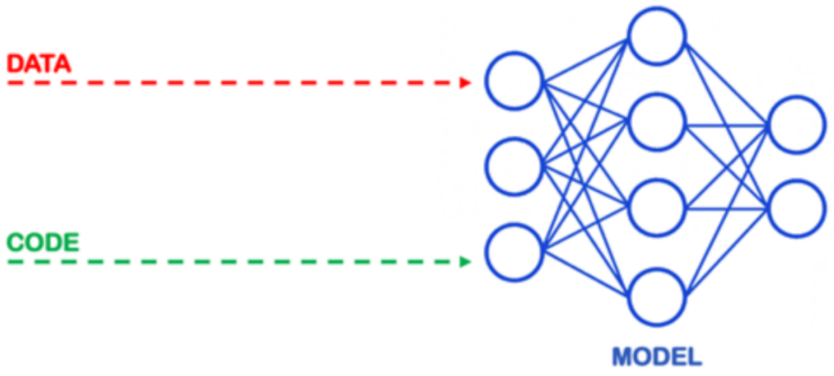

Figure 1.5 – Machine learning = data + code

While code is meticulously crafted in the development environment, data comes from multiple sources for training, testing, and inference. It is robust and changing over time in terms of volume, velocity, veracity, and variety. To keep up with evolving data, code evolves over time. For perspective, their relationship can be observed as if code and data live in separate planes that share the time dimension but are independent in all other aspects. The challenge of an ML development process is to create a bridge between these two planes in a controlled way:

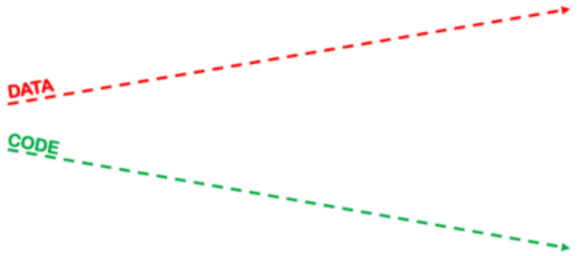

Figure 1.6 – Data and code progression over time

Data and code, with the progression of time, end up going in two directions with one objective of building and maintaining a robust and scalable ML system. This disconnect causes several challenges that need to be solved by anyone trying to put a ML model in production. It comes with challenges such as slow, brittle, fragmented, and inconsistent deployment, and a lack of reproducibility and traceability.

To overcome these challenges, MLOps offers a systematic approach by bridging data and code together over the progression of time. This is the solution to challenges posed by traditional software development methods with regard to ML applications. Using the MLOps method, data and code progress over time in one direction with one objective of building and maintaining a robust and scalable ML system:

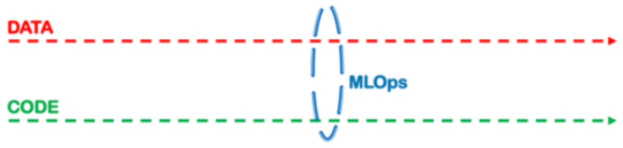

Figure 1.7 – MLOps – data and code progressing together

MLOps facilitates ML model development, deployment, and monitoring in a streamlined and systematic approach. It empowers data science and IT teams to collaborate, validate, and govern their operations. All the operations executed by the teams are recorded or audited, end-to-end traceable, and repeatable. In the coming sections, we will learn how MLOps enables data science and IT teams to build and maintain robust and scalable ML systems.

Trends of ML adoption in software development

Before we delve into the workings of the MLOps method and workflow, it is beneficial to understand the big picture and trends as to where and how MLOps is disrupting the world. As many applications are becoming AI-centric, software development is evolving to facilitate ML. ML will increasingly become part of software development, mainly due to the following reasons:

- **Investments**: In 2019, investments in global private AI clocked over $70 billion, with start-up investments related to AI over $37 billion, M&A $34 billion, IPOs $5 billion, and minority stake valued at around $2 billion. The forecast for AI globally shows fast growth in market value as AI reached $9.5 billion in 2018 and is anticipated to reach a market value of $118 billion by 2025. It has been assessed that growth in economic activity resulting from AI until 2030 will be of high value and significance. Currently, the US attracts ~50% of global VC funding, China ~39%, and 11% goes to Europe.
- **Big data**: Data is exponentially growing in volume, velocity, veracity, and variety. For instance, observations suggest data growing in volume at 61% per annum in Europe, and it is anticipated that four times more data will be created by 2025 than exists today. Data is a requisite raw material for developing AI.
- **Infrastructural developments and adoption**: Moore's law has been closely tracked and observed to have been realized prior to 2012. Post-2012, compute has been doubling every 3.4 months.
- **Increasing research and development**: AI research has been prospering in quality and quantity. A prominent growth of 300% is observed in the volume of peer-reviewed AI papers from 1998 to 2018, summing up to 9% of published conference papers and 3% of peer-reviewed journal publications.
- **Industry**: Based on a surveyed report, 47% of large companies have reported having adopted AI in at least one function or business unit. In 2019, it went up to 58% and is expected to increase.

Information

These points have been sourced from policy and investment recommendations for trustworthy AI – European commission (`https://ec.europa.eu/digital-single-market/en/news/policy-and-investment-recommendations-trustworthy-artificial-intelligence`) and AI Index 2019 (`https://hai.stanford.edu/research/ai-index-2019`).

All these developments indicate a strong push toward the industrialization of AI, and this is possible by bridging industry and research. MLOps will play a key role in the industrialization of AI. If you invest in learning this method, it will give you a headstart in your company or team and you could be a catalyst for operationalizing ML and industrializing AI.

So far, we have learned about some challenges and developments in IT, software development, and AI. Next, we will delve into understanding MLOps conceptually and learn in detail about a generic MLOps workflow that can be used commonly for any use case. These fundamentals will help you get a firm grasp of MLOps.

Understanding MLOps

Software development is interdisciplinary and is evolving to facilitate ML. MLOps is an emerging method to fuse ML with software development by integrating multiple domains as MLOps combines ML, DevOps, and data engineering, which aims to build, deploy, and maintain ML systems in production reliably and efficiently. Thus, MLOps can be expounded by this intersection.

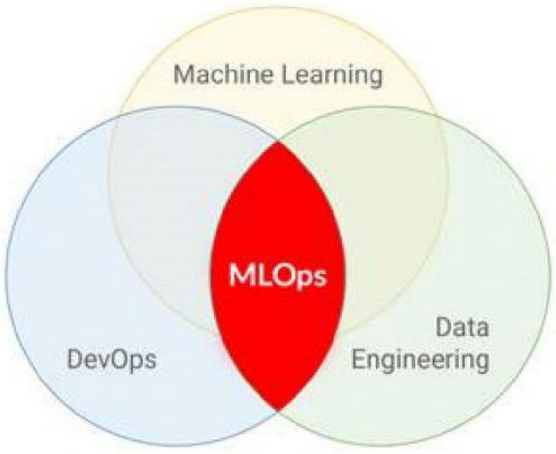

Figure 1.8 – MLOps intersection

To make this intersection (MLOps) operational, I have designed a modular framework by following the systematic *design science method proposed by Wieringa* (`https://doi.org/10.1007/978-3-662-43839-8`) to develop a workflow to bring these three together (Data Engineering, Machine Learning, and *DevOps*). Design science goes with the application of design to problems and context. Design science is the design and investigation of artifacts in a context. The artifact in this case is the MLOps workflow, which is designed iteratively by interacting with problem contexts (industry use cases for the application of AI):

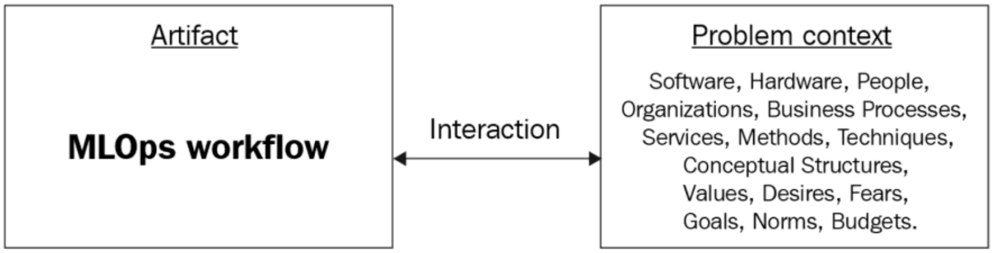
Artifact
MLOps workflow
Interaction
Problem context
Software, Hardware, People, Organizations, Business Processes, Services, Methods, Techniques, Conceptual Structures, Values, Desires, Fears, Goals, Norms, Budgets.

Figure 1.9 – Design science workflow

In a structured and iterative approach, the implementation of two cycles (the design cycle and the empirical cycle) was done for qualitative and quantitative analysis for MLOps workflow design through iterations. As a result of these cycles, an MLOps workflow is developed and validated by applying it to multiple problem contexts, that is, tens of ML use cases (for example, anomaly detection, real-time trading, predictive maintenance, recommender systems, virtual assistants, and so on) across multiple industries (for example, finance, manufacturing, healthcare, retail, the automotive industry, energy, and so on). I have applied and validated this MLOps workflow successfully in various projects across multiple industries to operationalize ML. In the next section, we will go through the concepts of the MLOps workflow designed as a result of the design science process.

Concepts and workflow of MLOps

In this section, we will learn about a generic MLOps workflow; it is the result of many design cycle iterations as discussed in the previous section. It brings together data engineering, ML, and DevOps in a streamlined fashion. *Figure 1.10* is a generic MLOps workflow; it is modular and flexible and can be used to build proofs of concept or to operationalize ML solutions in any business or industry:

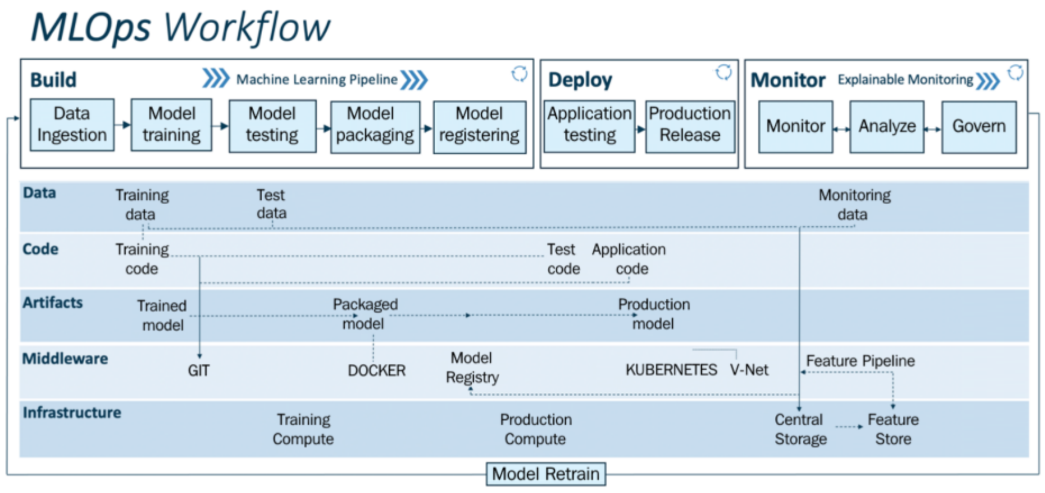

Figure 1.10 – MLOps workflow

This workflow is segmented into two modules:

- **MLOps pipeline** (build, deploy, and monitor) – the upper layer
- **Drivers**: Data, code, artifacts, middleware, and infrastructure – mid and lower layers

The upper layer is the MLOps pipeline (build, deploy, and monitor), which is enabled by drivers such as data, code, artifacts, middleware, and infrastructure. The MLOps pipeline is powered by an array of services, drivers, middleware, and infrastructure, and it crafts ML-driven solutions. By using this pipeline, a business or individual(s) can do quick prototyping, testing, and validating and deploy the model(s) to production at scale frugally and efficiently.

To understand the workings and implementation of the MLOps workflow, we will look at the implementation of each layer and step using a figurative business use case.

Discussing a use case

In this use case, we are to operationalize (prototyping and deploying for production) an image classification service to classify cats and dogs in a pet park in Barcelona, Spain. The service will identify cats and dogs in real time from the inference data coming from a CCTV camera installed in the pet park.

The pet park provide you access to the data and infrastructure needed to operationalize the service:

- **Data**: The pet park has given you access to their data lake containing 100,000 labeled images of cats and dogs, which we will use for training the model.
- **Infrastructure**: Public cloud (IaaS).

This use case resembles a real-life use case for operationalizing ML and is used to explain the workings and implementation of the MLOps workflow. Remember to look for an explanation for the implementation of this use case at every segment and step of the MLOps workflow. Now, let's look at the workings of every layer and step in detail.

The MLOps pipeline

The MLOps pipeline is the upper layer, which performs operations such as build, deploy, and monitor, which work modularly in sync with each other. Let's look into each module's functionality.

Build

The build module has the core ML pipeline, and this is purely for training, packaging, and versioning the ML models. It is powered by the required compute (for example, the CPU or GPU on the cloud or distributed computing) resources to run the ML training and pipeline:

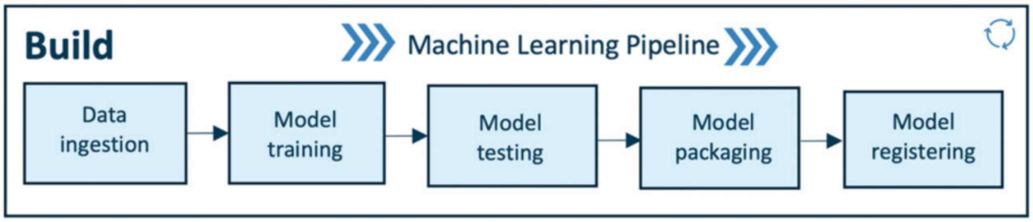

Figure 1.11 – MLOps – build pipeline

The pipeline works from left to right. Let's look at the functionality of each step in detail:

- **Data ingestion**: This step is a trigger step for the ML pipeline. It deals with the volume, velocity, veracity, and variety of data by extracting data from various data sources (for example, databases, data warehouses, or data lakes) and ingesting the required data for the model training step. Robust data pipelines connected to multiple data sources enable it to perform **extract, transform, and load** (**ETL**) operations to provide the necessary data for ML training purposes. In this step, we can split and version data for model training in the required format (for example, the training or test set). As a result of this step, any experiment (that is, model training) can be audited and is back-traceable.

  For a better understanding of the data ingestion step, here is the previously described use case implementation:

  Use case implementation

  As you have access to the pet park's data lake, you can now procure data to get started. Using data pipelines (part of the data ingestion step), you do the following:

  1. Extract, transform, and load 100,000 images of cats and dogs.
  2. Split and version this data into a train and test split (with an 80% and 20% split).

  Versioning this data will enable end-to-end traceability for trained models.

  Congrats – now you are ready to start training and testing the ML model using this data.

- **Model training**: After procuring the required data for ML model training in the previous step, this step will enable model training; it has modular scripts or code that perform all the traditional steps in ML, such as data preprocessing, feature engineering, and feature scaling before training or retraining any model. Following this, the ML model is trained while performing hyperparameter tuning to fit the model to the dataset (training set). This step can be done manually, but efficient and automatic solutions such as **Grid Search** or **Random Search** exist. As a result, all important steps of ML model training are executed with a ML model as the output of this step.

  Use case implementation

In this step, we implement all the important steps to train the image classification model. The goal is to train a ML model to classify cats and dogs. For this case, we train a **convolutional neural network** (**CNN** – `https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb`) for the image classification service. The following steps are implemented: data preprocessing, feature engineering, and feature scaling before training, followed by training the model with hyperparameter tuning. As a result, we have a CNN model to classify cats and dogs with 97% accuracy.

- **Model testing**: In this step, we evaluate the trained model performance on a separated set of data points named test data (which was split and versioned in the data ingestion step). The inference of the trained model is evaluated according to selected metrics as per the use case. The output of this step is a report on the trained model's performance.

  Use case implementation

  We test the trained model on test data (we split data earlier in the *Data ingestion* step) to evaluate the trained model's performance. In this case, we look for precision and the recall score to validate the model's performance in classifying cats and dogs to assess false positives and true positives to get a realistic understanding of the model's performance. If and when we are satisfied with the results, we can proceed to the next step, or else reiterate the previous steps to get a decent performing model for the pet park image classification service.

- **Model packaging**: After the trained model has been tested in the previous step, the model can be serialized into a file or containerized (using Docker) to be exported to the production environment.

  Use case implementation

  The model we trained and tested in the previous steps is serialized to an ONNX file and is ready to be deployed in the production environment.

- **Model registering**: In this step, the model that was serialized or containerized in the previous step is registered and stored in the model registry. A registered model is a logical collection or package of one or more files that assemble, represent, and execute your ML model. For example, multiple files can be registered as one model. For instance, a classification model can be comprised of a vectorizer, model weights, and serialized model files. All these files can be registered as one single model. After registering, the model (all files or a single file) can be downloaded and deployed as needed.

  Use case implementation

  The serialized model in the previous step is registered on the model registry and is available for quick deployment into the pet park production environment.

  By implementing the preceding steps, we successfully execute the ML pipeline designed for our use case. As a result, we have trained models on the model registry ready to be deployed in the production setup. Next, we will look into the workings of the deployment pipeline.

Deploy

The deploy module enables operationalizing the ML models we developed in the previous module (build). In this module, we test our model performance and behavior in a production or production-like (test) environment to ensure the robustness and scalability of the ML model for production use. *Figure 1.12* depicts the deploy pipeline, which has two components – production testing and production release – and the deployment pipeline is enabled by streamlined CI/CD pipelines connecting the development to production environments:

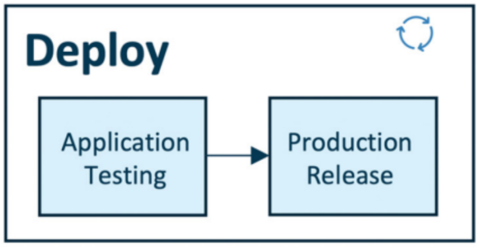

Figure 1.12 – MLOps – deploy pipeline

It works from left to right. Let's look at the functionality of each step in detail:

- **Application testing**: Before deploying an ML model to production, it is vital to test its robustness and performance via testing. Hence we have the "application testing" phase where we rigorously test all the trained models for robustness and performance in a production-like environment called a test environment. In the application testing phase, we deploy the models in the test environment (pre-production), which replicates the production environment.

  The ML model for testing is deployed as an API or streaming service in the test environment to deployment targets such as Kubernetes clusters, container instances, or scalable virtual machines or edge devices as per the need and use case. After the model is deployed for testing, we perform predictions using test data (which is not used for training the model; test data is sample data from a production environment) for the deployed model, during which model inference in batch or periodically is done to test the model deployed in the test environment for robustness and performance.

  The performance results are automatically or manually reviewed by a quality assurance expert. When the ML model's performance meets the standards, then it is approved to be deployed in the production environment where the model will be used to infer in batches or real time to make business decisions.

  Use case implementation

  We deploy the model as an API service on an on-premises computer in the pet park, which is set up for testing purposes. This computer is connected to a CCTV camera in the park to fetch real-time inference data to predict cats or dogs in the video frames. The model deployment is enabled by the CI/CD pipeline. In this step, we test the robustness of the model in a production-like environment, that is, whether the model is performing inference consistently, and an accuracy, fairness, and error analysis. At the end of this step, a quality assurance expert certifies the model if it meets the standards.

- **Production release**: Previously tested and approved models are deployed in the production environment for model inference to generate business or operational value. This production release is deployed to the production environment enabled by CI/CD pipelines.

  Use case implementation

  We deploy a previously tested and approved model (by a quality assurance expert) as an API service on a computer connected to CCTV in the pet park (production setup). This deployed model performs ML inference on the incoming video data from the CCTV camera in the pet park to classify cats or dogs in real time.

Monitor

The monitor module works in sync with the deploy module. Using explainable monitoring (discussed later in detail, in *Chapter 11*, *Key Principles for Monitoring Your ML System*), we can monitor, analyze, and govern the deployed ML application (ML model and application). Firstly, we can monitor the performance of the ML model (using pre-defined metrics) and the deployed application (using telemetry data). Secondly, model performance can be analyzed using a pre-defined explainability framework, and lastly, the ML application can be governed using alerts and actions based on the model's quality assurance and control. This ensures a robust monitoring mechanism for the production system:

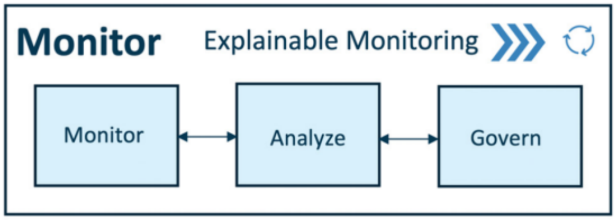

Figure 1.13 – MLOps – monitor pipeline

Let's see each of the abilities of the monitor module in detail:

- **Monitor**: The monitoring module captures critical information to monitor data integrity, model drift, and application performance. Application performance can be monitored using telemetry data. It depicts the device performance of a production system over a period of time. With telemetry data such as accelerometer, gyroscope, humidity, magnetometer, pressure, and temperature we can keep a check on the production system's performance, health, and longevity.

  Use case implementation

In real time, we will monitor three things – data integrity, model drift, and application performance – for the deployed API service on the park's computer. Metrics such as accuracy, F1 score, precision, and recall are tracked to data integrity and model drift. We monitor application performance by tracking the telemetry data of the production system (the on-premises computer in the park) running the deployed ML model to ensure the proper functioning of the production system. Telemetry data is monitored to foresee any anomalies or potential failures and fix them in advance. Telemetry data is logged and can be used to assess production system performance over time to check its health and longevity.

- **Analyze**: It is critical to analyze the model performance of ML models deployed in production systems to ensure optimal performance and governance in correlation to business decisions or impact. We use model explainability techniques to measure the model performance in real time. Using this, we evaluate important aspects such as model fairness, trust, bias, transparency, and error analysis with the intention of improving the model in correlation to business.

  Over time, the statistical properties of the target variable we are trying to predict can change in unforeseen ways. This change is called "model drift," for example, in a case where we have deployed a recommender system model to suggest suitable items for users. User behavior may change due to unforeseeable trends that could not be observed in historical data that was used for training the model. It is essential to consider such unforeseen factors to ensure deployed models provide the best and most relevant business value. When model drift is observed, then any of these actions should be performed:

  a) The product owner or the quality assurance expert needs to be alerted.

  b) The model needs to be switched or updated.

  c) Re-training the pipeline should be triggered to re-train and update the model as per the latest data or needs.

  Use case implementation

  We monitor the deployed model's performance in the production system (a computer connected to the CCTV in the pet park). We will analyze the accuracy, precision, and recall scores for the model periodically (once a day) to ensure the model's performance does not deteriorate below the threshold. When the model performance deteriorates below the threshold, we initiate system governing mechanisms (for example, a trigger to retrain the model).

- **Govern**: Monitoring and analyzing is done to govern the deployed application to drive optimal performance for the business (or the purpose of the ML system). After monitoring and analyzing the production data, we can generate certain alerts and actions to govern the system. For example, the product owner or the quality assurance expert gets alerted when model performance deteriorates (for example, low accuracy, high bias, and so on) below a pre-defined threshold. The product owner initiates a trigger to retrain and deploy an alternative model. Lastly, an important aspect of governance is "compliance" with the local and global laws and rules. For compliance, model explainability and transparency are vital. For this, model auditing and reporting are done to provide end-to-end traceability and explainability for production models.

  Use case implementation

  We monitor and analyze the deployed model's performance in the production system (a computer connected to the CCTV in the pet park). Based on the analysis of accuracy, precision, and recall scores for the deployed model, periodically (once a day), alerts are generated when the model's performance deteriorates below the pre-defined threshold. The product owner of the park generates actions, and these actions are based on the alerts. For example, an alert is generated notifying the product owner that the production model is 30% biased to detect dogs more than cats. The product owner then triggers the model re-training pipeline to update the model using the latest data to reduce the bias, resulting in a fair and robust model in production. This way, the ML system at the pet park in Barcelona is well-governed to serve the business needs.

This brings us to the end of the MLOps pipeline. All models trained, deployed, and monitored using the MLOps method are end-to-end traceable and their lineage is logged in order to trace the origins of the model, which includes the source code the model used to train, the data used to train and test the model, and parameters used to converge the model. Full lineage is useful to audit operations or to replicate the model, or when a blocker is hit, the logged ML model lineage is useful to backtrack the origins of the model or to observe and debug the cause of the blocker. As ML models generate data in production during inference, this data can be tied to the model training and deployment lineage to ensure the end-to-end lineage, and this is important for certain compliance requirements. Next, we will look into key drivers enabling the MLOps pipeline.

Drivers

These are the key drivers for the MLOps pipeline: data, code, artifacts, middleware, and infrastructure. Let's look into each of the drivers to get an overview of how they enable the MLOps pipeline:

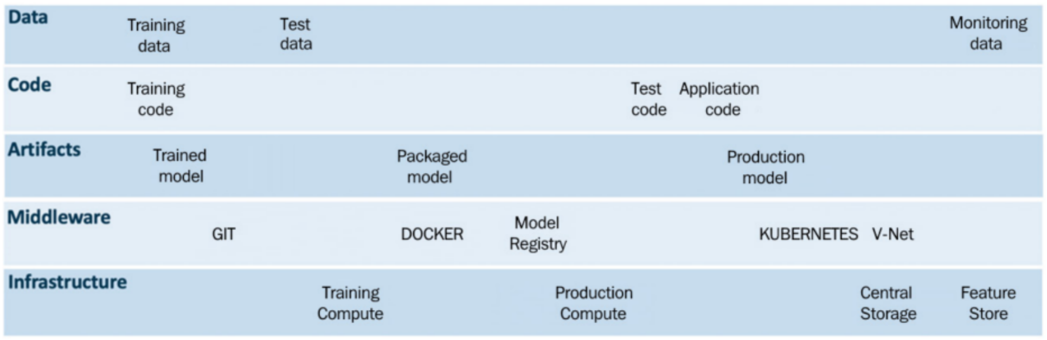

Figure 1.14 – MLOps drivers

Each of the key drivers for the MLOps pipeline are defined as follows:

- **Data**: Data can be in multiple forms, such as text, audio, video, and images. In traditional software applications, data quite often tends to be structured, whereas, for ML applications, it can be structured or unstructured. To manage data in ML applications, data is handled in these steps: data acquisition, data annotation, data cataloging, data preparation, data quality checking, data sampling, and data augmentation. Each step involves its own life cycle. This makes a whole new set of processes and tools necessary for ML applications. For efficient functioning of the ML pipeline, data is segmented and versioned into training data, testing data, and monitoring data (collected in production, for example, model inputs, outputs, and telemetry data). These data operations are part of the MLOps pipeline.
- **Code**: There are three essential modules of code that drive the MLOps pipeline: training code, testing code, and application code. These scripts or code are executed using the CI/CD and data pipelines to ensure the robust working of the MLOps pipeline. The source code management system (for example, using Git or Mercurial) will enable orchestration and play a vital role in managing and integrating seamlessly with CI, CD, and data pipelines. All of the code is staged and versioned in the source code management setup (for example, Git).
- **Artifacts**: The MLOps pipeline generates artifacts such as data, serialized models, code snippets, system logs, ML model training, and testing metrics information. All these artifacts are useful for the successful working of the MLOps pipeline, ensuring its traceability and sustainability. These artifacts are managed using middleware services such as the model registry, workspaces, logging services, source code management services, databases, and so on.

- **Middleware**: Middleware is computer software that offers services to software applications that are more than those available from the operating systems. Middleware services ensure multiple applications to automate and orchestrate processes for the MLOps pipeline. We can use a diverse set of middleware software and services depending on the use cases, for example, Git for source code management, VNets to enable the required network configurations, Docker for containerizing our models, and Kubernetes for container orchestration to automate application deployment, scaling, and management.
- **Infrastructure**: To ensure the successful working of the MLOps pipeline, we need essential compute and storage resources to train Test and deploy the ML models. Compute resources enable us to train, deploy and monitor our ML models. Two types of storages resources can facilitate ML operations, central storage and feature stores. A central storage stores the logs, artifacts, training, testing and monitoring data. A feature store is optional and complementary to central storage. It extracts, transforms and stores needed features for ML model training and inference using a feature pipeline. When it comes to the infrastructure, there are various options such as on-premises resources or **infrastructure as a service (IaaS)**, which is cloud services. These days, there are many cloud players providing IaaS, such as Amazon, Microsoft, Google, Alibaba, and so on. Having the right infrastructure for your use case will enable robust, efficient, and frugal operations for your team and company.

  A fully automated workflow is achievable with smart optimization and synergy of all these drivers with the MLOps pipeline. Some direct advantages of implementing an automated MLOps workflow is a spike in IT teams' efficiency (by reducing the time spent by data scientists and developers on mundane and repeatable tasks) and the optimization of resources, resulting in cost reductions, and both of these are great for any business.

Summary

In this chapter, we have learned about the evolution of software development and infrastructure to facilitate ML. We delved into the concepts of MLOps, followed by getting acquainted with a generic MLOps workflow that can be implemented in a wide range of ML solutions across multiple industries.

In the next chapter, you will learn how to characterize any ML problem into an MLOps-driven solution and start developing it using an MLOps workflow.

2 Characterizing Your Machine Learning Problem

In this chapter, you will get a fundamental understanding of the various types of **Machine Learning** (**ML**) solutions that can be built for production, and will learn to categorize the relevant operations in line with the business and technological needs of your organization. You will learn how to curate an implementation roadmap for operationalizing ML solutions, followed by procuring the necessary tools and infrastructure for any given problem. By the end of this chapter, you will have a solid understanding of how to architect robust and scalable ML solutions and procure the required data and tools for implementing these solutions.

ML Operations (**MLOps**) aims to bridge academia and industry using state-of-the-art engineering principles, and we will explore different elements from both industry and academia to get a holistic understanding and awareness of the possibilities. Before beginning to craft your MLOps solution, it is important to understand the various possibilities, setups, problems, solutions, and methodologies on offer for solving business-oriented problems. To achieve this understanding, we're going to cover the following main topics in this chapter:

- The ML solution development process
- Types of ML models

- Characterizing your MLOps
- An implementation roadmap for your solution
- Procuring the necessary data, tools, and infrastructure
- Introduction to a real-life business problem

Without further ado, let's jump in and explore the possibilities ML can enable by taking an in-depth look into the ML solution development process and examining different types of ML models to solve business problems.

The ML solution development process

ML offers many possibilities to augment and automate business. To get the best from ML, teams and people engaged in ML-driven business transformation need to understand both ML and the business itself. Efficient business transformation begins with having a rough understanding of the business, including aspects such as value-chain analysis, use-case identification, data mapping, and business simulations to validate the business transformation. *Figure 2.1* presents a process to develop ML solutions to augment or automate business operations:

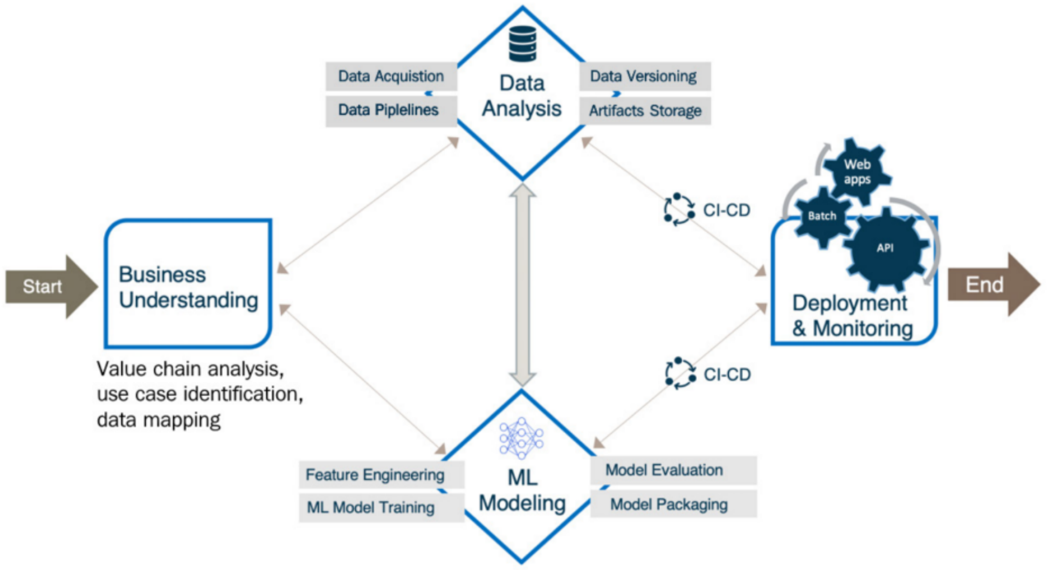

Figure 2.1 – ML solution development process

Business understanding is the genesis of developing an ML solution. After having a decent business understanding, we proceed to data analysis, where the right data is acquired, versioned, and stored. Data is consumed for ML modeling using data pipelines where feature engineering is done to get the right features to train the model. We evaluate the trained models and package them for deployment. Deployment and monitoring are done using a pipeline taking advantage of **Continuous Integration/Continuous Deployment** (**CI/CD**) features that enable real-time and continuous deployment to serve trained ML models to the users. This process ensures robust and scalable ML solutions.

Types of ML models

As there is a selection of ML and deep learning models that address the same business problem, it is essential to understand the landscape of ML models in order to make an efficient algorithm selection. There are around 15 types of ML techniques, these being categorized into 4 categories, namely **learning models**, **hybrid models**, **statistical models**, and **Human-In-The-Loop** (**HITL**) models, as shown in the following matrix (where each grid square reflects one of these categories) in *Figure 2.2*. It is worth noting that there are other possible ways of categorizing ML models and none of them are fully complete, and as such, these categorizations will serve appropriately for some scenarios and not for others. Here is our recommended categorization with which to look at ML models:

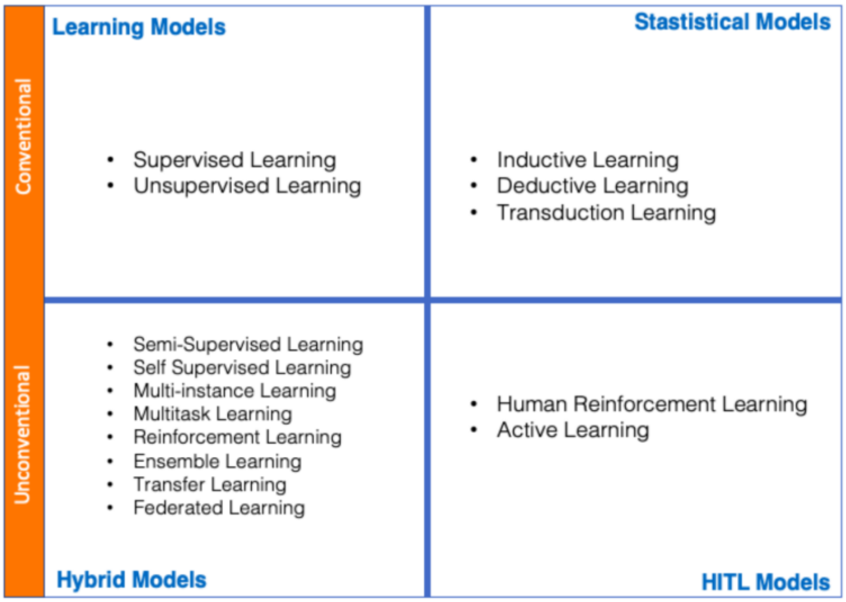

Figure 2.2 – Types of ML models

Learning models

First, we'll take a look at two types of standard learning models, **supervised learning** and **unsupervised learning**:

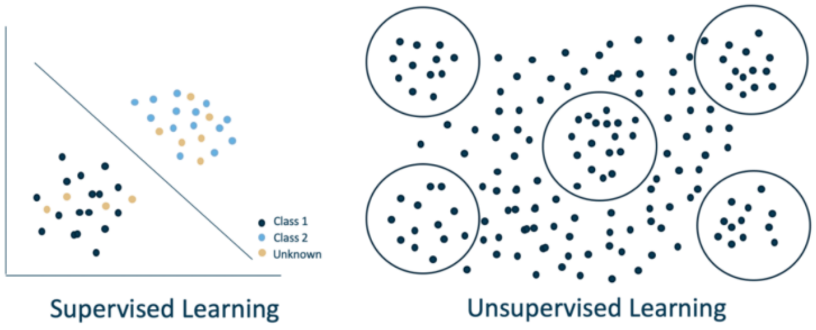

Figure 2.3 – Supervised versus unsupervised learning

Supervised learning

Supervised learning models or algorithms are trained based on labeled data. In the training data, the result of the input is marked or known. Hence a model is trained to predict the outcome when given an input based on the labeled data it learns from, and you tell the system which output corresponds with a given input in the system.

Supervised learning models are very effective on narrow AI cases and well-defined tasks but can only be harnessed where there is sufficient and comprehensive labeled data. We can see in *Figure 2.3*, in the case of supervised learning, that the model has learned to predict and classify an input.

Consider the example of an image classification model used to classify images of cats and dogs. A supervised learning model is trained on labeled data consisting of thousands of correctly labeled images of cats and dogs. The trained model then learns to classify a given input image as containing a dog or a cat.

Unsupervised learning

Unsupervised learning has nothing to do with a machine running around and doing things without human supervision. Unsupervised learning models or algorithms learn from unlabeled data. Unsupervised learning can be used to mine insights and identify patterns from unlabeled data. Unsupervised algorithms are widely used for clustering or anomaly detection without relying on any labels. These algorithms can be pattern-finding algorithms; when data is fed to such an algorithm, it will identify patterns and turn those into a recipe for taking a new data input without a label and applying the correct label to it.

Unsupervised learning is used mainly for analytics, though you could also use it for automation and ML. It is recommended not to use these algorithms in production due to their dynamic nature that changes outputs on every training cycle. However, they can be useful to automate certain processes such as segmenting incoming data or identifying anomalies in real time.

Let's discuss an example of clustering news articles into relevant groups. Let's assume you have thousands of news articles without any labels and you would like to identify the types or categories of articles. To perform unsupervised learning on these articles, we can input a bunch of articles into the algorithm and converge it to put similar things together (that is, clustering) in four groups. Then, we look at the clusters and discover that similar articles have been grouped together in categories such as politics, sports, science, and health. This is a way of mining patterns in the data.

Hybrid models

There have been rapid developments in ML by combining conventional methods to develop hybrid models to solve diverse business and research problems. Let's look into some hybrid models and how they work. *Figure 2.4* shows various hybrid models:

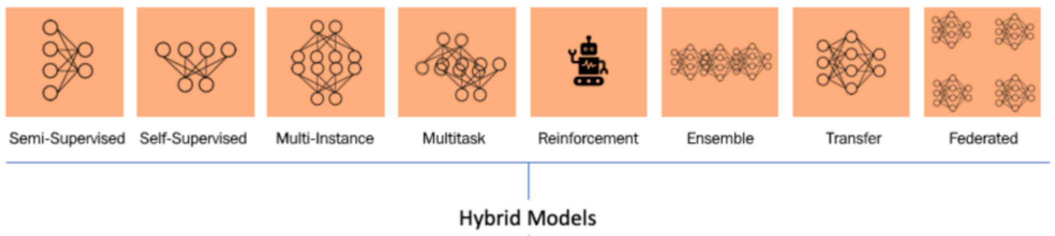

Figure 2.4 – Types of hybrid models

Semi-supervised learning

Semi-supervised learning is a hybrid of supervised learning, used in cases where only a few samples are labeled and a large number of samples are not labeled. Semi-supervised learning enables efficient use of the data available (though not all of it is labeled), including the unlabeled data. For example, a text document classifier is a typical example of a semi-supervised learning program. It will be very difficult to locate a large number of labeled text documents in this case, so semi-supervised learning is ideal. This is due to the fact that making someone read through entire text documents just to assign a basic classification is inefficient. As a result, semi-supervised learning enables the algorithm to learn from a limited number of labeled text documents while classifying the large number of unlabeled text documents present in the training data.

Self-supervised learning

Self-supervised learning problems are unsupervised learning problems where data is not labeled; these problems are translated into supervised learning problems in order to apply algorithms for supervised learning to solve them sustainably. Usually, self-supervised algorithms are used to solve an alternate task in which they supervise themselves to solve the problem or generate an output. One example of self-supervised learning is **Generative Adversarial Networks** (**GANs**); these are commonly used to generate synthetic data by training on labeled and/or unlabeled data. With proper training, GAN models can generate a relevant output in a self-supervised manner. For example, a GAN could generate a human face based on a text description input, such as *gender: male, age: 30, color: brown*, and so on.

Multi-instance learning

Multi-instance learning is a supervised learning problem in which data is not labeled by individual data samples, but cumulatively in categories or classes. Compared to typical supervised learning, where labeling is done for each data sample, such as news articles labeled in categories such as politics, science, and sports, with multi-instance learning, labeling is done categorically. In such scenarios, individual samples are collectively labeled in multiple classes, and by using supervised learning algorithms, we can make predictions.

Multitask learning

Multitask learning is an incarnation of supervised learning that involves training a model on one dataset and using that model to solve multiple tasks or problems. For example, for natural language processing, we use word embeddings or **Bidirectional Encoder Representations from Transformers** (**BERT**) embeddings models, which are trained on one large corpus of data. (BERT is a pre-trained model, trained on a large text corpus. The model has a deep understanding of how a given human language works.) And these models can be used to solve many supervised learning tasks such as text classification, keyword extraction, sentiment analysis, and more.

Reinforcement learning

Reinforcement learning is a type of learning in which an agent, such as a robot system, learns to operate in a defined environment to perform sequential decision-making tasks or achieve a pre-defined goal. Simultaneously, the agent learns based on continuously evaluated feedback and rewards from the environment. Both feedback and rewards are used to shape the learning of the agent, as shown in *Figure 2.5*. An example is Google's AlphaGo, which recently outperformed the world's leading Go player. After 40 days of self-training using feedback and rewards, AlphaGo was able to beat the world's best human Go player:

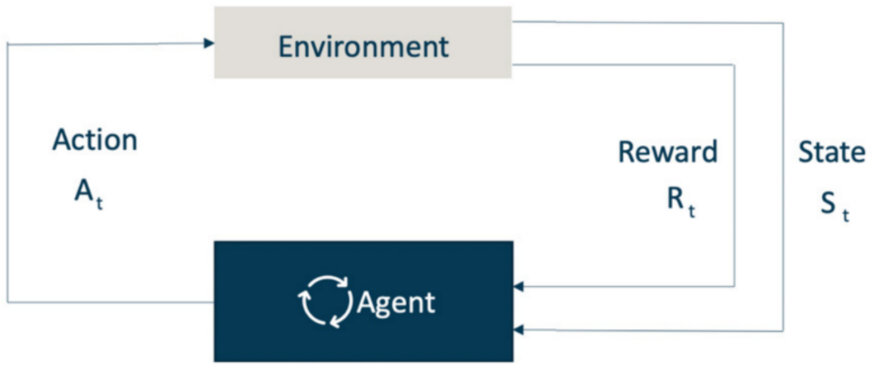

Figure 2.5 – Reinforcement learning

Ensemble learning

Ensemble learning is a hybrid model that involves two or more models trained on the same data. Predictions are made using each model individually and a collective prediction is made as a result of combining all outputs and averaging them to determine the final outcome or prediction. An example of this is the random forest algorithm, which is an ensemble learning method for classification or regression tasks. It operates by composing several decision trees while training, and creates a prediction as output by averaging the predictions of all the decision trees.

Transfer learning

We humans have an innate ability to transfer knowledge to and from one another. This same principle is translated to ML, where a model is trained to perform a task and it is transferred to another model as a starting point for training or fine-tuning for performing another task. This type of learning is popular in deep learning, where pre-trained models are used to solve computer vision or natural language processing problems by fine-tuning or training using a pre-trained model. Learning from pre-trained models gives a huge jumpstart as models don't need to be trained from scratch, saving large amounts of training data. For example, we can train a sentiment classifier model using training data containing only a few labeled data samples. This is possible with transfer learning using a pre-trained BERT model (which is trained on a large corpus of labeled data). This enables the transfer of learning from one model to another.

Federated learning

Federated learning is a way of performing ML in a collaborative fashion (synergy between cloud and edge). The training process is distributed across multiple devices, storing only a local sample of the data. Data is neither exchanged nor transferred between devices or the cloud to maintain data privacy and security. Instead of sharing data, locally trained models are shared to learn from each other to train global models. Let's discuss an example of federated learning in hospitals (as shown in *Figure 2.6*) where patient data is confidential and cannot be shared with third parties. In this case, ML training is done locally in the hospitals (at the edge) and global models are trained centrally (on the cloud) without sharing the data. Models trained locally are fine-tuned to produce global models. Instead of ingesting data in the central ML pipeline, locally trained models are ingested. Global models learn by tuning their parameters from local models to converge on optimal performance, concatenating the learning of local models:

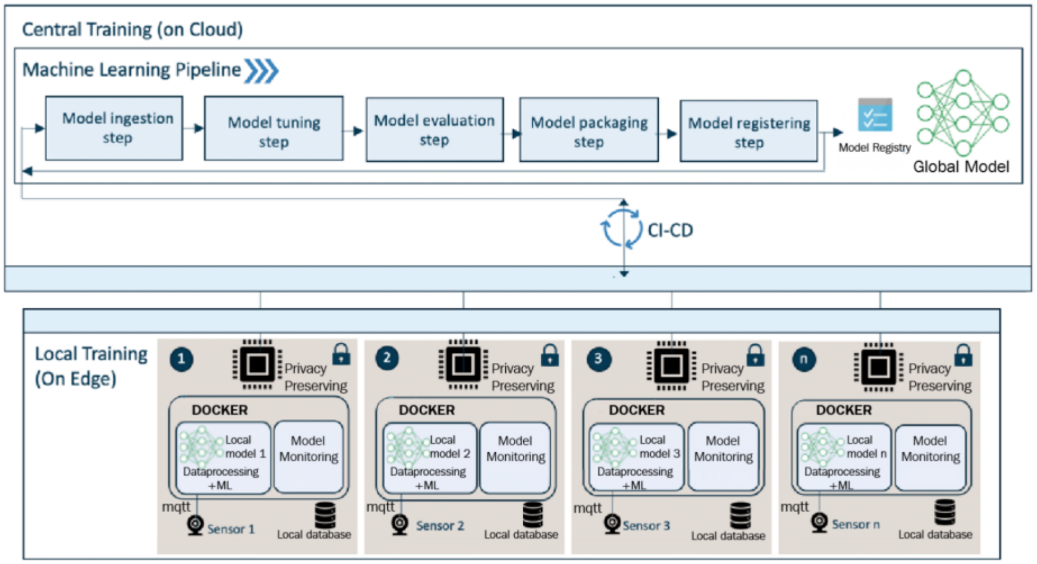

Figure 2.6 – Federated learning architecture

Statistical models

In some cases, statistical models are efficient at making decisions. It is vital to know where statistical models can be used to get the best value or decisions. There are three types of statistical models: inductive learning, deductive learning, and transductive learning. *Figure 2.7* shows the relationship between these types of statistical models:

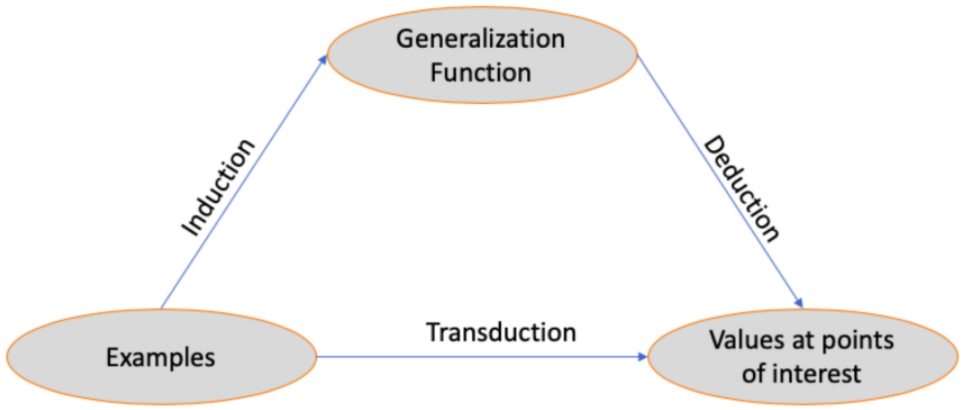

Figure 2.7 – Relationship between the three types of statistical models

Inductive learning is a statistical method that generalizes from specific examples in the training data, using this evidence to determine the most likely outcome. It involves a process of learning by example, where a system tries to generalize a general function or rule from a set of observed instances. For example, when we fit an ML model, it is a process of induction. The ML model is a generalization of the specific examples in the training dataset. For instance, using linear regression when fitting the model to the training data generalizes specific examples in the training data by the function $Y = a + bX$. Such generalizations are made in inductive learning.

Deductive learning refers to using general rules to determine specific outcomes. The outcomes of deductive learning are deterministic and specific, whereas for inductive reasoning, the conclusions are probabilistic or generalized. In a way, deduction is the reverse of induction. If induction goes from the specific to the general, deduction goes from the general to the specific.

Transductive learning is a method for reasoning about outcomes based on specific training data samples (in the training dataset). This method is different from inductive learning, where predictions are generalized over the training data. In transductive learning, specific or similar data samples from the training data are compared to reason about or predict an outcome. For example, in the case of the *k*-nearest neighbors algorithm, it uses specific data samples on which to base its outcome rather than generalizing the outcome or modeling with the training data.

HITL models

There are two types of **HITL** models: **human-centered reinforcement learning** models and **active learning** models. In these models, human-machine collaboration enables the algorithm to mimic human-like behaviors and outcomes. A key driver for these ML solutions is the *human in the loop (hence HITL)*. Humans validate, label, and retrain the models to maintain the accuracy of the model:

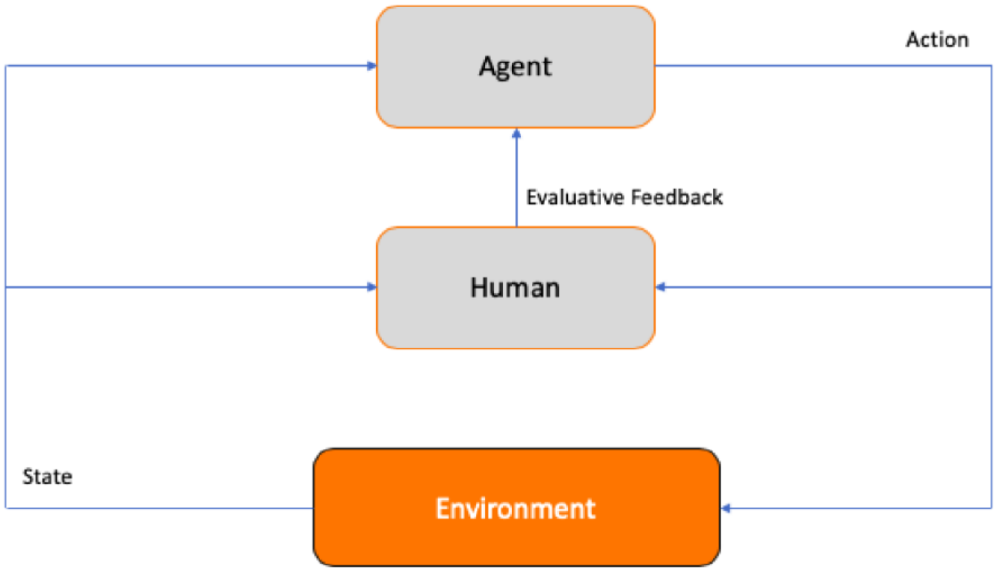

Figure 2.8 – Workflow of human-centered reinforcement learning

Human-centered reinforcement learning is a hybrid of reinforcement learning, as it involves humans in the loop to monitor the agent's learning and provide evaluative feedback to shape the learning of the agent. Human-centered reinforcement learning is also known as *interactive reinforcement learning*. Each time the agent takes action, the observing human expert can provide evaluative feedback that describes the quality of the selected action taken by the agent based on the human expert's knowledge, as shown in *Figure 2.8*.

Based on the feedback received from the task environment and human expert, the agent augments its behavior and actions. Human reinforcement learning is highly efficient in environments where the agent has to learn or mimic human behavior. To learn more, read the paper *Human-Centered Reinforcement Learning: A Survey* (`https://ieeexplore.ieee.org/abstract/document/8708686`).

Active learning is a method where the trained model can query a HITL (the human user) during the inference process to resolve incertitude during the learning process. For example, this could be a question-answering chatbot asking the human user for validation by asking yes or no questions.

These are the types of ML solutions possible to build for production to solve problems in the real world. Now that you are aware of the possibilities for crafting ML solutions, as the next step, it is critical to categorize your MLOps in line with your business and technological needs. It's important for you to be able to identify the right requirements, tools, methodology, and infrastructure needed to support your business and MLOps, hence we will look into structuring MLOps in the next section.

Structuring your MLOps

The primary goal of MLOps is to make an organization or set of individuals collaborate efficiently to build data and ML-driven assets to solve their business problems. As a result, overall performance and transparency are increased. Working in silos or developing functionalities repeatedly can be extremely costly and time-consuming.

In this section, we will explore how MLOps can be structured within organizations. Getting the MLOps process right is of prime importance. By selecting the right process and tools for your MLOps, you and your team are all set to implement a robust, scalable, frugal, and sustainable MLOps process. For example, I recently helped one of my clients in the healthcare industry to build and optimize their MLOps, which resulted in 76% cost optimization (for storage and compute resources) compared to their previous traditional operations.

The client's team of data scientists witnessed having 30% of their time freed up from mundane and repetitive daily tasks (for example, data wrangling, ML pipeline, and hyperparameter tuning) – such can be the impact of having an efficient MLOps process. By implementing efficient MLOps, your team can be assured of efficiency, high performance, and great collaboration that is repeatable and traceable within your organization.

MLOps can be categorized into **small data ops**, **big data ops**, **large-scale MLOps**, and **hybrid MLOps** (this categorization is based on the author's experience and is a recommended way to approach MLOps for teams and organizations):

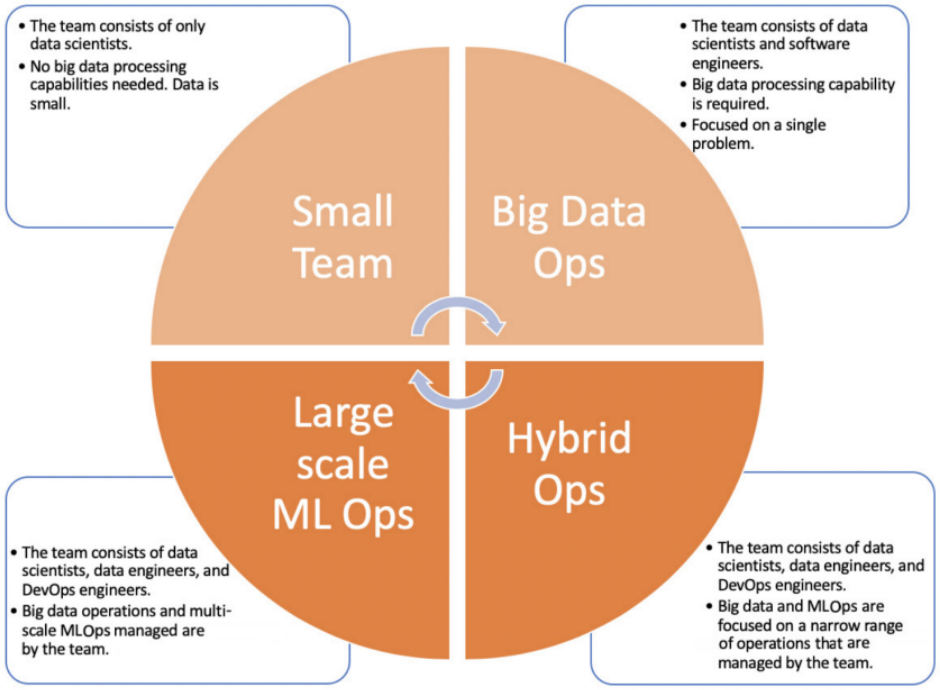
• The team consists of only data scientists.
• No big data processing capabilities needed. Data is small.
• The team consists of data scientists and software engineers.
• Big data processing capability is required.
• Focused on a single problem.
Small Team
Big Data Ops
Large scale ML Ops
Hybrid Ops
• The team consists of data scientists, data engineers, and DevOps engineers.
• Big data operations and multi-scale MLOps managed are by the team.
• The team consists of data scientists, data engineers, and DevOps engineers.
• Big data and MLOps are focused on a narrow range of operations that are managed by the team.

Figure 2.9 – Categories of MLOps

As shown in *Figure 2.9*, MLOps within organizations can be broadly categorized into four different categories depending on team size, and the ML applications, business models, data scale, tools, and infrastructure used to execute operations. In terms of data, many scenarios do not need big data (anything above 1 TB) operations, as simple operations can be effective for small- or medium-scale data. The differences between data scales are as follows:

- **Big data**: A quantity of data that cannot fit in the memory of a single typical computer; for instance, > 1 TB
- **Medium-scale data**: A quantity of data that can fit in the memory of a single server; for instance, from 10 GB to 1 TB
- **Small-scale data**: A quantity of data that easily fits in the memory of a laptop or a PC; for instance, < 10 GB

With these factors in mind, let's look into the MLOps categories to identify the suitable process and scale for implementing MLOps for your business problems or organization.

Small data ops

A small start-up with a team of data scientists seeking to build ML models for narrow and well-defined problems can be agile and highly collaborative. Usually, in such cases, ML models are trained locally on the respective data scientists' computers and then forgotten about, or scaled out and deployed on the cloud for inference. In these scenarios, there can be some general pitfalls, such as the team lacking a streamlined CI/CD approach for deploying models. However, they might manage to have central or distributed data sources that are managed carefully by the team, and the training code can be versioned and maintained in a central repository. When operations start to scale, such teams are prone to the following:

- Running into situations where much of the work is repeated by multiple people including tasks such as crafting data, ML pipelines doing the same job, or training similar types of ML models.
- Working in silos and having minimal understanding of the parallel work of their teammates. This leads to less transparency.
- Incurring huge costs, or higher costs than expected, due to the mundane and repeated work.
- Code and data starting to grow independently.
- Artifacts not being audited and hence are non-repeatable.

Any of these can be costly and unsustainable for the team. If you are working in a team or have a setup like the following, you can categorize your operations as small data ops:

- The team consists of only data scientists.
- You only work with Python environments and manage everything in the Python framework. Choosing Python can be a result of having many ML libraries and tools ready to plug and play for quick prototyping and building solutions. The number of ML libraries for a language such as Java, for example, is quite a lot smaller compared to those available for Python.
- Little to no big data processing is required as the data scientists use small data (<10 GB).
- Quick ML model development starts with a local computer, then scales out to the cloud for massive computation resources.
- High support requirements for open source technologies such as PyTorch, TensorFlow, and scikit-learn for any type of ML, from classical learning to deep, supervised, and unsupervised learning.

Big data ops

This can be a team of experienced data scientists and engineers working in a start-up or an SME where they have the requirement for large-scale big data processing to perform ML training or inference. They use big data tools such as **Kafka**, **Spark**, or **Hadoop** to build and orchestrate their data pipelines. High-powered processors such as GPUs or TPUs are used in such scenarios to speed up data processing and ML training. The development of ML models is led by data scientists and deploying the models is orchestrated by data/software engineers. A strong focus is given to developing models and less importance is placed on monitoring the models. As they continue with their operations, this type of team is prone to the following:

- A lack of traceability for model training and monitoring
- A lack of reproducible artifacts
- Incurring huge costs, or more than expected, due to mundane and repeated work
- Code and data starting to grow independently

Any of these can be costly and unsustainable for a team.

If you are working in a team or have a setup as described in the following points, you can categorize your operations as big data ops:

- The team consists of data scientists/engineers.
- There are high requirements for big data processing capacity.
- Databricks is a key framework to share and collaborate inside teams and between organizations.
- ML model development happens in the cloud by utilizing one of many ML workflow management tools such as **Spark MLlib**.
- There are low support requirements for open source technologies such as PyTorch and TensorFlow for deep learning.

Hybrid MLOps

Hybrid teams operate with experienced data scientists, data engineers, and DevOps engineers, and these teams make use of ML capabilities to support their business operations. They are further ahead in implementing MLOps compared to other teams. They work with big data and open source software tools such as PyTorch, TensorFlow, and scikit-learn, and hence have a requirement for efficient collaboration. They often work on well-defined problems by implementing robust and scalable software engineering practices. However, this team is still prone to challenges such as the following:

- Incurring huge costs, or more than expected, due to mundane and repeated work to be done by data scientists, such as repeating data cleaning or feature engineering.
- Inefficient model monitoring and retraining mechanisms.

Any of these can be costly and unsustainable for the team.

If you are working in a team or have a setup as described in the following points, you can categorize your operations as Hybrid Ops:

- The team consists of data scientists, data engineers, and DevOps engineers.
- High requirement for efficient and effective collaboration.
- High requirement for big data processing capacity.
- High support requirements for open source technologies such as PyTorch, TensorFlow, and scikit-learn for any kind of ML, from classical to deep learning, and from supervised to unsupervised learning.

Large-scale MLOps

Large-scale operations are common in big companies with large or medium-sized engineering teams consisting of data scientists, data engineers, and DevOps engineers. They have data operations on the scale of big data, or with various types of data on various scales, veracity, and velocity. Usually, their teams have multiple legacy systems to manage to support their business operations. Such teams or organizations are prone to the following:

- Incurring huge costs, or more than expected, due to mundane and repeated work.
- Code and data starting to grow independently.

- Having bureaucratic and highly regulated processes and quality checks.
- Highly entangled systems and processes – when one thing breaks, everything breaks.

Any of these can be costly and unsustainable for the team.

If you are working in a team or have a setup as described in the following points, you can categorize your operations as large-scale ops:

- The team consists of data scientists, data engineers, and DevOps engineers.
- Large-scale inference and operations.
- Big data operations.
- ML model management on multiple resources.
- Big or multiple teams.
- Multiple use cases and models.

Once you have characterized your MLOps as per your business and technological needs, a solid implementation roadmap ensures smooth development and implementation of a robust and scalable MLOps solution for your organization. For example, a fintech start-up processing 0-1,000 transactions a day would need small-scale data ops compared to a larger financial institution that needs large-scale MLOps. Such categorization enables a team or organization to be more efficient and robust.

An implementation roadmap for your solution

Having a well-defined method and milestones ensures the successful delivery of the desired ML solution (using MLOps methods). In this section, we will discuss a generic implementation roadmap that can facilitate MLOps for any ML problem in detail. The goal of this roadmap is to solve the problem with the right solution:

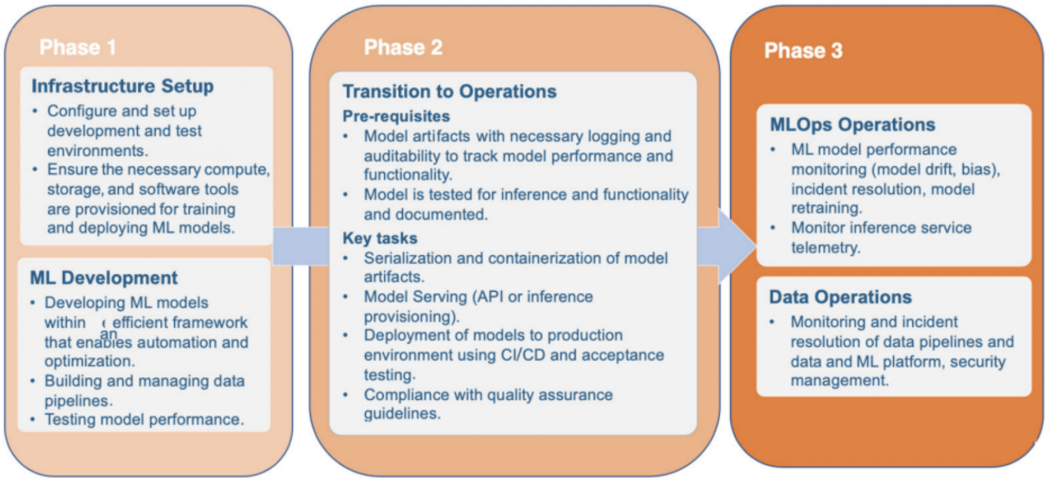
Phase 1
Infrastructure Setup
Configure and set up development and test environments.
Ensure the necessary compute, storage, and software tools are provisioned for training and deploying ML models.
ML Development
Developing ML models within an efficient framework that enables automation and optimization.
Building and managing data pipelines.
Testing model performance.
Phase 2
Transition to Operations
Pre-requisites
Model artifacts with necessary logging and auditability to track model performance and functionality.
Model is tested for inference and functionality and documented.
Key tasks
Serialization and containerization of model artifacts.
Model Serving (API or inference provisioning).
Deployment of models to production environment using CI/CD and acceptance testing.
Compliance with quality assurance guidelines.
Phase 3
MLOps Operations
ML model performance monitoring (model drift, bias), incident resolution, model retraining.
Monitor inference service telemetry.
Data Operations
Monitoring and incident resolution of data pipelines and data and ML platform, security management.

Figure 2.10 – Implementation roadmap for an MLOps-based solution

Using the preceding roadmap, we can transition from ML development to MLOps with clear milestones, as shown in these three phases for MLOps implementation. Now, let's look into these three phases of the roadmap in more detail. It's worth noting that after the following section on theory, we will get into the practical implementation of the roadmap and work on a real-world business use case.

Phase 1 – ML development

This is the genesis of implementing the MLOps framework for your problem; before beginning to implement the requirements, the problem and solution must be clear and vivid. In this phase, we take into account the system requirements to design and implement a robust and scalable MLOps framework. We begin by selecting the right tools and infrastructure needed (storage, compute, and so on) to implement the MLOps.

When the infrastructure is set up, we should be provisioned with the necessary workspace and the development and test environments to execute ML experiments (training and testing). We train the ML models using the development environment and test the models for performance and functionality using test data in the development or test environments, depending on the workflow or requirement. When infrastructure is set up and the first ML model is trained, tested, serialized, and packaged, phase 1 of your MLOps framework is set up and validated for robustness. Serializing and containerizing is an important process to standardize and get the models ready for deployments.

Next, we move to implement phase 2.

Phase 2 – Transition to operations

Phase 2 is about transitioning to operations, and it involves serializing and containerizing the models trained in phase 1 and getting them ready for deployment. This enables standardized, efficient deployments. The models are served in the form of APIs or independent artifacts for batch inference. When a model is packaged and ready to be served, it is deployed in the production environment using streamlined CI/CD pipelines upon passing quality assurance checks. By the end of phase 2, you will have packaged models served and deployed in the production environment performing inference in real time.

Phase 3 – Operations

Phase 3 is the core operations phase for deployed models in phase 2. In this phase, we monitor the deployed model performance in terms of model drift, bias, or other metrics (we will delve into these terms and metrics in the coming chapters). Based on the model's performance, we can enable continual learning via periodic model retraining and enable alerts and actions. Simultaneously, we monitor logs in telemetry data for the production environment to detect any possible errors and resolve them on the go to ensure the uninterrupted working of the production system. We also manage data pipelines, the ML platform, and security on the go. With the successful implementation of this phase, we can monitor the deployed models and retrain them in a robust, scalable, and secure manner.

In most cases, all three phases need to be implemented for your ML solution, but in some cases just phases 1 and 2 are enough; for instance, when the ML models make batch inferences and need not do inference in real time. By achieving these milestones and implementing all three phases, we have set up a robust and scalable ML life cycle for our applications systematically and sustainably.

Procuring data, requirements, and tools

Implementing successful MLOps depends on certain factors such as procuring appropriate training data, and having high standards, and appropriate requirements, tools, and infrastructure.

In this section, we will delve into these factors that make robust and scalable MLOps.

Data

I used to believe that learning about data meant mastering tools such as Python, SQL, and regression. The tool is only as good as the person and their understanding of the context around it. The context and domain matter, from data cleaning to modeling to interpretation. The best tools in the world won't fix a bad problem definition (or lack of one). Knowing what problem to solve is a very context-driven and business-dependent decision. Once you are aware of the problem and context, it enables you to discern the right training data needed to solve the problem.

Training data is a vital part of ML systems. It plays a vital role in developing ML systems compared to traditional software systems. As we have seen in the previous chapter, both code and training data work in parallel to develop and maintain an ML system. It is not only about the algorithm but also about the data. There are two aspects to ensure you have the right data for algorithm training, which are to provide both the right quantity and quality of data:

- **Data quantity**: Data scientists echo a common argument about their models, arguing that model performance is not good because the quantity of data they were given was not sufficient to produce good model performance. If they had more data, the performance would have been better – are you familiar with such arguments? In most cases, more data might not really help, as quality also is an important factor. For instance, your models can learn more insights and characteristics from your data if you have more samples for each class. For example, if you analyze anomalous financial transactions with many samples in your data, you will discover more types of anomalous transactions. If there is only one anomalous case, then ML is not useful.

  The data requirements for ML projects should not solely focus on data quantity itself, but also on the quality, which means the focus should not be on the number of data samples but rather on the diversity of data samples. However, in some cases, there are constraints on the quantity of data available to tackle some problems. For example, let's suppose we work on models to predict the churn rate for an insurance company. In that case, we can be restricted to considering data from a limited period or using a limited number of samples due to the availability of data for a certain time period; for example, 5 years (whereas the insurance company might have operated for the last 50 years). The goal is to acquire data of the maximum possible quantity and quality to train the best-performing ML models.

- **Data quality**: Data quality is an important factor for training ML models; it impacts model performance. The more comprehensive or higher the quality of the data, the better the ML model or application will work. Hence the process before the training is important: cleaning, augmenting, and scaling the data. There are some important dimensions of data quality to consider, such as consistency, correctness, and completeness.

  Data consistency refers to the correspondence and coherence of the data samples throughout the dataset. Data correctness is the degree of accuracy and the degree to which you can rely on the data to truly reflect events. Data correctness is dependent on how the data was collected. The sparsity of data for each characteristic (for example, whether the data covers a comprehensive range of possible values to reflect an event) reflects data completeness.

  With an appropriate quantity of good-quality data, you can be sure that your ML models and applications will perform above the required standards. Hence, having the right standards is vital for the application to perform and solve business problems in the most efficient ways.

Requirements

The product or business/tech problem owner plays a key role in facilitating the building of a robust ML system efficiently by identifying requirements and tailoring them with regard to the scope of data, collection of data, and required data formats. These requirements are vital inputs for developers of ML systems, such as data scientists or ML engineers, to start architecting the solution to address the problem by analyzing and correlating the given dataset based on the requirements. ML solution requirements should consist of comprehensive data requirements. Data requirement specifications consist of information about the quality and quantity of the data. The requirements can be more extensive; for example, they can contain estimations about anticipated or expected predictive performance expressed in terms of the performance metrics determined during requirements analysis and elicitation.

There is a surge in services provided by popular cloud service providers such as Microsoft, AWS, and Google, which are complemented by data processing tools such as Airflow, Databricks, and Data Lake. These are crafted to enable ML and deep learning, for which there are great frameworks available such as scikit-learn, Spark MLlib, PyTorch, TensorFlow, MXNet, and CNTK, among others. Tools and frameworks are many, but procuring the right tools is a matter of choice and the context of your ML solution and operations setup. Having the right tools will ensure high efficiency and automation for your MLOps workflow. The options are many, the sky's the limit, but we have to start from somewhere to reach the sky. For this reason, we will look to give you some hands-on experience from here onward. It is always better to learn from real-life problems, and we will do so by using the real-life business problem described in the next section.

Discussing a real-life business problem

We will be implementing the following business problem to get hands-on experience. I recommend you read this section multiple times to get a good understanding of the business problem; it makes it easier to implement it.

Important note

Problem context:

You work as a data scientist in a small team with three other data scientists for a cargo shipping company based in the port of Turku in Finland. 90% of the goods imported into Finland come via cargo ships at the ports across the country. For cargo shipping, weather conditions and logistics can be challenging at times at the ports. Rainy conditions can distort operations and logistics at the ports, which can affect the supply chain operations. Forecasting rainy conditions in advance gives the possibility to optimize resources such as human resources, logistics, and transport resources for efficient supply chain operations at ports. Business-wise, forecasting rainy conditions in advance enables ports to reduce operational costs by up to ~20% by enabling efficient planning and scheduling of human resources, logistics, and transport resources for supply chain operations.

Task:

You as a data scientist are tasked with developing an ML-driven solution to forecast weather conditions 4 hours in advance at the port of Turku in Finland. That will enable the port to optimize its resources, thereby enabling cost-savings of up to 20%. To get started, you are provided with a historic weather dataset covering a timeline of 10 years from the port of Turku (the dataset can be accessed in the next chapter). Your task is to build a continuous-learning-driven ML solution to optimize operations at the port of Turku.

To solve this problem, we will use Microsoft Azure, one of the most widely used cloud services, and MLflow, an open source ML development tool, to get hands-on with using resources. This way, we will get experience working on the cloud and with open source software. Before starting the hands-on implementation in the next chapter, please make sure to do the following:

1. Create a free Azure subscription from `https://azure.microsoft.com/` (takes 5 minutes).
2. Create an Azure Machine Learning service application with the name `MLOps_WS`. This can be done from your Azure portal by clicking **Create a resource**. Then type `Machine Learning` into the search field and select the **Machine Learning** option. Then, follow the detailed instructions in the next chapter (*Chapter 3, Code Meets Data*) to create the Azure Machine Learning service resource with the name `MLOps_WS`.

Now, with this, you are all set to get hands-on with implementing an MLOps framework for the preceding business problem.

Summary

In this chapter, we have learned about the ML solution development process, how to identify a suitable ML solution to a problem, and how to categorize operations to implement suitable MLOps. We got a glimpse into a generic implementation roadmap and saw some tips for procuring essentials such as tools, data, and infrastructure to implement your ML application. Lastly, we went through the business problem to be solved in the next chapter by implementing an MLOps workflow (discussed in *Chapter 1*, *Fundamentals of MLOps Workflow*) in which we'll get some hands-on experience in MLOps.

In the next chapter, we will go from theory to practical implementation. The chapter gets hands-on when we start with setting up MLOps tools on Azure and start coding to clean the data to address the business problem and get plenty of hands-on experience.

Index