Alma Mater Studiorum · Università di Bologna

DOTTORATO DI RICERCA IN
DATA SCIENCE AND COMPUTATION

Ciclo XXXIV

Settore Concorsuale: 02/A1 - FISICA SPERIMENTALE DELLE INTERAZIONI FONDAMENTALI

Settore Scientifico Disciplinare: FIS/01 - FISICA SPERIMENTALE

Machine Learning as a Service for High Energy Physics (MLaaS4HEP): a service for ML-based data analyses

PRESENTATA DA:
Luca Giommi

COORDINATORE DOTTORATO:
Prof. Daniele Bonacorsi

SUPERVISORE:
Prof. Daniele Bonacorsi

CO-SUPERVISORI:
Dott. Valentin Kuznetsov
Dott. Daniele Spiga
Dott. Claudio Grandi

Esame finale anno 2023

'Start by doing what's necessary, then do what's possible, and suddenly you are doing the impossible.'

Saint Francis of Assisi

CONTENTS

ABSTRACT

With the CERN LHC program underway, there has been an acceleration of data growth in the High Energy Physics (HEP) field. By the end of Run 2 the CERN experiments were already operating at the Peta-Byte (PB) level, producing $\mathcal{O}(100)$ PB of data each year. The upcoming HL-LHC program will extend it further to the exascale, and the usage of Machine Learning (ML) in HEP will be critical. ML techniques have been successfully used in online and offline reconstruction programs, detector simulation, object reconstruction, identification, Monte Carlo generation, and beyond. Nevertheless, the development of a ML project and its implementation for production use is a highly time-consuming task and requires specific skills. Generally, HEP analysts do not have the skills in data science to tackle such challenges on their own. Furthermore, complicating this scenario is the existing gap between HEP and ML communities which is partly due to the fact that HEP data is stored in ROOT data format, which is mostly unknown outside of the HEP community.

The work presented in this thesis is focused on the development of a ML as a Service (MLaaS) solution for HEP, aiming to provide a cloud service that allows HEP users to run ML pipelines via HTTP calls. These pipelines are executed by using the MLaaS4HEP framework, which allows reading data, processing data, and training ML models directly using ROOT files of arbitrary size from local or distributed data sources. Such a solution would help to bridge the gap between ML and HEP communities, by providing HEP users non-expert in ML with a tool that allows them to apply ML techniques in their analyses in a streamlined manner.

Over the years the MLaaS4HEP framework has been developed, validated and tested and new features have been added. A first MLaaS solution has been developed by automatizing the deployment of a platform equipped with the MLaaS4HEP framework. Then, a service with APIs has been developed, so that a user after being authenticated and authorized can submit MLaaS4HEP workflows producing trained ML models ready for the inference phase. A working prototype of this service is currently running on a VM of INFN-Cloud and is compliant to be added to the INFN Cloud portfolio of services.

The thesis is structured as follows:

Chapter 1 provides an overview of the data science world and in particular of ML. Examples of the different types of learning are provided with a particular focus on supervised learning techniques.

Chapter 2 is focused on cloud computing and the reason why it is a paradigm on the rise nowadays. Details on the main technologies it relies on are provided, as well as the structure of cloud native applications. Finally, the theme of cloud security is discussed, deepening the life cycle of an identity, in particular, the authentication and authorization

phases.

Chapter 3 is focused on the HEP world. In particular, details are provided on some HEP analyses which have subsequently been taken as use cases. Then, an overview of the LHC experiments is provided with a particular focus on CMS and all the technologies behind the CMS computing model. Finally, data science tools and applications of ML techniques in HEP are discussed.

Chapter 4 represents the original contribution of this thesis. It describes the architecture of the MLaaS4HEP framework, its validation, testing, and new features integration. The development phases of a MLaaS solution are then described, firstly by automatizing the deployment of a platform equipped with the MLaaS4HEP framework, and later by developing a service with APIs that allows an authenticated and authorized user to submit MLaaS4HEP workflows to train ML models and use them to make predictions.

Dedicated to Federico and to my growing family

1 MACHINE LEARNING

Machine Learning (ML) has transformed the way a business and organization operates, becoming a useful means to understand patterns and trends in complex data that should be used to succeed or maximize profits [1], as well as to make scientific discoveries. For example, ML techniques are used to understand how a customer thinks and behaves, to predict the future profits or losses that a company may face. This is what happens, for example, if you visit the Amazon web page and you begin to search and view some products. Indeed you will be presented with other products that are similar to what you were looking at. These recommendations are not coded into the system but are suggestions given by a ML model. The model will look at all your history and use that information to show you similar products.

Nowadays the terms "Artificial Intelligence" (AI) and "ML" are often used interchangeably. Although these are connected, there are meaningful differences. AI is the capability of a computer system to make human-like decisions and therefore mimic human cognitive functions, such as problem-solving and learning [2]. With AI a computer system uses logic and math to simulate the reasoning that allows people to learn from new information and make decisions. ML is an application of AI. Indeed, ML is referred to how a computer system develops its intelligence, i.e. it is the process of using mathematical models of data to help a computer learn without direct instruction. This allows a computer system to continue learning and improving on its own based on experience.

In this chapter, we introduce the world of data science and the figure of data scientist, where ML plays a key role. To follow, we define ML and we retrace its history. Then, we give an overview of the different types of learning in ML, and considering the content of the original contribution of this thesis described in Ch. 4, we deepen the topic of the supervised learning approach. Here we cover some of the most used algorithms, and how they are trained and evaluated.

1.1 DATA SCIENCE

The terms "data science" and "data mining" are often used interchangeably. Providing a high-level definition, data science is the set of fundamental principles that guide the extraction of knowledge from data, whereas data mining is the extraction of knowledge from data using technologies that incorporate these principles [3]. Anyway in general, the term "data science" is more broadly applied than "data mining".

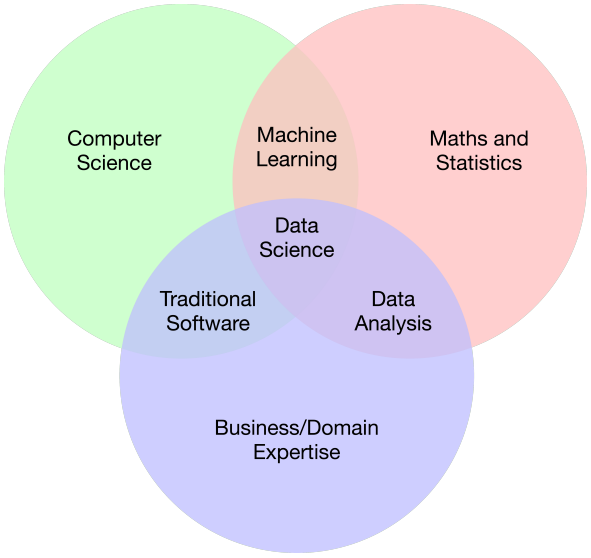


Fig. 1.1: Data science Venn diagram. It shows an overview of the skills required in the data science domain.

Data science combines several fields, including statistics, scientific methods, computer science, ML, and data analysis to extract value from data. See Fig. 1.1 for the data science Venn diagram that shows an overview of the skills required in the data science domain.

The typical data science lifecycle comprehends several steps [4–6].

1. **Problem understanding**. Understand the problem that needs to be tackled, identify the central objectives of the project, and ask relevant questions.
2. **Data collection**. Collect data from different sources.
3. **Data cleaning**. Clean and prepare data, e.g. by managing inconsistencies between values and missing values.
4. **Data exploration**. Explore data by plotting histograms or distribution curves to visualize the general trend of data. This allows you to understand data and start to formulate hypotheses.
5. **Data transformation and feature engineering**. Use the domain knowledge to transform raw data into informative features that represent the problem. In ML, a feature is an attribute or a measurable property of a phenomenon being observed.
6. **Predictive modeling**. Build one or more models (e.g. ML models), evaluate the performance using one or more metrics, and use it/them to make predictions.
7. **Findings report**. Lastly, present the obtained results, e.g. through reports or presentations.

The process described so far is iterative as probably during the first iteration of the lifecycle new insights can be obtained that can help to better tackle one or more steps of the lifecycle to generate more powerful insights and obtain better results. Additionally, parts

of the lifecycle can be automated and this practice is commonly referred to as Automated ML (AutoML)

The ultimate goal of data science is to improve decision-making, and this is of direct interest to businesses. Data-driven decision-making is the practice of basing decisions on the analysis of data rather than purely on intuition. Those involved in data science are called data scientists, who combine a wide range of skills to analyze the collected data. In 2012, the Harvard Business Review published an article with title "Data Scientist: The Sexiest Job of the 21st Century" [7] that highlighted how the role of data scientist was becoming more and more requested in the job market. This professional figure was relatively new at the time, but as more companies tried to make sense of data, they realized they needed people who could combine analytics, programming, and experimentation skills [8]. Since there was not yet a well-defined career path for people who knew to program and analyze such data, data scientists had different educational backgrounds. The most common qualification in an informal survey of 35 data scientists at the time was a Ph.D. in experimental physics, but there were also psychologists, astronomers, and meteorologists. Most had PhDs in some scientific field, were good at math, and knew how to code. Ten years later, the job is more in demand than ever by recruiters and employers. From 2012 to 2019 postings for data scientists on Indeed had risen by 256%, and the U.S. Bureau of Labor Statistics predicts that data science will have more growth than any other field between 2022 and 2029. And also the wages are pretty good: indeed for example in California, the median salary for an experienced data scientist is approaching $200,000.

The data scientist job has changed over the years.

- **Its scope has been redefined**. In 2012, a data scientist was supposed to do all required tasks in a data science application. Nowadays instead, there is a proliferation of related jobs to tackle many of the data scientist tasks, including data engineer, data analyst, data-oriented product managers, ML engineer, and AI specialist. In the "Jobs on the Rise" reports for 2021 and 2022, LinkedIn stated that some of these jobs are more popular than data scientists in U.S. This proliferation is due to the fact that no single worker can have all the skills needed to successfully implement a complex AI or analytics system. As a result, companies need to identify all the different roles necessary to effectively deploy data science models in their businesses, and make sure they are present and collaborating in teams.

- **It has become better institutionalized**. In 2012, there were no degree programs in data science and data scientists were recruited from other fields. Now there are hundreds of degree programs (most are master's degree programs but there are also Ph.D. programs) in data science or in related fields of AI and analytics. There is also a huge number of certificates and online courses in data science-related fields.

- **The technology it relies on has made great strides**. One reason why the data scientist job is changing over time is that the technologies used by data scientists are changing. Some technology trends were already there in 2012, e.g. the use of open source tools and the move to cloud-based data storage and processing, but there are

others that have caught on in recent years, e.g. some parts of the data science process are increasingly automated.

- **The ethics of data science has grown**. A major change in data science over the past decade is the need for an ethical dimension. Probably the turning point for data science ethics was the U.S. presidential election in 2016, where data scientists attempted to influence voters using social media. Since that time, great attention has been paid to issues of transparency, algorithmic bias, and responsible use of analytics and AI.

An interesting and wide survey conducted by Kaggle over 25,000 data scientists and ML engineers in 2021 gives an overview of the figure of data scientist nowadays [9]. In the following, the main information extracted from the survey are reported.

- The 82% of users identify as men.
- More than half of all data scientists are between the ages of 22 and 34.
- Data scientists live and work all around the globe: 24.4% of them reside in India, 12.2% in the U.S., and 4.3% in Brazil. These countries are the three most representative ones.
- Over 62% of data scientists obtained either a Master's or a doctoral degree.
- Coursera remains the most popular ongoing data science learning resource, followed by Kaggle Learn Courses.
- Over 60% of data scientists have not more than 5 years of programming experience, and over 75% not more than 10 years.
- More than 55% of data scientists have less than three years experience in ML.
- The mean salary for data scientists in the U.S. is between $150,000 and $200,000
- Over half of companies where data scientists work have less than 250 employees.
- Over half of data scientists still work at companies with five or fewer people on the data science team, whereas one in five work on a team with more than 20 data scientists.
- Over 70% of data scientists have spent money in the last five years on cloud computing products.
- Amazon Web Services, Google Cloud Platform, and Microsoft Azure are the three big players in cloud computing, where Amazon Web Service is the most used.
- Over 40% of data scientists use ML solutions offered by cloud providers, and the most popular choice of such products is Amazon SageMaker (see Sec. 2.2.2.1).
- Jupyter-based Integrated Development Environments (IDEs) is the go-to tool for data scientists, with over 73% of Kaggle data scientists using it. Then, Visual Studio Code (VSCode) is in the second position with 38%.

1.1.1 *Data engineering and Big Data*

Data engineering is the process of designing and building systems that allow people to collect, process, and analyze raw data from multiple sources and formats. Data engineering and processing are critical to supporting data science, but they are more general than data science [3]. The latter needs access to data and often benefits from sophisticated data engineering that data processing technologies can facilitate, but such technologies are not data science per se. Indeed, the use of data processing technologies is a key factor for many data-oriented business tasks that do not involve data-driven decision-making or extracting knowledge, e.g. modern web system processing, and efficient transaction processing.

Big data essentially are datasets that are too large for traditional data processing solutions and therefore require new processing solutions. As traditional technologies, big data technologies (e.g. Hadoop, Spark, and MongoDB) are used for many tasks, including data engineering, and only occasionally are used to implement data mining techniques, even if this has been happening more and more often in recent times, given the growth of data produced.

Big data is often described by at least five characteristics, known as the 5Vs of big data.

- **Volume**. The size and amount of data being produced and analyzed is dramatically increasing. According to a study published by the International Data Corporation (IDC) in 2021 [10, 11], an amount of 64.2 ZB[1] of data was created or replicated in 2020, and "the amount of digital data created over the next five years will be greater than twice the amount of data created since the advent of digital storage".

- **Velocity**. It refers to the speed at which data is generated, distributed, and collected. In many cases, sets of big data are updated on a real- or near-real-time basis, instead of daily, weekly or monthly updates typically made in traditional data warehouses.

- **Variety**. It refers to the many types of data that are available, including structured data (e.g. transactions and financial records), unstructured data (e.g. text, documents, and multimedia files), and semi-structured data (e.g. web server logs and streaming data from sensors).

- **Value**. The worth and usefulness of information that can be gathered by the processing and analysis of data.

- **Veracity**. It refers to the assurance of integrity/quality/accuracy of data. As data is collected from multiple sources, its accuracy must be verified before using it for business insights.

Other characteristics of big data can be added, e.g. valence, variability, and volatility. Volume, velocity, and variety of data generated and distributed over the internet give insight into how people are using technology in their daily lives. You can see an overview in Fig. 1.2 referred to data of 2020.

1 $1\,ZB = 10^{21}\,B$, i.e a million of PB.

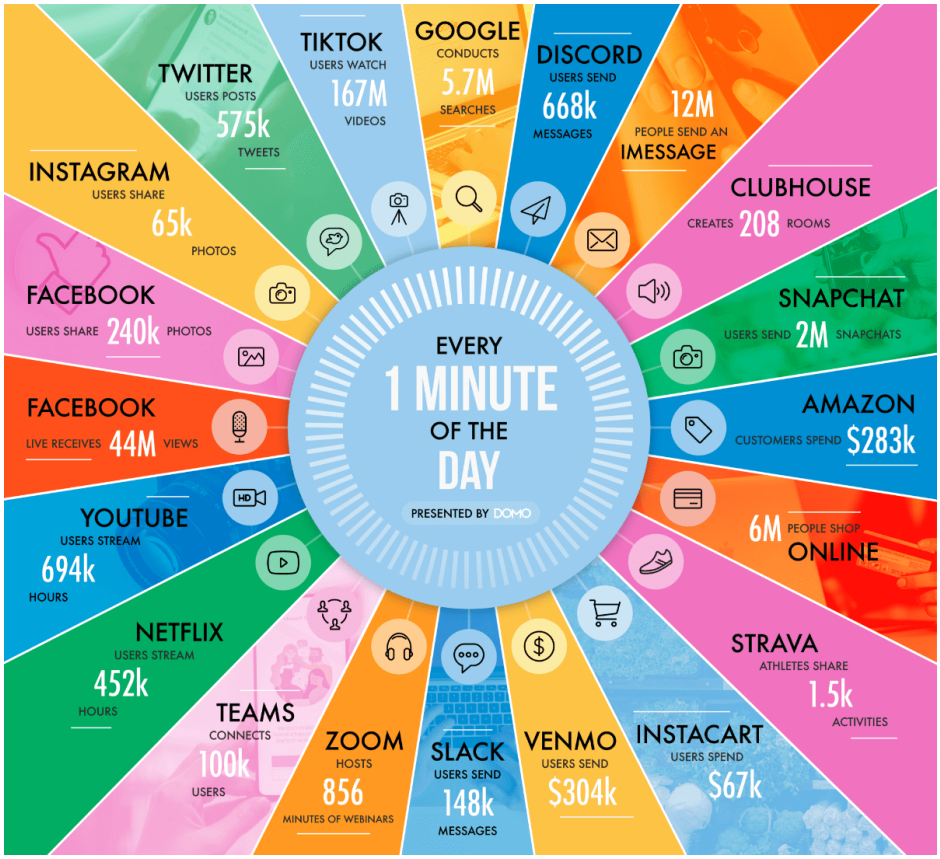


Fig. 1.2: Representation offered by the DOMO company about how much data is created every minute in our increasingly data-driven world in 2020 [12].

1.2 WHAT IS MACHINE LEARNING AND WHEN TO USE IT?

After this introduction to the data science world, let's focus our attention on the main topic of this chapter, i.e. ML. The definition of ML is relatively difficult to summarize as it is often provided and explained differently by different people or groups. Here is a first general one:

> *[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.*
>
> Arthur Samuel, 1959

And here is a more engineering-oriented definition:

> *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*
>
> Tom Mitchell, 1997

To better understand the latter definition, let's see how a ML program works, taking the spam filter as an example [13]. The spam filter uses a given set of examples of spam emails (that can be flagged by the user) and a set of examples of regular emails (i.e. not spam)

to learn how to flag emails as spam. The set of emails that the system uses to learn is called *training set*. Each element/example in the training set is called a *training instance* (or *sample*). In this situation, the task T of the filter is to flag spam new emails, the experience E is the training set, and the performance measure P needs to be defined (e.g. the ratio of correctly classified emails can be used).

Problems like the recognition of spam emails can be also tackled using traditional programming techniques. Let's see when the use of ML is great.

- When existing traditional solutions require long lists of rules or a lot of fine-tuning. Often a ML algorithm simplifies the code and performs better than a traditional approach.
- When a traditional approach does not give a good solution.
- When dealing with a fluctuating environment. In this situation, a ML solution can adapt better to new data.
- When dealing with complex problems and with a large amount of data.

ML involves the use of ML algorithms and models [14]. These two terms are often used interchangeably but have different concepts behind them. An "algorithm" in ML is a procedure (i.e. a program) that is run on data to create a ML "model". Algorithms learn from data or are fit on a dataset. Instead, a ML "model" is the output of a ML algorithm run on data, it represents what is learned by a ML algorithm. The model is what is saved after running a ML algorithm on training data: it represents the rules, numbers, and any algorithm data structures required to make predictions. Training a model means running an algorithm to find the parameters of the model that will make it best fit the training data.

1.3 TIMELINE OF MACHINE LEARNING HISTORY

ML was first conceived from the mathematical modeling of Neural Networks (NN) [15, 16]. In 1943, the logician Walter Pitts and the neuroscientist Warren McCulloch published the first mathematical modeling of a NN in order to create algorithms that mimic the processes of human thought.

In 1950, Alan Turing proposed the "Turing test" which aimed to determine if a computer has real intelligence. To pass the test, a computer must be able to fool a human into believing that it too is human. In 1952, Arthur Samuel wrote the first computer learning program. It consisted in the checkers game where an IBM computer improved at the game the more it played, understanding which moves led to winning strategies and incorporating those moves into its program. In 1956, John McCarthy met many well-known scientists and researchers for six to eight weeks at Dartmouth college to brainstorm about thinking machines. This event is considered the birth of AI. In 1957, Frank Rosenblatt designed the first NN for computers called "Perceptron", which received visual inputs like images, and created in response output like labels.

In 1963, Donald Michie developed the MENACE program able to learn how to perfectly play a match of the tic-tac-toe game. In 1967, the "nearest neighbor" algorithm was written, that granted computers the ability to use very basic pattern recognition. In particular, this algorithm was used to plan routes for traveling salesman.

In 1970, the mathematician and computer scientist Seppo Linnainmaa published the general method for automatic differentiation of discrete connected networks of nested differentiable functions. This corresponds to the modern version of backpropagation. In 1973, the British government cut the funds for research on AI in its universities. This event started the so-called "AI winter". In 1979, a group of researchers at Stanford University created a robot called "the Chart" able to navigate obstacles within a room. In the same year, Kunihiko Fukushima published a work on neocognitron, that is a hierarchical and multilayered type of Artificial NN (ANN) used for pattern recognition tasks.

In 1981, Gerald Dejong introduced the explanation-based learning where a computer learns to analyze training data and formulate a general rule to discard information it believes unimportant. In 1982, John Hopfield popularized the so-called "Hopfield network", a type of Recurrent NN (RNN), first introduced in 1974. In 1985, Terry Sejnowski created the NetTalk program which learns to pronounce words similar to how babies do. In 1986, two psychologists David Rumelhart and James McClelland introduced a framework called parallel distributed processing which uses NN models for ML purposes. In 1989, Christopher Watkins developed a model-free reinforcement algorithm called Q-learning, that looks for the best action to take in any state.

In 1992, Gerald Tesauro developed the TD-Gammon program, based on an ANN, able to play the backgammon game and rival the best players of this game. In 1995 two papers were published: one about random decision forests (a method of ensemble learning) written by Tim Kam Ho, and one about Support Vector Machines (SVMs) written by Vladimir Vapnik and Corinna Cortes. In 1996, a chess-playing computer program developed by IBM, called Deep Blue, beat Garry Kasparov who was the world champion of chess at the time. In 1997, Sepp Hochreiter and Jurgen Schmidhuber published a paper about their work on the Long Short-Term Memory (LSTM) architecture, a type of RNN. In 1998, a team led by Yann LeCun released the MNIST dataset which became widely adopted as an evaluation benchmark for handwriting recognition.

In 2002, the first open-source software library for ML, called Torch, was released. In 2006, the Netflix prize competition was launched, where the goal was to create a ML algorithm more accurate than Netflix's proprietary recommendation software for users. In the same year, Geoffrey Hinton coined the term "deep learning" referring to the algorithms that help computers in recognizing different types of objects and text characters in videos and pictures. In 2009, Fei-Fei Li invented the ImageNet database to help visual object recognition.

In 2010, Anthony Goldbloom and Ben Hamner launched the Kaggle platform, originally used for ML competitions. In 2011, using a combination of Natural Language Processing (NLP) and information retrieval techniques, IBM's Watson beat two champions on the Jeopardy game show. In 2012, a deep Convolutional NN (CNN) called AlexNet was introduced

in the ImageNet Large Scale Visual Recognition Challenge that dramatically improved accuracy in AI image recognition. In the same year, the Google Brain team, launched in 2011, created a NN that learns to recognize cats by watching images taken from frames of YouTube videos. In 2014, Facebook developed DeepFace, a software algorithm able to identify individuals on photos as accurately as a human can. In the same year, Google unveiled its proprietary platform Sibyl for massively parallel ML, internally used by Google to predict users' behavior and provide recommendations. In the same year, Ian Goodfellow and his colleagues designed the Generative Adversarial Network (GAN), that use two NNs pitted one against the other in order to generate new and artificial instances of data that can pass for real data. They are widely used in image, video, and voice generation. In 2016, the AlphaGo program developed by Google DeepMind became the first program to beat a professional human Go player using a combination of ML and tree search techniques. The AlphaGo algorithm managed to win five games out of five in the Go competition against the European champion Fan Hui.

1.3.1 *Rise of Machine Learning*

According to a study carried out by Fortune Business Insights [17], the global ML market size was valued at $15.44 billion in 2021, and it is expected to grow up to $209.91 billion by 2029. The global impact of the COVID-19 pandemic was unprecedented, with ML technology seeing higher than expected demand compared to pre-pandemic levels: the global ML market exhibited a higher growth of 36% in 2020 compared to 2019. This is attributed to a significant acceleration in the ML technology adoption in many areas, e.g. healthcare and automotive. In the pandemic scenario, the application of AI technology helped to cope with the difficult situation. For example, several countries are currently using population surveillance methods based on AI to track and trace COVID-19 cases.

Nowadays, ML (as well as also data science and AI) is becoming more and more popular, as also highlighted by Google Trends which tracks the popularity of searched terms (see Fig. 1.3).

ML is among the core technologies at the basis of most worldwide activities aiming at extracting actionable insight from data. There are many examples, such as detecting tumors in brain scans, identifying people based on pictures or voice recordings, automatically marking offensive comments on discussion forums, automatically summarizing long documents, creating a personal assistant or a chatbot, forecasting business revenue for the next few years, detecting credit card fraud, recommending a product that might interest a customer, and many others. Therefore, although AI and ML algorithms have been around for a long time, why has not the increase we see today happened earlier?

- **Rise of Big Data**. There is an abundance and growing of data collected and stored right now, data that has not been collected for individuals or at scale before. They can come from a myriad of sources, e.g. transaction processing systems, documents, customer databases, emails, medical records, internet clickstream logs, mobile apps, and social networks [19]. Big data also includes machine-generated data, e.g. server

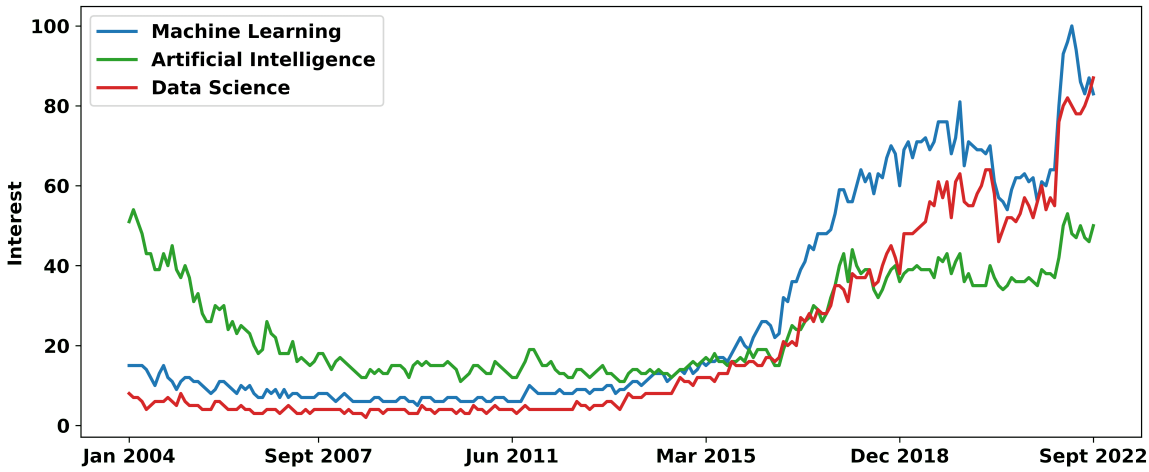


Fig. 1.3: Interest over time for ML, AI, and data science in terms of search frequency on Google. The numbers on the y-axis represent the search interest with respect to the highest point of the graph in relation to the indicated period. The value 100 indicates the highest search frequency of the term, 50 indicates half of the searches, while a score of 0 indicates that not enough data was found for the term. The plot is produced using official data from Google Trends [18].

and network log files, data from sensors on industrial equipment, manufacturing machines, and internet of things devices.

- **Technology progresses**. The development of big data technologies, and the development of computing resources (based on CPUs and GPUs) which over time have become increasingly abundant and cheaper, have made the processing of big data possible as well as the use of increasingly complex models, considerably reducing the time required for single operations. In addition, the rise of cloud computing allowed access to storage and computing power with pay-as-you-go pricing enabling the "democratization" of resources (which anyone can access, by paying).

1.4 MACHINE LEARNING LIBRARIES AND FRAMEWORKS

There are many libraries and frameworks that provide access to many ML algorithms through different programming languages. According to a report by the SlashData company [20], about 70% of ML developers and data scientists use Python as programming language for ML. In comparison, only 17% use R. As result, Python-based tools dominate the ML domain. According to the survey conducted by Kaggle in 2021 [9], Scikit-learn is the most used ML library, followed by TensorFlow, XGBoost, Keras, and PyTorch (see Fig. 1.4). Tensorflow, Keras, and Pytorch are designed for NNs, are open-source, and provide GPU support.

- **TensorFlow** is a framework developed by Google and released in 2015. It is based on graph computation, and thanks to TensorBoard it allows a developer to better visualize the NN construction and it makes the debugging easier. Moreover, Tensorflow allows the usage of other algorithms besides NNs, like Decision Forests.

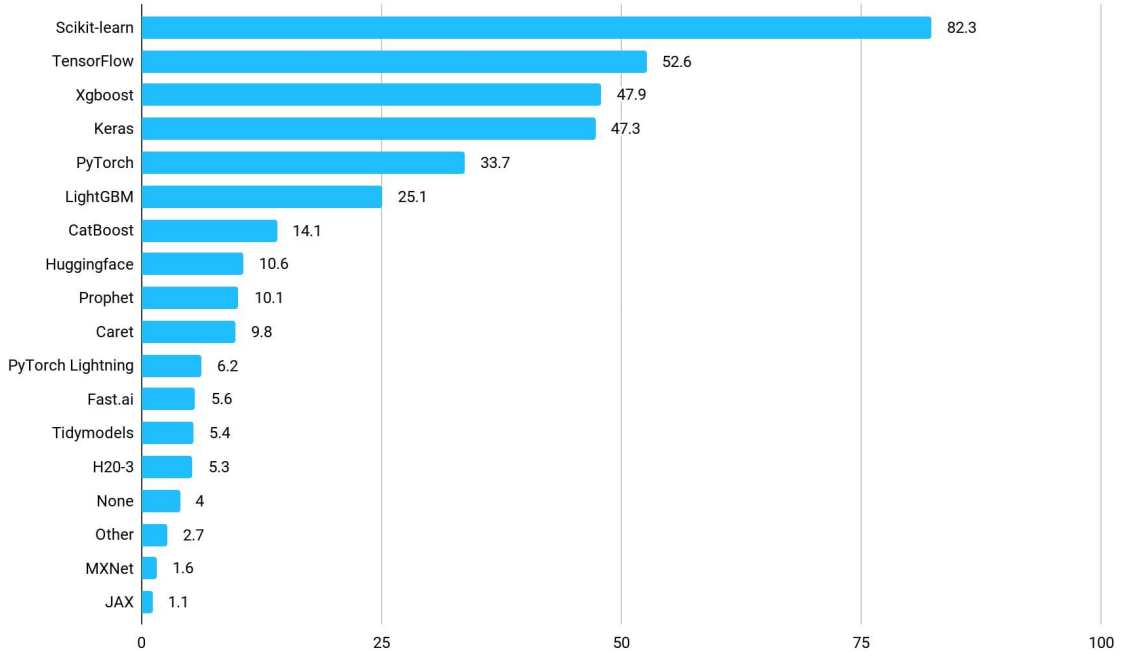


Fig. 1.4: ML frameworks and libraries percentage usage according to the survey conducted by Kaggle in 2021 [9].

- **Keras** is a library released in 2015. Up until version 2.4, Keras supported multiple backends, including TensorFlow and Theano, but as of the official version 2.6, only TensorFlow is supported. Keras functions are a wrapper to TensorFlow framework, meaning that a user can define an algorithm with Keras, which is easier to use, then drop down into TensorFlow when he/she needs to use a feature that Keras does not have.
- **PyTorch** is a framework based on Torch library and on tensor computing, developed by the Facebook AI research group and open-sourced on GitHub in 2017.
- **Scikit-learn** is an open-source library released in 2007 that includes a variety of ML algorithms, and also tools for model selection and data pre-processing. It is based on other libraries, like Matplotlib, Pandas, and NumPy. It does not provide any GPU support.
- **XGBoost** (eXtreme Gradient Boosting) is an open-source library released in 2014 which provides an implementation of Gradient Boosted Decision Trees (more details in Sec. 1.8.1.5). It provides GPU support.

1.5 TYPES OF LEARNING

There are different types of learning in ML, which can be classified into categories that are not exclusive and can be freely combined by following some simple criteria.

- Whether ML algorithms are trained with human supervision, hence supervised, unsupervised, semisupervised, and reinforcement learning.
- Whether ML algorithms can learn incrementally on the fly, hence online and batch learning.
- Whether ML algorithms work by comparing new data points to known data points, or by detecting patterns in training data and building a predictive model, hence instance-based and model-based learning.

1.5.1 *Supervised/unsupervised learning*

Depending on the amount and type of supervision that ML algorithms get during the training, four main categories can be identified: supervised, unsupervised, semisupervised, and reinforcement learning.

1.5.1.1 *Supervised learning*

In supervised learning, the training data given to the algorithm includes the desired solutions, called labels. There are two different types of supervised learning tasks: classification and regression. In the former, classification algorithms are used to predict/classify discrete values, such as true or false, male or female, spam or not spam. In the latter, regression algorithms are used to predict continuous values, such as age, price, and salary. Some of the most important supervised learning algorithms are: Linear Regression, Logistic Regression, SVM, k-Nearest Neighbor (kNN), Decision Tree and Random Forest, NN. Some NN architectures can be unsupervised, e.g. autoencoders, and also semisupervised.

Supervised learning algorithms are used in many areas, e.g. image- and object-recognition, predictive analytics, sentiment analysis, and spam detection, just to mention a few.

Let's see now a couple of examples to better understand supervised problems [21]. Let's suppose to have a collection of data about Italian housing prices and plot the data (see Fig. 1.5) where the size of different houses in square meters is shown on the x-axis, and the price of different houses in thousands of euros is shown on the y-axis. Given this data, we want to know how much value we can get for a house of 150 square meters. A learning algorithm can fit a straight line to data (pink straight line in Fig. 1.5), and based on that it looks like the house can be sold for about €275,000. A better solution that the learning algorithm can find is to fit a quadratic function or a second-order polynomial to data (blue curved line in Fig. 1.5). If we use this model to make a prediction we can state that the house can be sold for about €325,000. Each of these two different approaches is an example of a supervised learning algorithm: the term "supervised" here refers to the fact that we gave the algorithm a dataset containing the right answers, i.e. for each entry in this dataset we knew the right price, that is used by the algorithm to learn how to produce predictions for new houses. This is also a regression problem since here we are trying to predict a continuous value output, i.e. the price of a house. In this example, to simplify we used

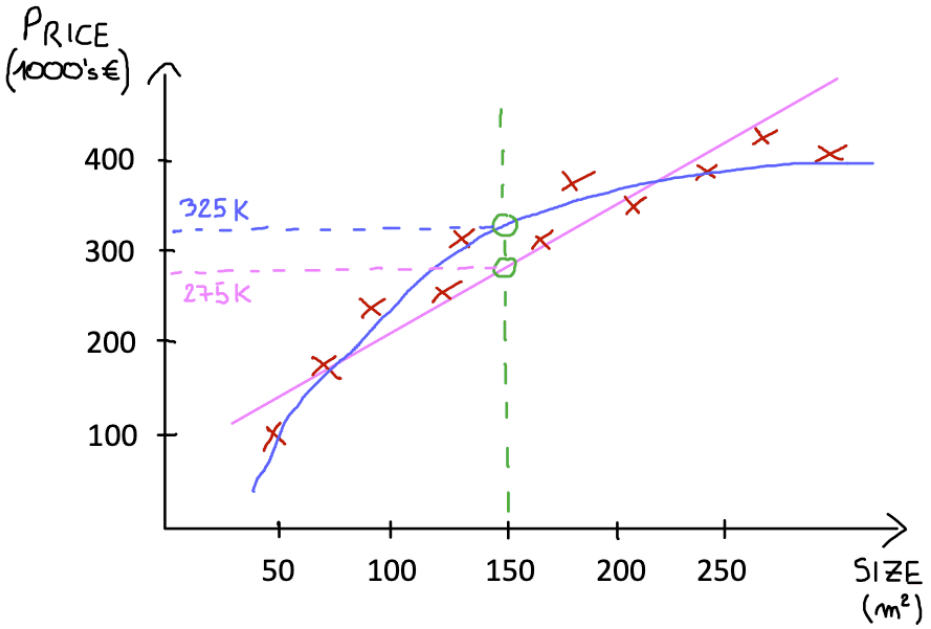


Fig. 1.5: Example of regression problem tackled in supervised learning. Details in the text.

prices as discrete values, but usually the price of a house is a scalar number in the domain of the real numbers.

As another example of supervised learning, let's suppose to look at medical records where we want to predict the nature of breast cancer as benign or malignant. We can take the medical dataset and plot the size of the tumor on the x-axis and a binary value (1 or 0) on the y-axis depending on the malignant/benign nature of the tumor, as in Fig. 1.6 in which four examples of benign tumors and four examples of malignant tumors are shown. Let's suppose to have a patient with a breast tumor with a known size, and we want to know which is the probability that this breast tumor is benign or malignant. This is an example of classification problem since we are trying to predict a discrete value output, i.e. 0 or 1 (benign or malignant). In typical classification problems, often it happens to have more than two values as possible output. For example, let's suppose to have 3 types of breast cancers, and we want to predict the discrete values of 0 (benign), 1, 2, and 3 (all malignant). This is still a classification problem, or better a multiclass classification problem: these other discrete values in the output set correspond to no cancer, or cancer of type 1, or 2, or 3. In these examples, we are using only one attribute (i.e. the tumor size) to predict whether an additional tumor in the dataset is benign or malignant.

Typically, in ML problems there are many attributes (i.e. features). In our example, additionally we could know the age of the patient and add it as information to the size of the tumor (see Fig. 1.7). In this example, a learning algorithm should be able to throw a straight line through the data displayed on a two-dimensional plot that best separate malignant tumors from benign tumors.

In real-life cases, there are many more attributes (e.g. in our example we can have tumor thickness, uniformity of cell size of the tumor, uniformity of cell shape of the tumor), up to a very large number.

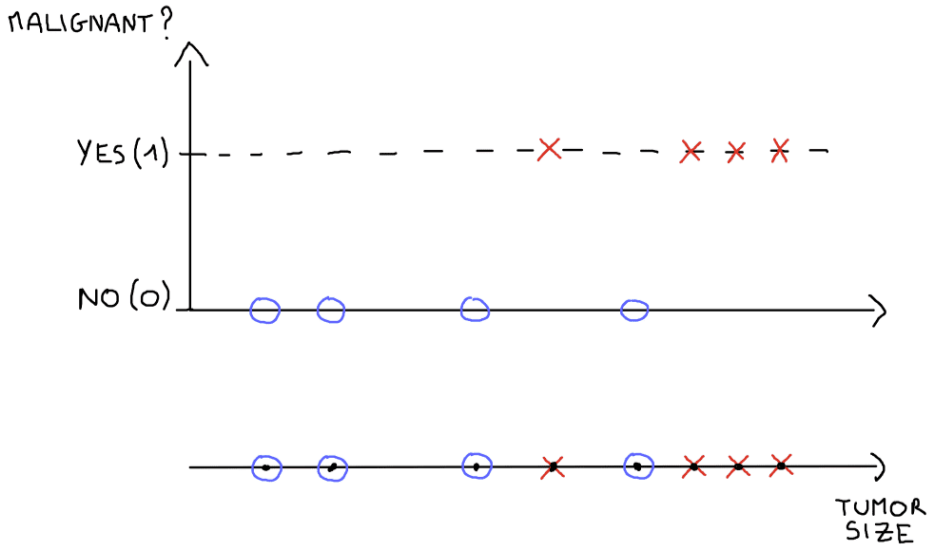


Fig. 1.6: Example of classification problem tackled in supervised learning. Details are in the text.

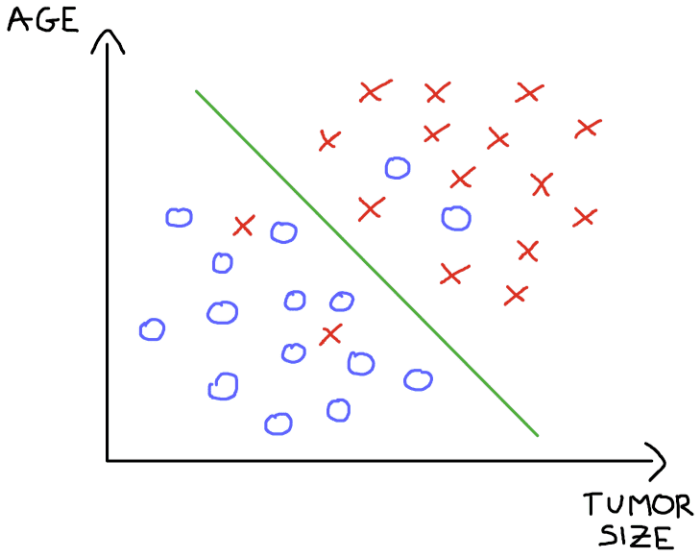


Fig. 1.7: Example of classification problem tackled in supervised learning with two features. Details are in the text.

1.5.1.2 *Unsupervised learning*

In unsupervised learning, the training data is unlabeled [13]. Unsupervised learning is concerned with recognizing patterns in data, and the objective is not to forecast a particular variable, rather it is to understand the environment represented by the data. The different approaches to unsupervised learning with some of the most important related algorithms are: clustering (e.g. K-Means, DBSCAN), anomaly and novelty detection (e.g. One-class SVM), visualization and dimensionality reduction (e.g. Principal Component Analysis - PCA), association rule learning (e.g. Apriori).

Unsupervised learning algorithms are used in many areas, e.g. market segmentation, recommender systems, genetic and species grouping in biology, and computer security research, just to mention a few.

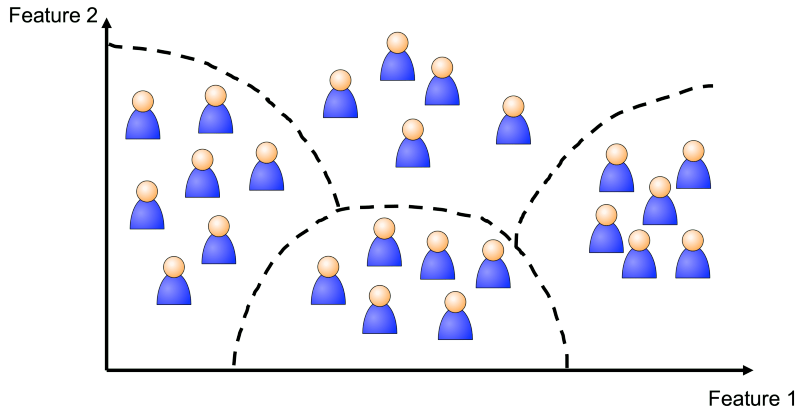


Fig. 1.8: Example of clusters based on two features that group similar visitors.

Let's take an example where you have data about your blog visitors and you want to detect groups of similar visitors. A clustering algorithm can be used where you do not tell the algorithm which group a visitor belongs to, but the algorithm itself will find those connections without your help. For example, it might notice that 30% of your visitors are young females who love novels and generally read your blog in the afternoon, while 30% are cooking lovers who visit during the weekends. Using a hierarchical clustering algorithm, it may also break down each group into smaller groups, and this may help you target the posts you write for each group. See Fig. 1.8 for a simple representation of boundaries drawn by a clustering algorithm based on two features, which divide visitors into four different groups.

A different approach can be explained through the so-called "cocktail party problem". In this example, a party takes place in a room full of people where everyone is talking at the same time, so with overlapping voices, making it difficult for everyone to understand what others are saying. There are microphones in the room, at different distances from each speaker, that record a different combination of any speaker's voice with different volumes depending on the distance. In this condition, a specific unsupervised learning algorithm that implements Independent Component Analysis (ICA) can be used to find structures in microphone recordings, identifying the source voice patterns, despite the different volume levels in different microphone recordings, to ultimately disentangle them.

1.5.1.3 *Semisupervised learning*

In semisupervised learning, the training data is partially labeled. Usually, labeling data is time-consuming and costly, and so could happen to have plenty of unlabeled instances, and few labeled instances. Most semisupervised learning algorithms are combinations of supervised and unsupervised algorithms. For example, Deep Belief Networks (DBNs) are based on unsupervised components, called Restricted Boltzmann Machines (RBMs), stacked on top of one another. RBMs are trained sequentially in an unsupervised manner and subsequently, the whole system is fine-tuned using supervised learning techniques.

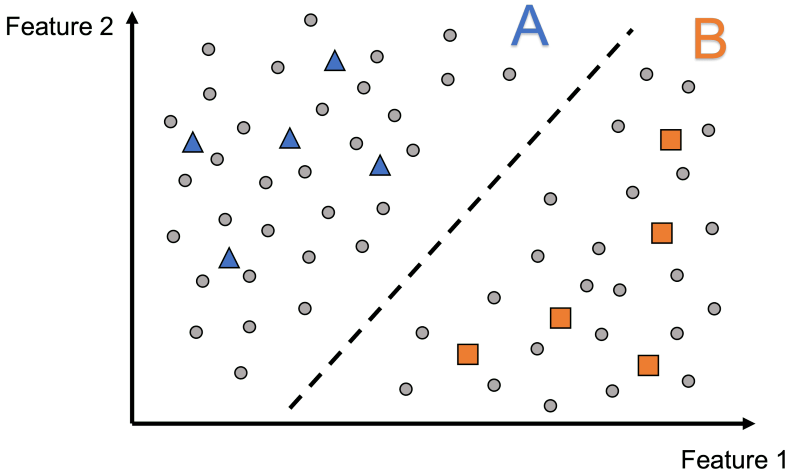


Fig. 1.9: Example of semisupervised learning with two classes, purchasers (A) and non-purchasers (B). Details are in the text.

Semisupervised learning algorithms are used in many areas, e.g. speech analysis, web content classification, protein sequence classification, and text document classifier, just to mention a few.

Let's take an example where we are interested in predicting whether a customer will purchase a particular product using some features, like age, income level, and so on [22]. Suppose further that we have a small amount of labeled data (where labels indicate whether customers bought the product or not) and a much larger amount of unlabeled data. We can use unsupervised learning to cluster potential customers. In this situation two clusters, A and B, are formed: the purchasers from the labeled data all correspond to points in cluster A (blue triangles in Fig. 1.9) while the non-purchasers from the labeled data all correspond to points in the other Cluster B (orange squares in Fig. 1.9). We might reasonably classify all individuals in Cluster A as buyers and all individuals in Cluster B as non-buyers, even those of unlabeled data (gray circles in Fig. 1.9).

Another example is Google Photos [13]. Once you upload all your family photos to this service, it automatically recognizes that the same person A shows up in photos 2, 7, and 21, while another person B shows up in photos 4, 8, and 14. This one just described is the unsupervised part of the algorithm (clustering). Subsequently, the system asks you who these people are, and after you provide the information, the service is able to name everyone in every photo, resulting in a useful tool for searching photos.

1.5.1.4 *Reinforcement learning*

In reinforcement learning, the learning system (called agent) learns to behave in an environment by performing actions and seeing the results. For each good action, the agent gets a reward in return, and for each bad action, the agent gets a penalty. The agent must learn by itself what is the best strategy, called policy, to get the maximum reward over time. Some of the main used reinforcement learning algorithms are Q-Learning and State–action–reward–state–action (SARSA).

Reinforcement learning algorithms are used in many areas, e.g. autonomous driving, industry automation, trading and finance, and gaming, just to mention a few.

Let's take as example of reinforcement learning a toddler learning not to touch a hot cup of milk [23]. The toddler's experience typically includes a set of occasions when the toddler is in front of a hot cup of milk and has to decide whether to touch the cup or not. Presumably, every time he touches it, the result is a high level of pain, and every time he does not touch it, a much lower level of pain results (i.e. that of an unsatisfied curiosity). Eventually, the toddler learns that it is best for him not to touch the hot cup. The training examples do not explain what the toddler should do, but instead classify the different actions that he takes. However, he uses the examples to reinforce the better actions, learning what he should do in similar situations.

Reinforcement learning algorithms are particularly useful for learning how to play a game. Imagine a situation in chess where you can choose between different actions and you want to identify the best action. It is not a trivial task to find which action is the best at a certain stage of the game, so we cannot easily create supervised learning examples. Instead, if you use reinforcement learning, all you need to do is to take some action and report how well things went, creating in this way a training example. The task of the reinforcement learning algorithm is to find the best line of play using the information coming from the different examples.

1.5.2 *Batch and online learning*

Depending on whether a system can learn incrementally from a stream of incoming data or not, two types of learning are identified: online learning and batch learning [13].

In online learning, the ML model is incrementally trained by feeding it data instances sequentially, and this can be done individually or in small groups called mini-batches. This approach is great for systems that receive data as a continuous flow and when you have limited computing resources, since the learning is fast and cheap. Indeed, once an online learning system has learned about new data instances, they can be discarded as the system does not need them anymore. Online learning algorithms can also be used when dealing with huge datasets that cannot fit in the RAM of the training node. An important parameter of online learning algorithms is the learning rate, i.e. how fast the algorithms should adapt to changing data. With a high learning rate, the algorithm rapidly adapts to new data, but it also tends to quickly forget the old data. Conversely, with a low learning rate, the algorithm learns more slowly, but it will also be less sensitive to noise (i.e. random fluctuations) in new data or to outliers[2].

In batch learning (also called offline learning), the ML algorithm is incapable of learning incrementally and therefore the ML model must be trained using all the available data. Since it generally takes a lot of time and computing resources, it is typically done offline. If you want a batch learning algorithm to know about new data, the model should be trained from scratch on the full dataset (containing both old and new data), and replace

2 Sample that differs significantly from the others.

the old trained ML model with the new one. Unfortunately, training a ML model using the entire dataset can take many hours and a lot of computing resources (and therefore also a high cost), so if the ML model needs to adapt to rapidly changing data, a more reactive solution is needed and an online learning algorithm should be used.

1.5.3 *Instance-based versus model-based learning*

Depending on the ability of a ML algorithm to generalize to new data, two types of learning are identified: instance-based and model-based learning.

In instance-based learning, a ML algorithm learns the samples by heart and then generalizes to new cases by using a similarity measure to compare them to the learned samples.

In model-based learning, a model of given examples is built and then used to make predictions. More in detail the steps to perform are: study the data, select the algorithm/s fully specifying the architecture, define a performance measure/metric (that can be either a utility/fitness function that measures how good the model is or a cost function that measures how bad it is), train the model/s on the training data, and finally apply the model to make predictions on new cases. This is what a typical ML project looks like.

1.6 ISSUES IN MACHINE LEARNING PROJECTS

When a data scientist decides to use ML to address a specific problem, he/she may be faced with a number of issues, which can be summarized in bad data and bad ML algorithm.

- **Insufficient quantity of training data**. A ML algorithm to work properly generally needs a lot of data: even for simple problems you typically need thousands of examples, while for complex problems (e.g. image or speech recognition) you may need millions of examples.
- **Nonrepresentative training data**. Training data must be representative of new cases you want to generalize to and this is not always easy to have. If not, you will have *sampling noise* when the sample is too small, while you will have *sampling bias* with very large samples.
- **Poor-quality data**. If training data is full of errors, outliers, and noise (e.g., due to poor measurements quality), a ML algorithm will have difficulty detecting the underlying patterns. For this reason, a crucial phase in the data science lifecycle is to clean up the training data.
- **Irrelevant features**. A ML algorithm will only be capable to learn if the training data contains enough relevant features and not too many irrelevant features. So a critical part of the success of a ML project is coming up with a good set of features, and this process is called feature engineering.
- **Overfitting training data**. We say overfitting when the model performs well on the training data but it does not generalize well. If the training set is noisy or if it is too

small, then the model is likely to detect patterns in the noise itself, and obviously, these patterns will not generalize to new instances. The aim is to have the right balance between perfectly fitting the training data and keeping the model simple enough to ensure generalization. The means used to reduce the risk of overfitting is called regularization and the amount of regularization to apply during the learning can be controlled by a specific hyperparameter. A hyperparameter is a parameter specific of the ML algorithm (not of the model), and it is not affected by the learning procedure itself: it is set before training and remains constant during training. If you set a very large value for the regularization hyperparameter, the learning algorithm will almost certainly not overfit the training data, but at the same time, it will be less likely to find a good solution. Hyperparameters tuning is a crucial part of building a good ML model.

- **Underfitting training data**. Underfitting occurs when your model is too simple to learn the underlying structure of data. The problem can be fixed by selecting a more powerful model (with more parameters), using better features (feature engineering), and reducing constraints on the model (e.g. reducing regularization).

1.7 FEATURE ENGINEERING TECHNIQUES

Feature engineering is the process of selecting, manipulating, and transforming raw data into features that can be used to improve the performance of ML models. There are several techniques that help in this procedure and should be adopted or at least tried in ML projects [24].

- **Imputation**. Imputation deals with handling missing values in data. Actually, sometimes this phase is considered a part of the data cleaning phase in the data science lifecycle. There are two types of imputation: categorical and numerical. In the former, missing categorical values are generally replaced by the most commonly occurring value in other records. In the latter, missing numerical values are generally replaced by the mean of the corresponding value in other records (or for example with a 0).

- **Discretization**. It consists of grouping sets of data together into a bin according to some logic. This could help prevent data from overfitting but comes at the cost of loss of granularity of data.

- **Categorical encoding**. It is the technique used to encode categorical features into numerical values that usually are simpler for an algorithm to be understood. One Hot Encoding is a popular technique of categorical encoding where categorical values are converted into simple numerical 1's and 0's without loss of information. As drawbacks it could result in a great increase in the number of features, and in the creation of highly correlated features.

- **Feature splitting**. Splitting features into two or more features can sometimes improve the value of the features themselves.

- **Creating and adding features**. It involves deriving new features from existing ones. This can be done by using simple mathematical operations to obtain for example the mean, median, mode, sum, difference, or product of values of two or more features.

- **Dimensionality reduction**. As the number of features or dimensions grows, the amount of data a model has to generalize grows exponentially (a problem known as the "curse of dimensionality"). There are two options to reduce dimensionality: feature elimination and feature extraction. In the former, statistical-based feature selection methods can be used to evaluate the relationship between features, in order to select only those that have the strongest relationship with the target and to remove highly correlated features. In feature extraction, new features are created to replace the original ones. A common technique adopted is PCA: it linearly transforms the data into a new coordinate system (of new features) where most of the variation in data can be described with fewer dimensions than the initial data.

- **Handling outliers**. Outliers are unusually low or high values in the dataset that are unlikely to occur in normal scenarios. The presence of outliers could negatively affect your prediction and they must be handled appropriately. There are various methods of handling outliers, e.g. removal and replacing (where the outliers are treated as missing values and replaced by using appropriate imputation).

- **Variable transformations**. They could help to normalize skewed data. A popular method is the logarithmic transformation, which allows you to compress larger numbers and relatively expand smaller numbers. This results in less skewed values, especially in the case of heavy-tailed distributions.

- **Scaling**. After a scaling operation, the continuous features become similar in terms of range. This operation is not required for many algorithms, but for example distance-based algorithms like k-NN and k-Means require scaled continuous features as model input. The commonly used processes of scaling are normalization and standardization. In the former, all the values in a feature are rescaled in the range 0 to 1 according to Eq. 1.1. In the latter, all the values in a feature are rescaled according to Eq. 1.2 (where μ is the mean and σ is the standard deviation of the training samples), so they have unit variance.

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1.1}$$

$$X_{new} = \frac{X - \mu}{\sigma} \tag{1.2}$$

1.8 FOCUS ON SUPERVISED LEARNING

In Sec. 1.1, we described the data science lifecycle as a series of steps, and the key phase in extracting knowledge from data is what we called predictive modeling. This can be

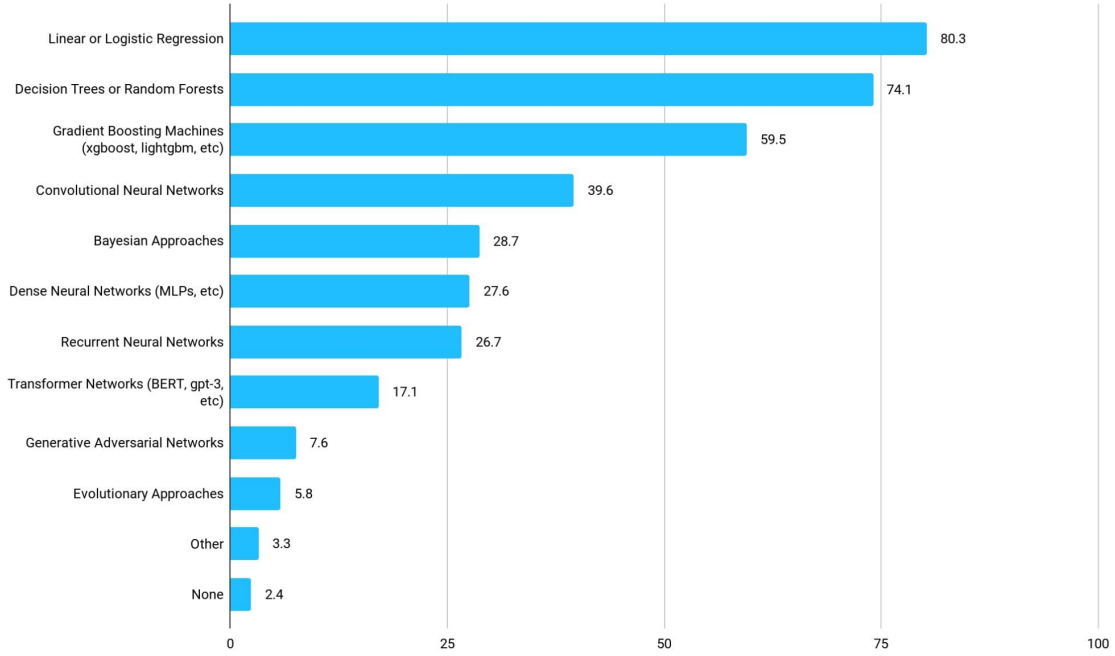


Fig. 1.10: ML algorithms percentage usage according to the survey conducted by Kaggle in 2021 [9].

achieved by exploiting ML algorithms typically with model-based learning, that in turn consists of a series of phases, as described in Sec. 1.5.3. Considering the content of the original contribution of this thesis described in Ch. 4, in this section we describe the predictive modeling phase in the specific case of supervised learning (with a focus on the classification task), giving an overview of how most common algorithms works in Sec. 1.8.1, which are the typical metrics used to evaluate a trained model in Sec. 1.8.2, and how a typical training and testing procedure works in Sec. 1.8.3.

1.8.1 *Most common algorithms*

As already introduced in Sec. 1.5.1.1, there are two different types of supervised learning tasks: classification and regression. Some regression algorithms can be used for classification, and vice versa.

In the following sections, we provide details on some of the most used algorithms: Linear Regression, Logistic Regression, SVM, kNN, Decision Tree and Random Forest, and NN. See Fig. 1.10 for a representation of the percentage usage of the main ML algorithms (not only of type supervised) according to the survey conducted by Kaggle in 2021 [9].

1.8.1.1 *Linear Regression*

A Linear Regression model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the *bias term* (also called the *intercept term*), as shown in Eq. 1.3.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n \tag{1.3}$$

where $\hat{y}$ is the predicted value, n is the number of features, x_i is the i^{th} feature value, θ_j is the j^{th} model parameter (including the bias term θ_0 and the feature weights θ_1, θ_2, ..., θ_n). This equation can be written more concisely using the vectorized form, as shown in Eq. 1.4.

$$\hat{y} = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta} \cdot \boldsymbol{x} \tag{1.4}$$

where $\boldsymbol{\theta}$ is the model parameter vector, $\boldsymbol{x}$ is the instance feature vector with x_0 always equal to 1, $\boldsymbol{\theta} \cdot \boldsymbol{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\boldsymbol{x}$ (also written as $\boldsymbol{\theta}^\intercal x$, where $\boldsymbol{\theta}^\intercal$ is the transpose of $\boldsymbol{\theta}$), $h_{\boldsymbol{\theta}}$ is the hypothesis function using the model parameter $\boldsymbol{\theta}$. We recall that training a model means setting the parameters so that the model best fits the training set, and for this purpose a measure of how well (or poorly) the model fits the training data should be chosen. The most common performance measure of a regression model is the Root Mean Square Error (RMSE), which is the square root of the Mean Squared Error (MSE). The MSE of a Linear Regression hypothesis $h_{\boldsymbol{\theta}}$ on a training set $\mathbf{X}$ of size m is calculated with Eq. 1.5. To train a Linear Regression model, the value of $\boldsymbol{\theta}$ that minimizes the RMSE should be computed, and it is simpler to minimize the MSE than the RMSE, since both lead to the same result.

$$MSE(\boldsymbol{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^\intercal \boldsymbol{x}^{(i)} - y^{(i)} \right)^2 \tag{1.5}$$

In order to find the value of $\boldsymbol{\theta}$ that minimizes the cost function, there is a closed-form solution, i.e. a mathematical equation that directly gives the result, called Normal Equation (Eq. 1.6).

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^\intercal \boldsymbol{X} \right)^{-1} \boldsymbol{X}^\intercal \boldsymbol{y} \tag{1.6}$$

In this equation, $\hat{\boldsymbol{\theta}}$ is the value of $\boldsymbol{\theta}$ that minimizes the cost function, and $\boldsymbol{y}$ is the vector of target values (containing $y^{(1)}$, ..., $y^{(m)}$). The computational complexity of inverting the $\boldsymbol{X}^\intercal \boldsymbol{X}$ matrix is about $\mathcal{O}\left(n^{2.4}\right)$ to $\mathcal{O}\left(n^3\right)$ (where n is the number of features) depending on the implementation, whereas using the Singular Value Decomposition (SVD) technique [25] the computational complexity is about $\mathcal{O}\left(n^2\right)$. Both the Normal Equation and the SVD approach become very slow when the number of features increases. However, there is a very different way to train a Linear Regression model that is better suited for cases where there are many features or too many training instances to fit in memory, called *Gradient Descent*.

The basic idea of Gradient Descent is to iteratively adjust parameters to minimize a cost function. It starts by filling θ with random values (called random initialization), then it measures the local gradient of the error function with regard to the parameter vector $\boldsymbol{\theta}$, and it goes gradually with small steps in the direction of the descending gradient (the

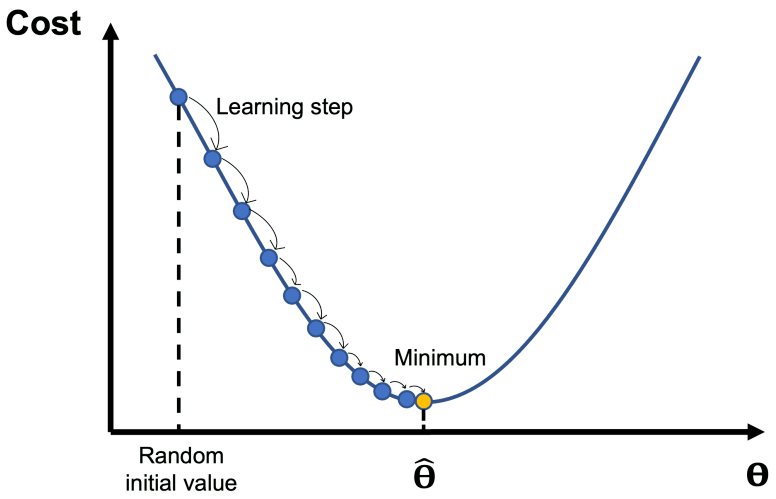


Fig. 1.11: Representation of how Gradient Descent works. The model parameters are initialized randomly and are adjusted repeatedly to minimize the cost function. The learning step size is proportional to the slope of the cost function so that gradually the steps get smaller as the parameters get closer to the minimum.

steepest slope), each time attempting to decrease the cost function (e.g. MSE) until the algorithm converges to the minimum (see Fig. 1.11).

An important parameter in Gradient Descent is the learning step size, proportional to the slope of the cost function and to the learning rate hyperparameter. If the learning rate is too small, then the algorithm will take many iterations to converge and therefore it will take a long time, whereas if the learning rate is too high, it might jump across the concavity and end up on the other side. This might cause the algorithm to diverge failing to find a good solution. Finally, not all cost functions have a regular shape and they can have more than one local minimum, making convergence to the global minimum difficult. Fortunately, in a Linear Regression model, the MSE cost function is a convex function, so there are no local minima but only one global minimum, and since it is a continuous function with a slope that never changes abruptly, the Gradient Descent is guaranteed to approach the global minimum. A good practice when using Gradient Descent is to make sure that all features have a similar scale (and therefore implement feature scaling if necessary), otherwise it will take much longer to converge.

To implement Gradient Descent, the gradient vector of the MSE with regard to the vector of parameters $\boldsymbol{\theta}$ must be computed (Eq. 1.7), which contains all the partial derivatives of the cost function, one for each model parameter θ_j.

$$\nabla_{\boldsymbol{\theta}} MSE\left(\boldsymbol{\theta}\right) = \frac{2}{m}\boldsymbol{X}^{\intercal}\left(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\right) \tag{1.7}$$

Finally, the $\boldsymbol{\theta}$ vector after a Gradient Descent step is computed with Eq. 1.8

$$\boldsymbol{\theta}^{(next\ step)} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} MSE\left(\boldsymbol{\theta}\right) \tag{1.8}$$

where η is the learning rate. Eq. 1.7 involves calculations over the full training set $\mathbf{X}$ at each Gradient Descent step, and for this reason the algorithm is called *Batch Gradient*

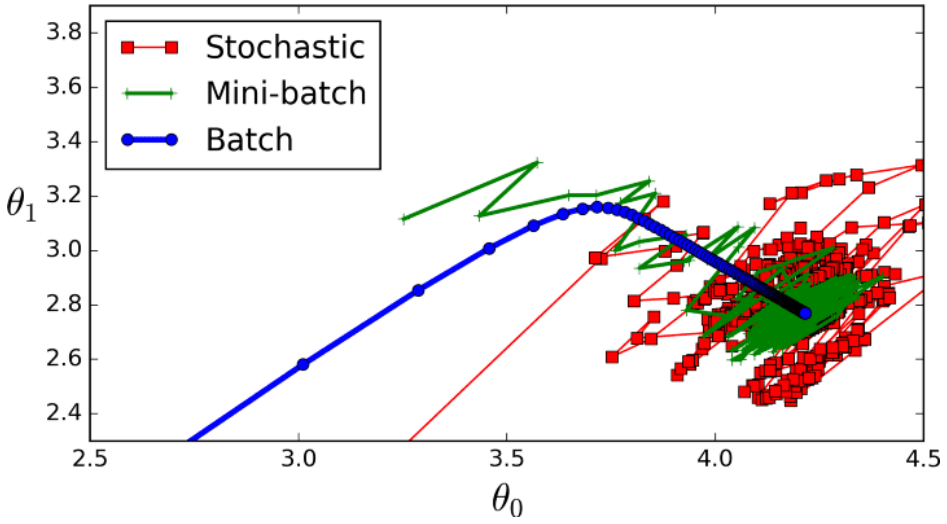


Fig. 1.12: Example of Stochastic, Mini-batch, and Batch Gradient Descent paths in a two-dimensional parameters space [26].

Descent. As a result, it is very slow on very large training sets. However, Gradient Descent scales well with the number of features, so training a Linear Regression model using Gradient Descent when dealing with many features is much faster than using the Normal Equation or SVD decomposition.

Another solution is the *Stochastic Gradient Descent* which selects a random instance in the training set at every step and computes the gradient on that single instance, getting much faster. On the other hand, due to its stochastic (random) nature, this algorithm is much less regular than Batch Gradient Descent. Indeed, instead of smoothly decreasing until it reaches the minimum, the cost function goes up and down, decreasing only on average. And when it is close to the minimum, it continues to bounce around, never settling down (red line in Fig. 1.12). Stochastic Gradient Descent is useful when the cost function is very irregular since it can help the algorithm to jump out of local minima, and has a better chance of finding the global minimum than Batch Gradient Descent.

The last Gradient Descent algorithm is the *Mini-batch Gradient Descent*. At each step it computes the gradient on small random sets of instances called mini-batches, resulting in less erratic progress in parameter space and a bit closer to the minimum than with Stochastic GD (green line in Fig. 1.12), but it may be harder for it to escape from local minima.

A Linear Regression model can be also used to fit nonlinear data, e.g. you can just add powers or products of features as new features, and then train the linear model on the extended set of features. This is called Polynomial Regression.

We have already introduced the concept of overfitting in Sec. 1.6. It happens when the model performs well on the training data but it does not generalize well. To reduce overfitting we have to regularize the model (i.e., to constrain it): indeed the fewer degrees of freedom the model has, the harder it will be for it to overfit data. To regularize a polynomial model you can simply reduce the number of polynomial degrees. For a linear model, regularization is typically obtained by constraining the weights of the model. There are

three main ways to do it in Linear Regression models, called Ridge Regression, Lasso Regression, and Elastic Net.

In the *Ridge Regression* (also called *L2*), a regularization term (also called l_2 penalty) equal to $\alpha \sum_{i=1}^{n} \theta_i^2$ is added to the cost function, where the α hyperparameter controls how much you want to regularize the model: when $\alpha = 0$ the Ridge Regression is simply Linear Regression, whereas for large values of α all weights become very close to zero and the model becomes a flat line going through the data mean. The Ridge Regression cost function (identified with $J(\boldsymbol{\theta})$) is:

$$J(\boldsymbol{\theta}) = MSE(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2 \tag{1.9}$$

In the *Lasso Regression* (also called *L1*), a regularization term (also called l_1 penalty) equal to $\alpha \sum_{i=1}^{n} |\theta_i|$ is added to the cost function, which becomes:

$$J(\boldsymbol{\theta}) = MSE(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} |\theta_i| \tag{1.10}$$

The Lasso Regression tends to eliminate the weights of the least important features, setting them to zero. In other words, it performs feature selection producing a sparse model (i.e. with few nonzero feature weights).

In the *Elastic Net Regression*, the regularization term is a mix of both Ridge and Lasso regularization terms, that can be controlled by the mix ratio term r. The Elastic Net cost function is:

$$J(\boldsymbol{\theta}) = MSE(\boldsymbol{\theta}) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} \theta_i^2 \tag{1.11}$$

Now the question is, when should you use plain Linear Regression (i.e., without regularization), Ridge, Lasso, or Elastic Net Regression? In general, it is always good to have at least a little bit of regularization. Ridge is a good default, but if you think that only a few features are useful, then you should use Lasso or Elastic Net. Moreover, Elastic Net is in general preferred over Lasso because Lasso behaves erratically when the number of features is greater than the number of training instances or when several features are highly correlated.

1.8.1.2 *Logistic Regression*

Logistic Regression (also called Logit Regression) is commonly used to estimate the probability that an instance belongs to a particular class. When the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (also called the positive class, labeled with "1"), if not it predicts that the instance does not belong to that class (and so it belongs to the negative class, labeled with "0"). In this way, the model becomes a binary classifier. A Logistic Regression model is similar to a Linear Regression model since it computes a weighted sum of the input features adding a bias term, but at the end, it gives as output the *logistic* of the result (see Eq. 1.12).

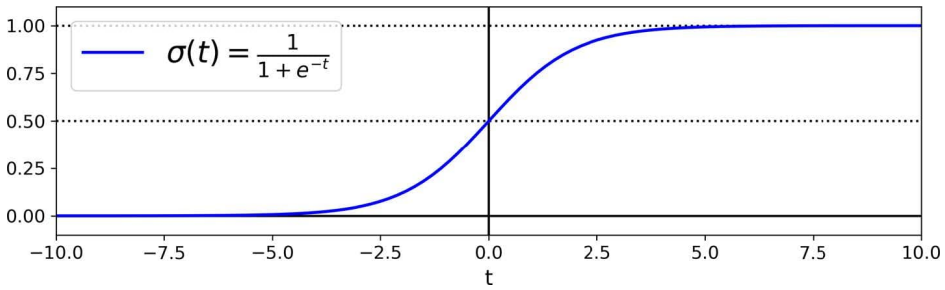


Fig. 1.13: Logistic function [13].

$$\hat{p} = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma\left(\boldsymbol{x}^{\intercal}\boldsymbol{\theta}\right) \tag{1.12}$$

The logistic $\sigma(\cdot)$ is a sigmoid function that gives a number between 0 and 1 as output. It is defined by Eq. 1.13 and the trend can be seen in Fig. 1.13.

$$\sigma(t) = \frac{1}{1 + exp(-t)} \tag{1.13}$$

The objective of the Logistic Regression training is to set the parameter vector $\boldsymbol{\theta}$ so that the model gives high probabilities for positive instances (y = 1) and low probabilities for negative instances (y = 0). The cost function over the whole training set is called *log loss* and is reported in Eq. 1.14.

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) log\left(1 - \hat{p}^{(i)}\right)\right] \tag{1.14}$$

There is no closed-form equation to compute the value of $\boldsymbol{\theta}$ that minimizes this cost function (equivalent to the Normal Equation), but since the cost function $J(\boldsymbol{\theta})$ is convex, the Gradient Descent or any other optimization algorithm is guaranteed to find the global minimum. Once the gradient vector containing all the partial derivatives is computed, it can be used in the Batch, Mini-batch or Stochastic Gradient Descent algorithm. Moreover, Logistic Regression models can be regularized using l_1 or l_2 penalties.

The decision boundary of a Logistic Regression model is a hyperplane in the space of the θ_i parameters. For example, the decision boundary is linear when dealing with two features.

The Logistic Regression model can be generalized to support multiple classes directly. This model is called *Softmax Regression* and the corresponding cost function is called *cross entropy*.

1.8.1.3 *Support Vector Machine*

The SVM is a powerful and versatile ML algorithm, capable to perform linear or nonlinear classification, regression, and even outlier detection. In particular, it is well suited for classification of complex small- or medium-sized datasets. The idea of a Linear SVM classifier can be explained using Fig. 1.14, where two classes can be easily separated with a straight line.

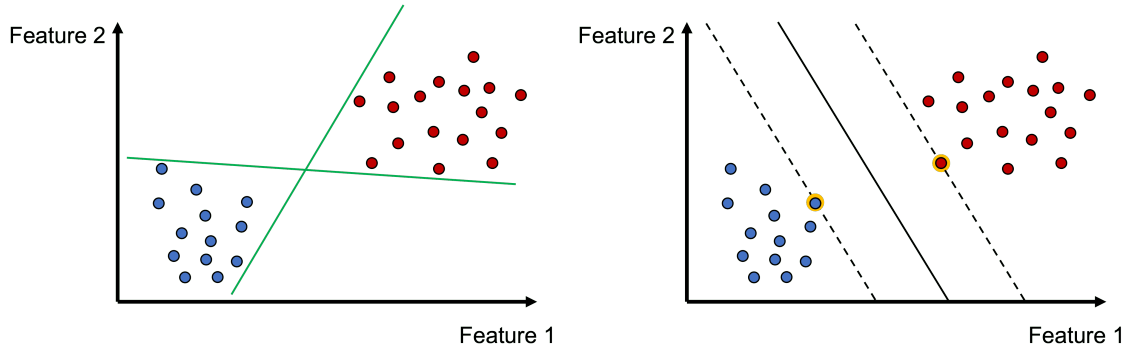


Fig. 1.14: Difference between boundaries obtained by two different linear classifiers (left) and the boundary given by SVM (right).

The plot on the left shows the decision boundaries of two linear classifiers (e.g. Logistic Regression): they are fine nevertheless are so close to some instances that the models will probably not perform as well on new instances. Whereas the plot on the right shows the decision boundary of a Linear SVM classifier: here the straight line not only separates the two classes but also stays as far away as possible from the closest training instances. Adding more training instances outside the "street" (area bounded by the two dashed lines) will not affect the decision boundary, since it is fully determined by the instances crossed by the dashed lines (that are circled in the right plot of Fig. 1.14). These instances are called support vectors.

Moreover, SVMs are sensitive to the scale of the features, so make sure that all features have a similar scale (and therefore implement feature scaling if necessary).

If we strictly impose that all instances must be off the street, this is called *hard margin classification*, but it only works if the data is linearly separable, and it is sensitive to outliers. To avoid issues, a more flexible model can be used that can find a good balance between keeping the street as large as possible and limiting the margin violations (i.e. instances on the wrong side or in the middle of the street). This is called *soft margin classification*. The SVM algorithm has many hyperparameters and one of them allows to manage the margin violations.

Although Linear SVM classifiers are efficient and perform very well in many cases, many datasets are not linearly separable. As we have already seen for Linear Regression, one approach is to add more features (e.g. polynomial features) in such a way that this can result in a linearly separable dataset. But this approach is not always good: indeed at a low polynomial degree, it cannot deal with very complex datasets, whereas with a high polynomial degree it produces a huge number of features making the model very slow. Fortunately, in SVMs you can change the kernel hyperparameter to have the same results without actually adding new features. The most common kernels you can use are: Linear, Polynomial, Gaussian RBF, and Sigmoid.

SVC, NuSVC, and LinearSVC are classes in Scikit-learn that implement SVC, and are capable of performing binary and multi-class classification on a dataset (see Fig. 1.15)

One method to implement an online SVM classifier (which means learning incrementally, typically as new instances arrive) is to use Stochastic Gradient Descent (e.g., using

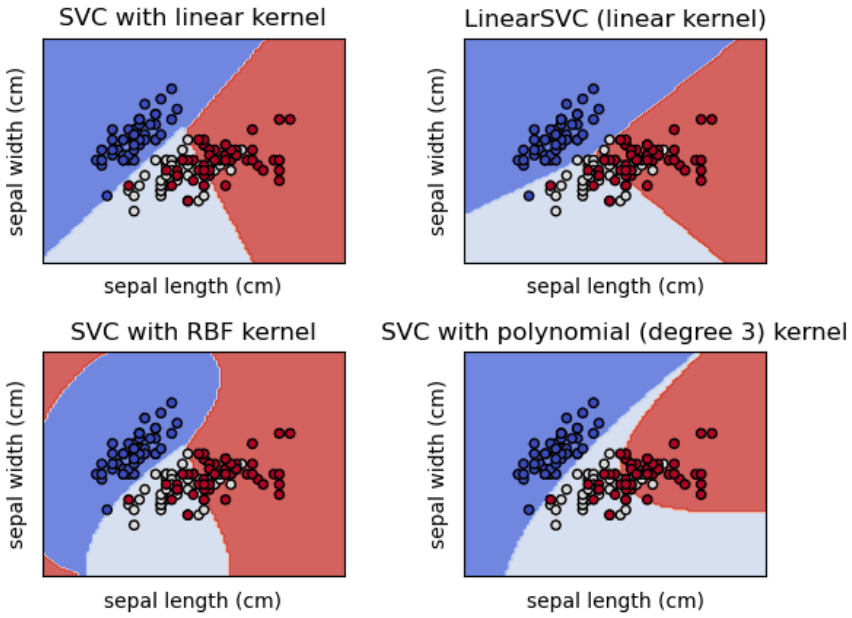


Fig. 1.15: Decision boundaries obtained by different implementations of SVC in Scikit-learn [27].

the SGDClassifier of Scikit-learn) to minimize the cost function of a Linear SVM classifier. It does not converge as fast as the LinearSVC class, but it can be useful to handle huge datasets that do not fit in memory.

1.8.1.4 *K-Nearest Neighbors*

The k-Nearest Neighbors (kNN) is an instance-based learning algorithm that can be used for regression and classification problems. Let's focus on the classification case. The principle behind kNN is to find a k number of training samples closest in distance to the new point, and predict the label from these: a new data point is assigned the data class which has the most representatives within the nearest neighbors of the point [28]. Therefore kNN does not attempt to construct a real model. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular. Moreover, it performs best with a low number of features. The standard Euclidean distance is the most common choice as a metric to compute the distance, but any other metric can be chosen. The optimal choice of the value for the k parameter is highly data-dependent: in general, a larger value suppresses the effects of noise, but at the same time makes the classification boundaries less distinct.

1.8.1.5 *Decision Tree and Random Forest*

Decision Trees are powerful algorithms capable of fitting complex datasets, can perform both classification and regression tasks, and even multioutput tasks.

Decision Trees use a tree structure to represent a number of possible decision paths and an outcome for each path. Let's take a concrete example to better understand their

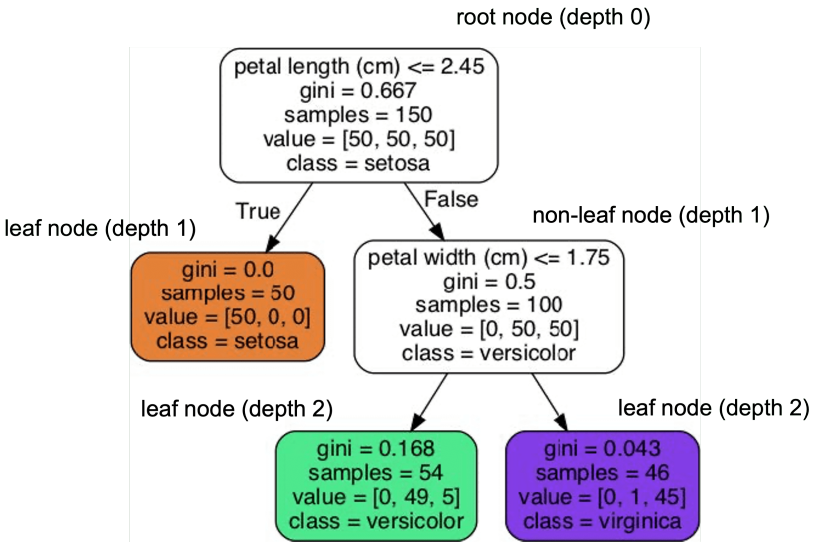


Fig. 1.16: How a Decision Tree makes predictions with the Iris Dataset.

logic and to see how a Decision Tree of Scikit-learn makes predictions in a classification problem, using the Iris Dataset provided by Scikit-learn [29] (see Fig. 1.16). This dataset consists of petal and sepal length and width values for three different types of irises flowers (setosa, versicolor, and virginica), stored in a 150x4 numpy.ndarray. Scikit-learn uses the Classification and Regression Tree (CART) algorithm which produces only binary trees, i.e. non-leaf nodes always have two children.

The starting point is at the *root node* on the top of Fig. 1.16 (depth 0). This node asks for each entry in the dataset whether the flower petal length is smaller than 2.45 cm. If it is, then you move down to the left child node (depth 1, left) which is a *leaf node*, which means that it does not have any child nodes. Such a node does not ask any questions, but combines the predicted class to all the flowers that fall in this node, in this case, an Iris setosa. Let's go back and suppose that the flower petal length is not smaller than 2.45 cm. In this case, you move down to the right child node which is a non-leaf node and it asks another question, i.e. is the petal width smaller than 1.75 cm? If it is, the flower is predicted to be an Iris versicolor (depth 2, on the left), and if not is predicted to be an Iris virginica (depth 2, on the right).

In Fig. 1.16 each node has a *samples* attribute, which counts how many training instances it applies to, a *value* attribute, which tells how many training instances of each class the node applies to, the *class* attribute, which tells the predicted class for the instances falling into the node, and the *gini* attribute, which measures the node impurity. A node is called "pure" (when gini=0) if all training instances belong to the same class. The Gini impurity is computed with Eq. 1.15.

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2 \tag{1.15}$$

where G_i is the gini score of the i^{th} node and $p_{i,k}$ is the ratio of instances of class k among the training instances in the i^{th} node. We can see that in the Fig. 1.16 the leaf nodes

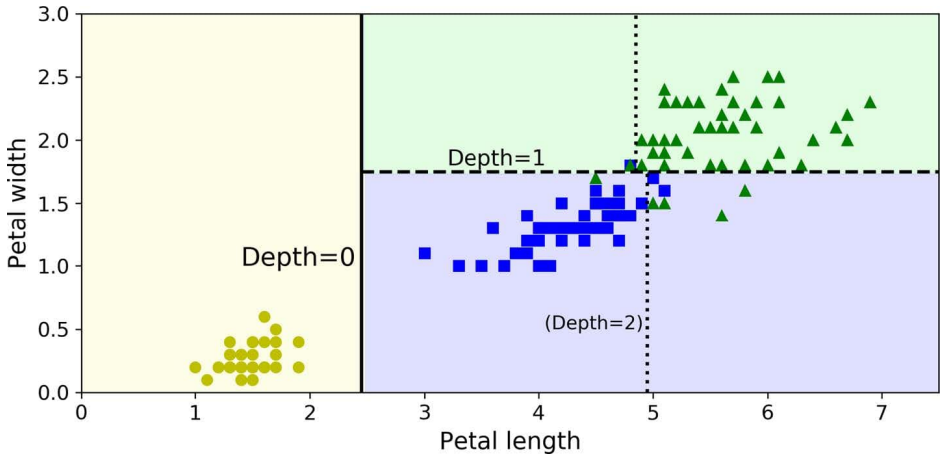


Fig. 1.17: Decision Boundaries of the Decision Tree discussed in the text with a non-leaf node of depth 2 [13].

of depth 2 are not pure, so two additional non-leaf nodes are necessary at least. See Fig. 1.17 for the decision boundaries given by a Decision tree with two non-leaf nodes of depth 2.

Moreover, a Decision Tree can estimate the probability that an instance belongs to a given class k: firstly it finds in the tree the leaf node for that instance, and then it returns the ratio of training instances of class k in this node.

But how does the CART algorithm training work? It starts by splitting the training set into two subsets using a single feature k and a threshold t_k. The values of k and t_k are chosen by searching the pair (k, t_k) that produces the purest subsets weighted by their size. The cost function that the algorithm tries to minimize is given by Eq. 1.16.

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \tag{1.16}$$

where $G_{left/right}$ measures the impurity of the left/right subset, whereas $m_{left/right}$ is the number of instances in the left/right subset. Once the CART algorithm has found the best pair and split the training set into two parts, then it splits the subsets using the same logic, and so on, stopping once it reaches the maximum depth (or reaches the value set for other hyperparameters). At the same time, these hyperparameters are used to regularize the model which allows avoiding overfitting. Other algorithms begin with training the Decision Tree without restrictions, and then pruning (deleting) unnecessary nodes. A node which children are all leaf nodes is defined as unnecessary if the purity improvement it gives is not statistically significant.

The computational complexity of a CART is $\mathcal{O}(n \times m\, log_2(m))$ where n is the number of features whereas m is the training set size. Instead of using the Gini impurity, the entropy impurity can be selected.

$$H_i = -\sum_{k=1}^{n} p_{i,k} log_2\left(p_{i,k}\right) \qquad\qquad p_{i,k} \neq 0 \tag{1.17}$$

Most of the time using one impurity measure instead of the other does not make a big difference, and they lead to similar trees. In such cases, the Gini impurity can be preferred

since it is slightly faster to compute. But when they differ, entropy tends to produce slightly more balanced trees.

Decision Trees are intuitive, and their decisions are easy to interpret: for this reason they are often called *white box models*, in contrast to Random Forests or NN which are considered *black box models*. Decision Trees make good predictions and you can easily check the calculations that they performed to make the predictions. Anyway, they have a few limitations: they produce orthogonal decision boundaries making them sensitive to training set rotation. Then, Decision Trees are very sensitive to small variations in the training data. Random Forests can limit this instability by averaging predictions over many trees, as described in the following.

Ensemble Learning and Random Forest

Ensemble Learning is a technique where the predictions of a group of predictors (called ensemble), such as classifiers or regressors, are aggregated. An Ensemble Learning algorithm is called Ensemble method and it often gives better predictions than the best individual predictor.

Let's suppose to have trained a few different classifiers (e.g Logistic Regression, SVM, kNN, and other classifiers). One simple way to create a better classifier is to aggregate the predictions of each classifier, and predict the class that gets the most votes. This majority-vote classifier is called a *hard voting* classifier, and often it achieves higher accuracy than the best classifier in the ensemble. Indeed, despite that each classifier is a *weak learner* (i.e. that it can do only slightly better than random guessing), the ensemble can be a *strong learner* (which achieves high accuracy) if the weak learners are in sufficient number and sufficiently diverse. Moreover, if all classifiers are able to estimate class probabilities, then we can tell the voting classifier to predict the class with the highest class probability, averaged over all the individual classifiers. This is called *soft voting* and it often achieves higher performance than hard voting because it gives more weight to highly confident votes.

Another approach to get a diverse set of classifiers instead of using very different training algorithms is to use the same training algorithm for every predictor and train them on different random subsets of the training set. This method is called *bagging* (short for bootstrap aggregating) when the sampling is done with replacement, whereas is called *pasting* when the sampling is done without replacement. In both cases, the training instances are sampled several times across multiple predictors, but only with bagging the training instances are sampled several times for the same predictor. After that all predictors are trained, the ensemble makes a prediction for a new instance by aggregating the predictions of all predictors, which typically means to take the statistical mode in case of classification (i.e. the most frequent prediction, as in the hard voting classifier) or the average in case of regression. It turns out that individually each predictor has a higher bias than if it was trained on the original training set, but the ensemble has a similar bias and a lower variance than a single predictor trained on the original training set (see Sec.

1.8.3 for the description of bias vs variance in ML models). Another good feature is that the predictors can all be trained (and can make predictions) in parallel, scaling very well.

With the bagging method, some instances may be sampled multiple times for a given predictor, while others may not be sampled at all (called out-of-bag instances). Since a predictor never sees the out-of-bag instances during the training, it can be evaluated on these instances. Furthermore, each predictor can be trained on a random subset of the input features, and this is particularly useful when you are dealing with high-dimensional inputs (such as images). The sampling of both training instances and features is called *Random Patches* method, whereas keeping all the training instances but sampling the features is called *Random Subspaces* method. Sampling features give even more predictor diversity, resulting in a bit more bias but with a lower variance.

Random Forest is an ensemble of Decision Trees, generally trained via the bagging method (or sometimes pasting). This algorithm introduces extra randomness when growing trees: instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This gives a greater tree diversity, which results in a higher bias but a lower variance, generally yielding an overall better model. Furthermore, it is possible to make trees even more random by adding random thresholds for each feature rather than searching for the best possible thresholds (as Decision Trees do). A forest of such trees is called an Extremely Randomized Trees ensemble (or *Extra-Trees* for short). This technique leads to more bias but a lower variance, and it makes Extra-Trees much faster to train than regular Random Forests, as finding the best possible threshold for each feature in every node is a time-consuming task. Another great quality of Random Forests is that they make it easy to measure the relative importance of each feature by looking at how much the tree nodes use that feature to reduce impurity on average (across all trees in the forest).

Boosting refers to any Ensemble method that combines several weak learners into a strong learner. The most of boosting methods trains predictors sequentially, each of them trying to correct its predecessor. Among the many boosting methods available, the most popular are AdaBoost (short for Adaptive Boosting) and Gradient Boosting.

In the technique adopted by **Adaboost**, a new predictor pays more attention to the training instances that the predecessor underfitted, i.e. the hard cases. When training an AdaBoost classifier, firstly the algorithm trains a base classifier (e.g. a Decision Tree) and uses it to make predictions on the training set, then it increases the relative weight of misclassified training instances. Afterwards it trains a second classifier, it updates the instance weights, and so on. As a drawback this sequential learning technique cannot be parallelized and it does not scale (or only partially), since each predictor can only be trained after that the previous predictor has been trained and then evaluated.

Another popular boosting algorithm is **Gradient Boosting**, where instead of adjusting the instance weights at each iteration as AdaBoost does, it tries to fit the new predictor to residual errors made by the previous predictor. The Python library XGBoost gives a very popular and optimized implementation of Gradient Boosting. It aims to be extremely fast, scalable, and portable and it is often present in solutions that win ML competitions. It

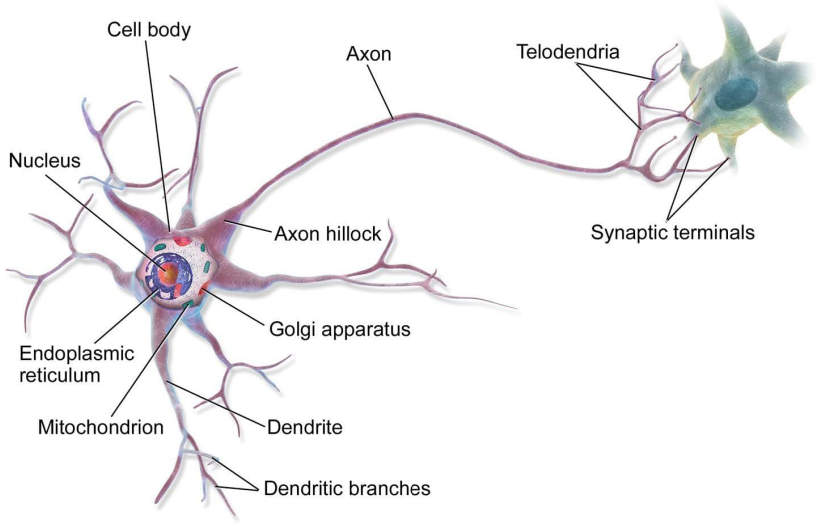


Fig. 1.18: Representation of a biological neuron [13].

considers the potential loss for all possible splits to create a new branch (especially useful in case of thousands of features, therefore thousands of possible splits), and it looks at the distribution of features across all data points in a leaf, using this information to reduce the search space for possible feature splits. It allows many hyperparameter settings to be investigated quickly and it is designed to limit overfitting.

An additional Ensemble method we report is called *stacking* (short for stacked generalization), that instead of using trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble, trains a model to perform this aggregation.

1.8.1.6 *Neural Network*

An ANN is a ML algorithm inspired by networks of biological neurons found in our brains. A biological neuron is a cell just like any other cell of our body, and it comprises three major parts: the cell body (also called soma), the dendrites, and the axon (see Fig. 1.18). The cell body is where the nucleus lies, where the DNA of the neuron is housed. The dendrites are like fibers branched in different directions and are connected to many cells. The axon is a very long extension which length may be just a few times longer than the cell body, or up to tens of thousands of times longer. At the end of the axon, there are synapses that allow contact with the dendrites. Neurons produce short electrical impulses that travel along axons and cause synapses to release chemical signals called neurotransmitters. When a neuron receives enough of these neurotransmitters within a few milliseconds, it emits its own electrical impulses. The axon transmits the signal to the dendrites of other neurons, whereas dendrites receive the signals from the axons of the surrounding neurons.

If the individual biological neurons are taken singularly, they seem to behave in a simple way, but since they are organized in a network of billions of neurons, where each neuron is typically connected to thousands of other neurons, this structure allows them to perform highly complex computations.

In 1943, McCulloch and Pitts first proposed a very simple model of the biological neuron, known as artificial neuron. But it was in 1957 that the first ANN was invented (the Perceptron), based on a slightly different artificial neuron called Threshold Logic Unit (TLU), or sometimes Linear Threshold Unit (LTU). The inputs and output of the neuron are numbers, and each input connection is associated with a weight. The TLU computes the weighted sum of its inputs, then applies a step function to this sum, and outputs the result, i.e.:

$$h_w(\boldsymbol{x}) = step(z) \qquad z = \boldsymbol{x}^\intercal \boldsymbol{w} \tag{1.18}$$

Two simple step functions that are commonly used are the Heaviside and the sign function. Assuming the threshold=0, they are defined as

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases} \tag{1.19}$$

Thanks to the step function, a single TLU can be used for simple binary classification. A Perceptron is composed of a single layer of TLUs where each TLU is connected to all inputs. A *fully connected layer* (or *dense layer*) is a layer of neurons, where each of them is connected to every neuron in the previous layer, i.e. its input neurons. The input neurons form the input layer and simply output whatever input they are fed. An extra bias feature is generally added ($x_0 = 1$), which typically is represented by a neuron (called bias neuron) that outputs 1 all the time. A Perceptron makes it possible to efficiently compute the outputs of a layer of artificial neurons for several instances at once according to the following equation:

$$h_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{X}) = \phi(\boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}) \tag{1.20}$$

where ϕ is the activation function, $\boldsymbol{X}$ is the matrix of input features (one row per instance and one column per feature), $\boldsymbol{W}$ contains the weights except for the bias (one row per input neuron and one column per artificial neuron in the layer), and $\boldsymbol{b}$ is the vector with the weights connected to the bias neuron (one bias term per artificial neuron).

Perceptrons are trained taking into account the error made by the network when it makes a prediction, by reinforcing those connections that help to reduce the error. The Perceptron receives one training instance at a time and makes its predictions for each instance. Then for each output neuron that produced a wrong prediction, it reinforces the weights of the inputs that would have contributed to the correct prediction. The rule followed is

$$w_{i,j}{}^{(\text{next step})} = w_{i,j} + \eta \left(y_j - \hat{y}_j\right) x_i \tag{1.21}$$

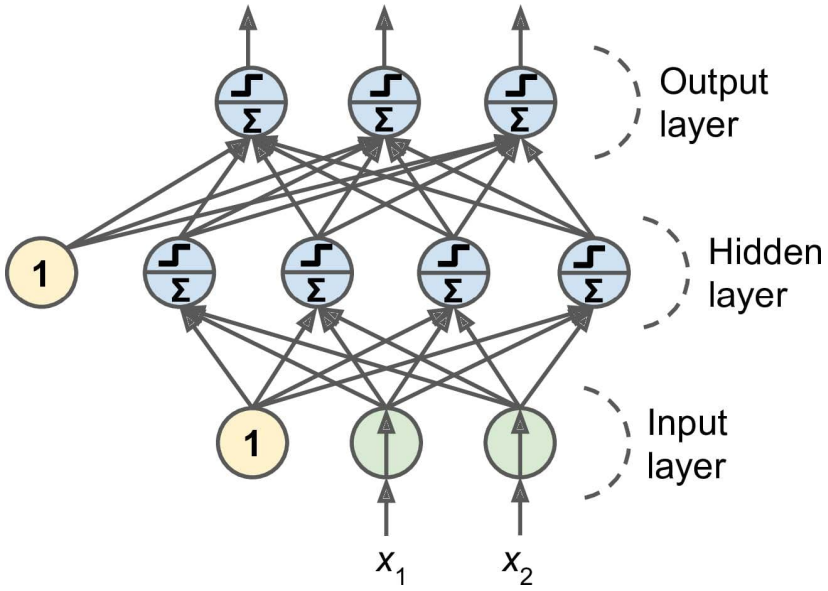


Fig. 1.19: Architecture of a MLP with two inputs, one hidden layer with four neurons, and three output layers [13].

where $w_{i,j}$ is the weight between the i^{th} input neuron and the j^{th} output neuron, η is the learning rate, y_j is the target output of the j^{th} output neuron for the current training instance, $\hat{y}_j$ is the output of the y^{th} output neuron for the current training instance, and x_i is the i^{th} input value of the current training instance. The decision boundary of each output neuron is linear, so actually, Perceptrons are not capable to learn complex patterns, but if the training instances are linearly separable, the Perceptrons converge to a solution. Contrary to Logistic Regression classifiers, Perceptrons do not output a class probability since they make predictions based on a threshold. This is one of the reasons to prefer Logistic Regression over Perceptrons.

Some of the limitations of the Perceptrons can be removed by stacking multiple Perceptrons. The resulting ANN is called MultiLayer Perceptron (MLP), composed of one input layer, one or more layers of TLUs (called *hidden layers*), and one final layer of TLUs (the *output layer*). See Fig. 1.19 for an example of MLP. Each layer except the output one includes a bias neuron, and it is fully connected to the next layer. As the signal flows only in one direction (from the inputs to the outputs), this architecture is an example of a Feed-forward NN (FNN). When an ANN contains many hidden layers, it is called Deep NN (DNN), and the part of ML that uses such ANN is called deep learning.

A MLP is trained using the *backpropagation* algorithm. It handles one mini-batch at a time, and it goes through the full training set multiple times (each pass through the full training set is called epoch). At the beginning, the weights of all the hidden layers are initialized randomly, or otherwise, the training would fail. Each mini-batch is passed to the network input layer and then to the first hidden layer. Here the algorithm computes the output of all the neurons for each instance of the mini-batch, then the result is passed to the next layer where the output is computed and passed to the next layer, and so on until we get the output by the output layer. This is the forward pass where all intermediate results are stored since they are needed for the backward pass. At this point the algorithm measures

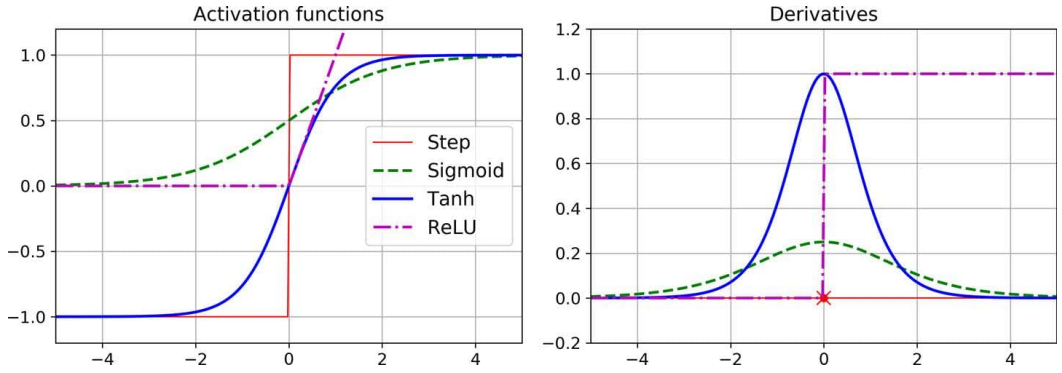


Fig. 1.20: Comparison of activation functions (step, sigmoid, tanh and ReLU) and their derivatives [13].

the output error of the network (using a loss function), and then by applying the chain rule[3] it computes how much each output connection contributed to the error. Then, the algorithm measures the error contribution coming from each connection in the previous hidden layer, working backward. This procedure is repeated for each hidden layer until it reaches the input layer. In this reverse pass performed for every training instance in the mini-batch, the algorithm measures the error gradient across all the weights of the network by propagating the error gradient backward through the network. Lastly, the algorithm performs a Gradient Descent step for the entire batch to adjust all the connection weights in the network, using the computed error gradients. The whole process is repeated for each batch and each epoch.

In order to make this algorithm properly work, the step function as activation function should be replaced. Indeed it contains only flat segments and therefore it is not possible to calculate the gradient. Other activation functions are commonly used: the logistic/sigmoid function (see Eq. 1.13), the hyperbolic tangent function ($tanh(z) = 2\sigma(2z) - 1$) and the Rectified Linear Unit function ($ReLU(z) = max(0, z)$). The first two have a well-defined nonzero derivative everywhere, enabling the Gradient Descent to make some progress at every step. The ReLU function is continuous but not differentiable at $z = 0$ (the slope changes abruptly, so the Gradient Descent can bounce around). However, it works very well, it is fast to compute, and the fact that it does not have a maximum output value helps to reduce some issues during Gradient Descent. For these reasons, it has become the default activation function. See Fig. 1.20 for a comparison of such activation functions and their derivatives.

Activation functions are important for ANNs since they add some nonlinearity between layers (without them it is like having a single layer and it is impossible to solve complex

3 The chain rule allows finding the derivative of a composite function [30]. Considering the composite function $h = g(f(x))$, its derivative is given by the chain rule: $\frac{dh}{dx} = \frac{dh}{du} \cdot \frac{du}{dx}$. Here u is the output of the inner function f (hence $u = f(x)$) which is then fed as input to the next function g to give h (hence $h = g(u)$). Therefore, the chain rule relates the net output h to the input x through an intermediate variable u. Now, to generalize the chain rule to higher dimensions let's consider the case where $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{u} \in \mathbb{R}^n$, that is the inner function f maps m inputs to n outputs, while the outer function g receives n inputs to produce an output h. For $i = 1, ..., m$ and for $j = 1, ..., n$ the chain rule becomes: $\frac{\partial h}{\partial x_i} = \sum_j \frac{\partial h}{\partial u_j} \cdot \frac{\partial u_j}{\partial x_i}$.

problems), which theoretically allows to sufficiently large DNN to approximate any continuous function.

MLPs can be used for both regression and classification tasks. In regression tasks, when a single value should be predicted (e.g. the price of a house) just a single output neuron is needed, whereas for multivariate regression one output neuron per output dimension is needed. Moreover generally you do not want to use any activation function for the output neurons so that they are free to output any range of values. The loss function typically used during training is the MSE.

For a binary classification problem, only a single output neuron is needed where usually a logistic activation function is used: in this way the output will be a number between 0 and 1, which can be interpreted as the estimated probability of the positive class. MLPs can be also used for multiclass classification tasks. Generally, if each instance can belong only to a single class, one output neuron for each positive class is dedicated and the softmax activation function [31] is used at the end of the whole output layer. Regarding the loss function, the log-loss (also called cross-entropy loss) is generally used as default.

But how many layers and how many neurons per layer does a NN need to give good results? For many problems, using a single hidden layer can give reasonable results: indeed a MLP with just one hidden layer (but with enough neurons) can theoretically model even the most complex functions. But for complex problems, DNN can model complex functions using far fewer neurons than shallow nets, enabling them to perform much better with the same amount of training data. The number of neurons in the input and in output layers is determined by the type of input and output the given task requires. Regarding the neurons per hidden layer, one approach is to size layers to form a pyramid, with fewer and fewer neurons in each layer moving towards the output layer. However, it seems that using the same number of neurons in all hidden layers performs just as well in most cases (or even better), and as an advantage, there is only one hyperparameter to tune (instead of one per layer). Moreover, depending on the dataset, sometimes it can help to have the first hidden layer bigger than the others, and in general better performance is reached by increasing the number of layers instead of the number of neurons per layer. Typically, a simple and efficient approach is to pick a model with more layers and neurons than you actually need, then use other techniques (e.g. early stopping and regularization) to prevent it from overfitting.

There are many other hyperparameters of a MLP that can be chosen and adjusted. The most important ones are: learning rate, optimizer, activation function, and number of iterations.

- **Learning rate**. Generally, the optimal value for the learning rate is about half of the maximum learning rate (i.e. the value above which the training algorithm diverges).
- **Optimizer**. It is a good practice to try and evaluate other optimizers than the Mini-batch Gradient Descent. The most popular algorithms are: momentum optimization, Nesterov Accelerated Gradient, AdaGrad, RMSProp, Adam and Nadam optimization.

- **Batch size**. Often large values of batch size lead to training instabilities (especially at the beginning of training) and the produced model may not generalize as well as a model trained with small batch size. Anyway, a good strategy can be to try to use a large batch size, using learning rate warmup (i.e. starting the training with a small learning rate, then increasing it), and if training is not stable or the final performance is not good, try to use a small batch size instead.
- **Activation function**. We have already discussed activation functions earlier in this Section, providing some examples for different tasks.
- **Number of iterations** (also called epochs). A good practice is to use a large value and then use the early stopping (for more details, see Sec. 1.8.3).

To avoid overfitting in training ANNs, it is a common practice to implement l_1 or l_2 regularization and the dropout. The latter is the most popular regularization technique for DNN where at each training step, each neuron (including the input neurons, but excluding the output neurons) has a given probability of being temporarily "dropped out", i.e. it will be completely ignored during that training step but it may be active during the next step. In case of overfitting, a way to remove it is to increase the dropout rate. Conversely, the dropout rate should be decreased if the model underfits the training set.

There are many other types of ANN, with an increasing complexity, which description goes beyond the scope of this thesis. In conclusion of this Section where we explored the main ML algorithms for supervised learning tasks, we show in Fig. 1.21 a comparison of decision boundaries produced by different ML algorithms of the Scikit-learn library on three different sets of data.

1.8.2 *Performance metrics*

To evaluate the goodness of a model and its performance, a proper set of metrics must be chosen. Classification and regression are the two types of supervised learning which have their respective set of metrics.

Regression models have continuous output, so the metrics should be based on calculating some sort of distance between the predicted value and the true one. Since the work of this thesis described in Ch. 4 focuses on a classification problem, the most common regression metrics are only listed, and are the following:

- Mean Absolute Error (MAE),
- MSE (see Sec. 1.8.1.1),
- RMSE (see Sec. 1.8.1.1),
- Coefficient of determination R^2 (R squared).

In binary classification problems, there are two classes of data, and each of them is assigned a label, i.e. 1 and 0, to distinguish the two classes also called positive and negative.

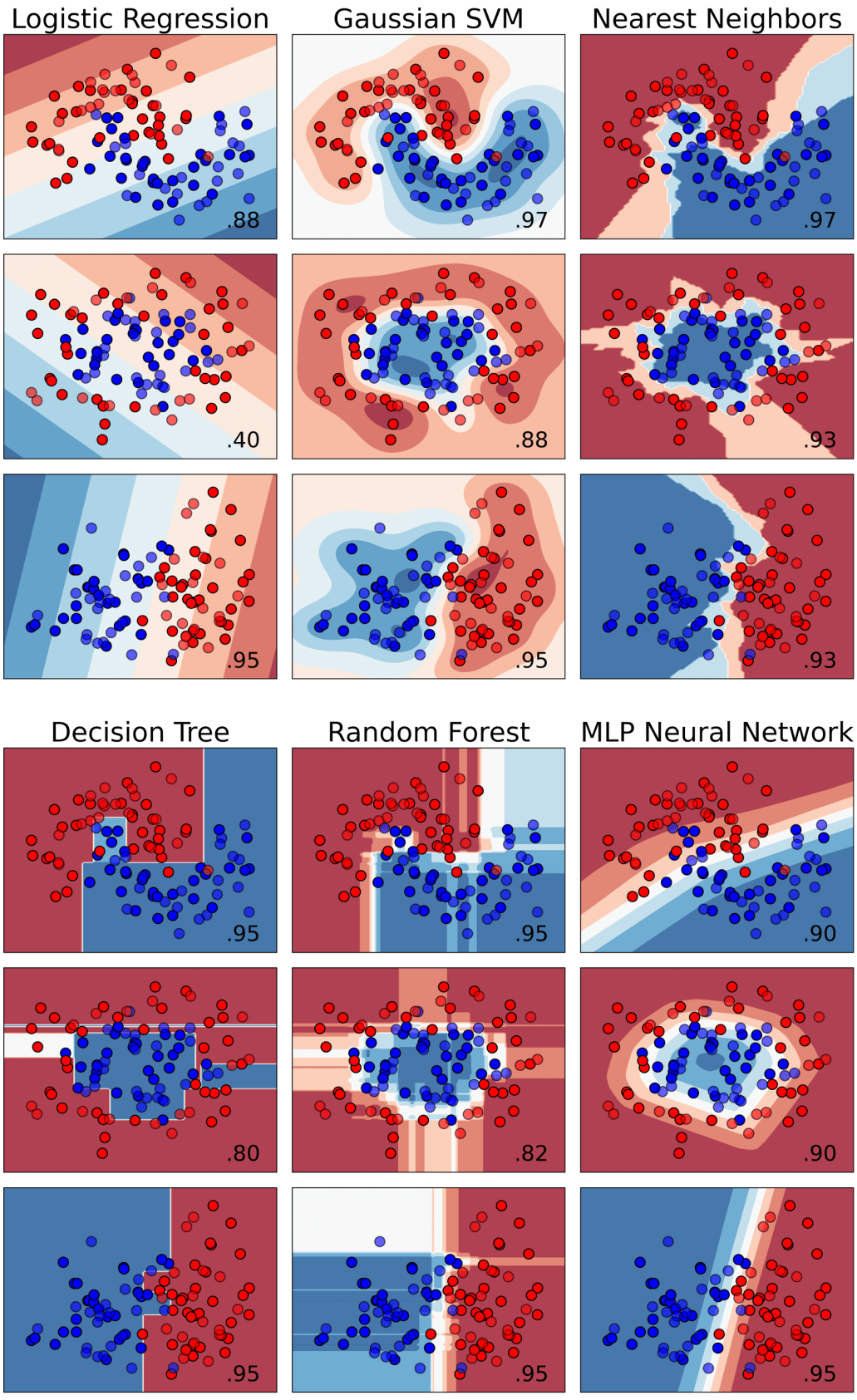


Fig. 1.21: Decision Boundaries using different ML algorithms from the Scikit-learn library, for three different sets of data. The plots show training points in solid colors and testing points semi-transparent. On the bottom right corner of each plot, the classification Accuracy of the model on the test set is reported (see Sec. 1.8.2 for the classification Accuracy definition).

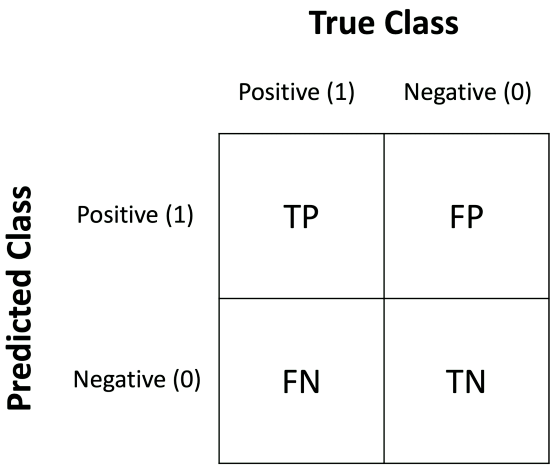


Fig. 1.22: Confusion matrix.

All measures of performance in binary classification problems are based on four variables obtained from the predictions of the ML model on a set of labeled data: TP (True Positive), TN (True Negative), FP (False Positive), FN (False Negative). TP refers to how many positive class samples the model predicted correctly. TN refers to how many negative class samples the model predicted correctly. FP refers to how many negative class samples the model predicted incorrectly, and it represents the Type-I error in statistical nomenclature. FN refers to how many positive class samples the model predicted incorrectly, and it represents the Type-II error in statistical nomenclature.

Confusion Matrix is a tabular visualization of the ground-truth labels versus model predictions. Each row of the confusion matrix represents the instances in a predicted class and each column represents the instances in an actual/true class. In each cell of the confusion matrix, the values we previously defined are reported, i.e. TP, FP, FN, and TN (see Fig 1.22).

To calculate the evaluation metrics of a multiclass classification problem, the problem should be firstly broken into multiple binary classification problems, also called the One-vs-Rest (OVR) strategy. In OVR, the true and predicted classes are re-calculated for each class, combining all other classes in a single class producing essentially a binary classification output. This is performed for each class in the multiclass classification, converting the multiclass output into multiple binary classification outputs equal to the number of classes.

The most common classification metrics are: Accuracy, Precision, Recall, F_1 score, Area Under the Receiver Operating Characteristic (ROC) Curve (AUC).

1.8.2.1 *Accuracy*

Classification Accuracy is the most common performance metric for classification problems. It is defined as the number of correct predictions divided by the total number of predictions, i.e.:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.22}$$

In most cases, Accuracy is just fine and it is sufficient to evaluate a model. Nevertheless, Accuracy is not a useful metric when dealing with the so-called "skewed classes". Let's consider the problem of cancer classification, where we have features of medical patients and we want to decide whether they have cancer or not. Let's suppose to train a logistic regression model that has a 1% of error on a test set. This means a 99% of correct diagnosis (and also 99% of Accuracy), so a very good model. But let's suppose that only 0.5% of patients really have cancer. Our model gives a worst Accuracy result compared to another model that simply predicts for all the patients do not to have cancer, which obtains an Accuracy of 99.5%. So when a non-learning algorithm (i.e. that always predicts 0 or 1) is better than a simple learning algorithm, it means we are in the peculiar case of skewed classes, and in such cases, an Accuracy metric that increases is not always an indicator of improvements in the model. For this reason, other metrics are necessary to have better information about a model.

1.8.2.2 *Precision*

The Precision metric is defined as the ratio of correctly predicted positive samples to the total positive samples, i.e.:

$$Precision = \frac{TP}{TP + FP} \tag{1.23}$$

Intuitively, precision is the ability of the model not to label as positive a sample that is negative, and its value should be as high as possible. Taking the previous example, precision would answer the question: "among all patients predicted to have cancer, how many actually have it?". A model that predicts the label 0 to all the samples, gives $TP = 0$ and Precision=0.

1.8.2.3 *Recall*

The Recall metric, also called Sensitivity or True Positive Rate (TPR), is defined as the ratio of correctly predicted positive samples to all samples in the positive class, i.e.:

$$Recall = \frac{TP}{TP + FN} \tag{1.24}$$

Intuitively, recall is the ability of the classifier to find all the positive samples, and its value should be as high as possible. Taking the previous example, precision would answer the question: "among all patients that actually have cancer, how many did the model predict to have it?". A model that predicts the label 0 to all the observations, gives $TP = 0$ and Recall=0.

1.8.2.4 *F1*

The F_1 score is the harmonic mean of Precision and Recall, i.e.:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{1.25}$$

Whereas the regular mean treats equally all values, the harmonic mean gives much more weight to low values. In this way, the model will only get a high F_1 score if both recall and precision are high. The F_1 score favors models that have similar precision and recall, but this is not always what we want: in some contexts, we want to mostly care about precision, and in other contexts, we want to mostly care about recall. But we cannot have both with a high value: increasing the precision reduces the recall and vice versa. This is called the precision/recall trade-off.

To understand this trade-off it is necessary to define the concept of *decision threshold*. It is a cut off on the probability predicted by the ML model for an observation to be classified, for example, as either 0 or 1. In general, for binary classification problems, the threshold value is set by default at 0.5, meaning that all the values equal to or higher than the threshold are mapped to 1 and all the others are mapped to 0. But for some classification problems, a such default threshold value is not optimal and the result is a poor model performance. So we can vary the value of the threshold to have a higher precision or a higher recall. This decision threshold can be applied not necessarily to the aforementioned probability, but also to other scores produced by the model for each sample. For example, Scikit-learn does not let directly set the threshold on the probability, but it gives access to the decision score that it uses to make predictions so that you can adjust the threshold on this score.

1.8.2.5 *Area Under the ROC Curve*

A ROC curve is a graph showing the performance of a classification model at all classification thresholds (see Fig. 1.23). This curve plots two parameters, the TPR (i.e. the Recall metric, see Eq. 1.24) and the False Positive Rate (FPR) defined as:

$$FPR = \frac{FP}{FP + TN} \tag{1.26}$$

AUC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). AUC ranges in value from 0 to 1. A model that gives 100% wrong predictions has an AUC of 0, a model that gives 100% correct predictions has an AUC of 1, whereas a model that makes predictions randomly has an AUC of 0.5. In the last case, the ROC curve is the diagonal line connecting the points (0,0) and (1,1). Therefore, the larger the area under the ROC curve, the better the model.

AUC is classification-threshold-invariant, as it measures the quality of the model predictions regardless of the classification threshold chosen [33]. This feature is not always desirable, like when there are wide disparities in the cost of FN vs FP and we want to minimize one type of classification error.

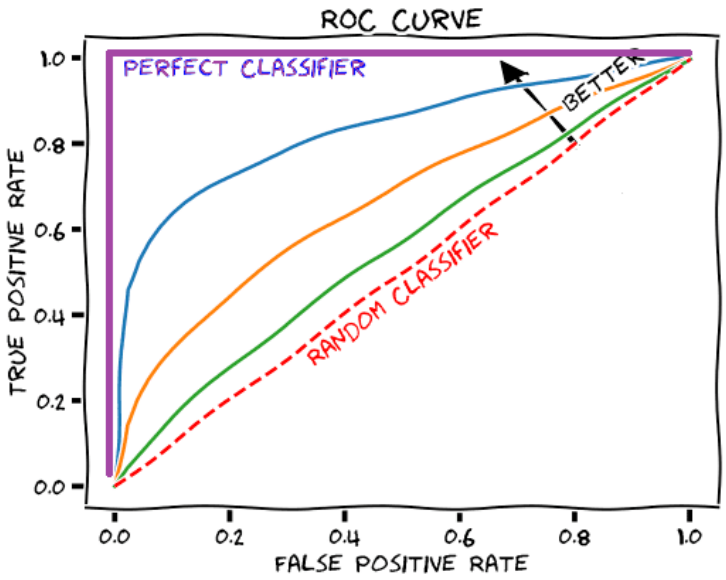


Fig. 1.23: ROC curve [32].

1.8.3 *Training and evaluation of a ML model*

To evaluate how well a model generalizes, we have to try it out on new samples. One way to do it is to use the model in production and monitor how well it performs. This is not a good idea if the model is really bad, and putting it into production is not always an easy and fast task. A more common solution is to divide the input data into three parts called training, validation, and test set.

The training set is used to train the ML model, while the test set is used to test it. In particular, the test set is used to get an estimate of the generalization error that a model will make on new instances before the model is launched in production. The estimate of the generalization error is computed with the loss function in the test set.

The validation set is used to evaluate (e.g. using the metrics reported in Sec. 1.8.2) and compare several candidates of models trained on the training set (where different algorithms and hyperparameters are used). Then, the best model is selected and trained on the set composed of training and validation sets, producing the final model. The latter is then evaluated on the test set to get an estimate of the generalization error.

A proper balance in the dimension of the three sets should be found. For example, if the validation set is too small, then model evaluations will be imprecise, while if the validation set is too large, then the trained models risk not having learned enough from the data. The most common size solutions adopted, calculated in relation to the input data size, are: 80%-10%-10%, 70%-15%-15%, and 60%-20%-20% for training, validation, and test set respectively.

Another solution that can be adopted is the *k-fold cross-validation* which uses many small validation sets. The set composed of training and validation sets is randomly divided into k equal-sized subsets called "folds". Then each model is evaluated once per fold after it is trained on the rest of the data, i.e. using the k-1 folds. This process is repeated k times,

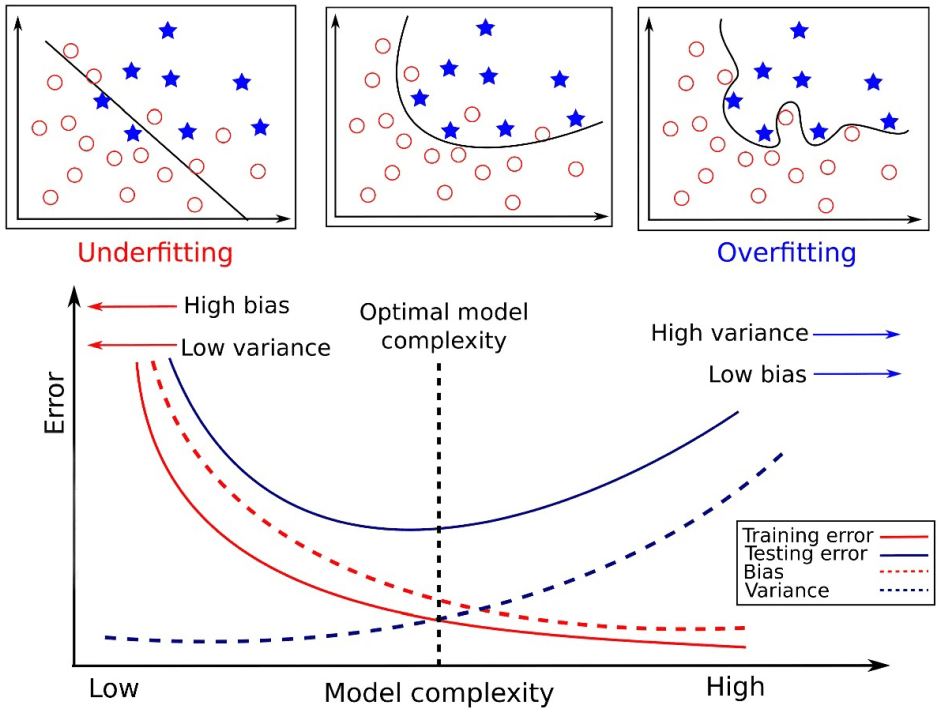


Fig. 1.24: Example of decomposition of the generalization error in bias, variance, and irreducible error [34].

each time varying the fold used for the validation and using the remaining ones for the training. By averaging out all the evaluations of the model, a much more accurate measure of its performance can be obtained, but at the same time, it takes longer to complete the whole procedure (there are k training processes).

The k-fold cross-validation procedure gives an estimate of the generalization performance of the model. The training set can be also used to have an estimate of the generalization error but this will not be a good estimate since the training set contains only samples that the model has already seen during the training and no new samples. If a model has good performance on training data but does not generalize according to the cross-validation metrics, then the model is overfitting. If the model performs poorly on both, then it is underfitting. This is one way to tell whether a model is too simple or too complex.

An important result of statistics is the model generalization error that can be expressed as the sum of three errors: bias (due to wrong assumptions), variance (due to the model excessive sensitivity to small variations in training data), and irreducible error (due to the noisiness of the data itself). Reducing the model complexity increases the bias and reduces the variance, whereas increasing the model complexity typically increases the variance and reduces the bias. To have a good model a data scientist needs to find a good bias/variance trade-off (see Fig. 1.24).

Another way to see if the model is too simple or too complex and if it is suffering from high bias (underfitting) or high variance (overfitting), is to use the *learning curves* which are plots of model performance on the training and validation sets as a function of the training set size, or the training iteration. In Fig. 1.25, the two plots give a simplified overview of

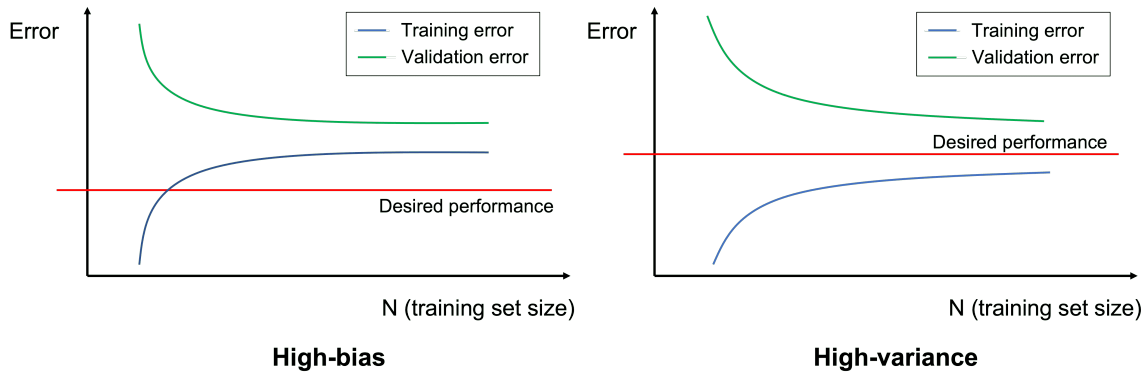


Fig. 1.25: Examples of learning curves. The learning diagram on the left shows a situation of high bias and underfitting: the training and validation error curves are very close and higher than the desired error. The learning diagram on the right shows a situation of high variance and overfitting: there is a large gap between training and validation error curves where the former is lower than the desired error.

the learning curves behavior (actually in real life they are more noisy). In the case of high bias, the training and validation error curves are very close and higher than the desired error. In this case, getting more data will not help much the model and the two errors will remain high, with the validation error higher than the training error but closer and closer. Whereas in the case of high variance, the training and validation error curves are divided by a large gap that can be reduced by getting more data. Nevertheless, the validation error curve will remain higher than the training error curve and the difference remains significant.

A way used to avoid overfitting in iterative learning algorithms, like Gradient Descent, is to stop the training as soon as the validation error reaches the minimum. This technique is called *early stopping*. With Stochastic and Mini-batch Gradient Descent, the curves are not so smooth and it could be difficult to know whether the minimum is being reached or not. One solution is to stop only after the validation error is above the minimum for some time and you are confident that the model will not get better. At this point, we can use the model parameters related to the point where the validation error was at the minimum.

2

CLOUD COMPUTING

To understand why the cloud metaphor is used to denote the technological revolution we are now seeing, we have to look at what a network actually is from an engineering perspective [35]. When multiple computers are connected in a network, this happens through multiple nodes. These are computers or other electronic devices that have the capability to connect to other devices or computers and route communication traffic between computers. Every node in the network mainly receives data from one endpoint and routes them to another. Rather than try to summarize all the elements that constitute the network, a metaphor was needed to symbolize an amorphous collection of machines. For this reason, the concept of cloud was introduced, which is also simple to draw and visualize. Drawing a cloud was a convenient and fast way to illustrate devices connected to a network.

The other side of the coin is that the term cloud was born from a marketing perspective and used as a general term to describe very different solutions that all had in common the use of Internet. In this perspective cloud computing can be identified today as the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing [36]. Instead of buying, owning, and maintaining physical data centers and servers, the user can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider.

In this chapter, we go into detail about cloud computing, giving an overview of the history of the term and describing its different types and uses. Particular attention is given to the primary technology components cloud computing relies on that go beyond the cloud computing itself which, however, in turn has contributed to their development. In the last part of the chapter, we focus on two aspects: the rise of cloud native applications based on microservices (Sec. 2.7) and the thematic of security in cloud computing (Sec. 2.8). The latter involves several aspects, and considering the content of the original contribution of this thesis described in Ch. 4, greater attention is given to the authentication and authorization phase of users and applications, that allows access to cloud services to be appropriately filtered.

2.1 A BRIEF HISTORY OF CLOUD COMPUTING

The origin of the idea related to cloud computing can be traced back to around the 1950s, when the computer scientist John McCarthy came up with the theory of “time-sharing”. At that time, computers were both huge and hugely expensive, so not every company could

afford to have one. According to the idea of time"time-sharing" users could rent the right to use the computational power of the computer, and share the cost of running it.

In the 1960s, J.C.R Licklider developed the ARPANET (Advanced Research Projects Agency Network): his vision was for a global computer network such that users could access programs and data from anywhere. Around this time, John McCarthy also stated that "computation may someday be organized as a public utility", like gas, electricity, and water.

> *If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility. [...] The computer utility could become the basis of a new and important industry.*
>
> John McCarthy, MIT's centennial celebration in 1961

In the 1970s, the creation of "virtual machines" (VMs) took the time-sharing model to a new level, allowing multiple computing environments to be hosted in one physical environment.

The ARPANET was formally decommissioned in 1990, after partnerships with the telecommunication and computer industry had assured expansion in the private sector and future commercialization of an expanded world-wide network, known as the Internet. In 1989 the World Wide Web was born at CERN and in January 1991 the first web servers outside CERN were switched on.

The first public mention of the cloud was in 1994 in an article in the Wired magazine about a company called General Magic. The company was founded by the creators of the Macintosh computer, Bill Atkinson and Andy Herzfeld. Their idea was that "a lot of different areas are converging on an electronic box that is in your pocket, that is with you all the time, that supports you in some way". Obviously at that time no "electronic box" would be able to do much computing that would be very interesting, so that had to be done on a server to which the device connected through a network. The product envisioned by General Magic was called Telescript and was a system to tie together all sorts of networks into one standardized interface that people would use to connect.

> *Now, instead of just having a device to program, we now have the entire Cloud out there, where a single program can go and travel to many different sources of information and create sort of a virtual service.*
>
> Bill and Andy's Excellent Adventure II, Wired 1994

In this quote there is not only the word "cloud" but also elements about the concept of cloud: the ability to access information and functionality from any device at any location. The particular technology offered by General Magic, however, did not catch on and the company was liquidated in the early 2000s.

In 1996, when the Internet and browsers were catching on, the Compaq company took the concept and word "cloud" to build a strategy on it. Marketing executive Steve Favoloro and technologist Sean O'Sullivan envisioned that file storage and business software

would move to the Internet and they were investigating on how their company would benefit from this. This was the start of the multibillion-dollar business selling servers to Internet providers. The term cloud is found in one of the internal document titled "Internet Solutions Division Strategy for Cloud Computing".

In 1997, another definition of cloud computing came from Professor Ramnath Chellapa of Emory University. He referred to cloud computing as the "computing paradigm where the boundaries of computing will be determined by economic rationale rather than technical limits alone".

In 1999, Salesforce became the first company to offer applications over the Internet. Three years later, the industry grew massively with video, music and other media being hosted and delivered online. In 2002, Amazon launched the Amazon Web Services (AWS) platform, a suite of enterprise-oriented services that provide remotely provisioned storage, computing resources, and business functionality [37]. Finally in 2006 the term "cloud computing" emerged in the commercial area. In this year Amazon launched its Elastic Compute Cloud (EC2) services that enabled organizations to "lease" computing capacity and processing power to run their enterprise applications. Google Apps also began providing browser-based enterprise applications in the same year, and three years later, the Google App Engine became another historic milestone. In 2007 a small start up called Netflix launched its video streaming website. Subsequently IBM launched SmartCloud, Apple launched iCloud, Microsoft launched Microsoft Azure, and then also other companies like Oracle and HP joined the game. In July 2010, Rackspace Hosting and NASA jointly launched an open source cloud software initiative known as OpenStack, intended to help organizations offering cloud computing services running on standard hardware.

"Cloud is the powerhouse that drives today's digital organizations" said the Sid Nag, research vice president at Gartner firm. Nowadays cloud computing infrastructure is the backbone of the delivery pipeline of just about every digital service, from social media and streaming entertainment to connected cars and autonomous Internet of Things (IoT) infrastructure. "Worldwide end-user spending on public cloud services is forecast to grow 20.4% in 2022 to total $494.7 billion, up from $410.9 billion in 2021, according to the latest forecast from Gartner. In 2023, end-user spending is expected to reach nearly $600 billion" [38].

2.2 DEFINITION OF CLOUD COMPUTING

The IT consultancy Gartner was one of the first to define cloud computing in 2008, subsequently slightly updated:

> *Cloud computing is a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service using Internet technologies.*
>
> Gartner Glossary

This definition highlights some of the key aspects of cloud computing: scalability, elasticity, and delivered as a service over the Internet.

A more precise and comprehensive definition is the one by the National Institute of Standards in Technology (NIST). This definition received industry-wide acceptance and it became the de facto standard definition of what is and what is not cloud.

> *When agencies or companies use this definition they have a tool to determine the extent to which the information technology implementations they are considering meet the cloud characteristics and models. This is important because by adopting an authentic cloud, they are more likely to reap the promised benefits of cloud — cost savings, energy savings, rapid deployment and customer empowerment. And matching an implementation to the cloud definition can assist in evaluating the security properties of the cloud.*
>
> NIST computer scientist Peter Mell, October 2011

NIST published its original definition back in 2009, followed by 15 drafts in more than three years [39]:

> *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*
>
> The NIST Definition of Cloud Computing, September 2011

Then the definition continues in describing the cloud model: it is composed of five essential characteristics, three service models, and four deployment models.

2.2.1 *Essential characteristics*

The essential characteristics were meant as the properties a cloud solution should have.

- **On-demand self-service**. It means that customers can use cloud computing without human contact with service providers, thanks to an user-friendly self-service system that allows to access various cloud resources as needed.
- **Network access**. It is the ability of network infrastructure to connect with a wide variety of devices, among which mobile phones, laptops, tablets and workstations, to enable seamless access to computing resources across these diverse platforms. Broad network access is what makes the cloud available to any device from any location.
- **Resource pooling**. It means that "provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data center)". Without this the goal of energy savings would not be realized.

- **Rapid elasticity**. It is the ability to "scale rapidly outward and inward commensurate with demand". For instance if a service like a website suddenly has a lot more users because of a flash sale, it needs to be able to scale up the computing power to handle this quickly, and when the sale is over it should scale back down. Not all cloud services come with this automatically.

- **Measured service**. It means that the customer should only pay for what they use to avoid paying for a machine standing idle in a data center. Typically this is done on a pay-per-use or charge-per-use basis. In practice vendors are not always making it as transparent, as NIST would have wanted them to ("resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service"). It can be very difficult or impossible for a customer to validate the correctness of the metering. The unit used to measure can differ greatly, e.g. it can be storage, processing, bandwidth, and active user accounts.

Although these characteristics were helpful eleven years ago for distinguishing between old-fashioned hosting and cloud solutions, the lines are more blurred today. In practice, these characteristics are good guidelines of what can be expected from most cloud solutions today, but they are not an accurate reflection of all cloud computing. We need to think these characteristics as common characteristics that we would expect to find most often.

2.2.2 *Service models*

There are three main types of cloud computing services, and each of them provides different levels of control, flexibility, and management so that users can select the proper set of services for their needs. A schematic representation of the three different service models and what users can manage is shown in Figure 2.1.

- **Infrastructure as a Service (IaaS)**. It is the most basic form where computing resources are provided and the cloud consumer installs and manages the needed software. IaaS provisions storage, compute, and networking services through either a virtualized image (for more details on virtualization see Sec. 2.4.3), a container (for more details on containers see Sec. 2.4.5) or directly on the computer systems (also known as "bare-metal") [41]. The general purpose of an IaaS environment is to provide consumers with a high level of control and responsibility over its configuration and utilization [37]. The IT resources provided are generally not preconfigured, leaving the administrative responsibility directly upon the consumer. This model is therefore aimed at those consumers that require a high level of control over the cloud-based environment they intend to create.

- **Platform as a Service (PaaS)**. This service model refers to a platform that can be used to program applications by the consumer. It is possible to write code and configure the service, whereas the vendor manages the underlying infrastructure. PaaS vendors

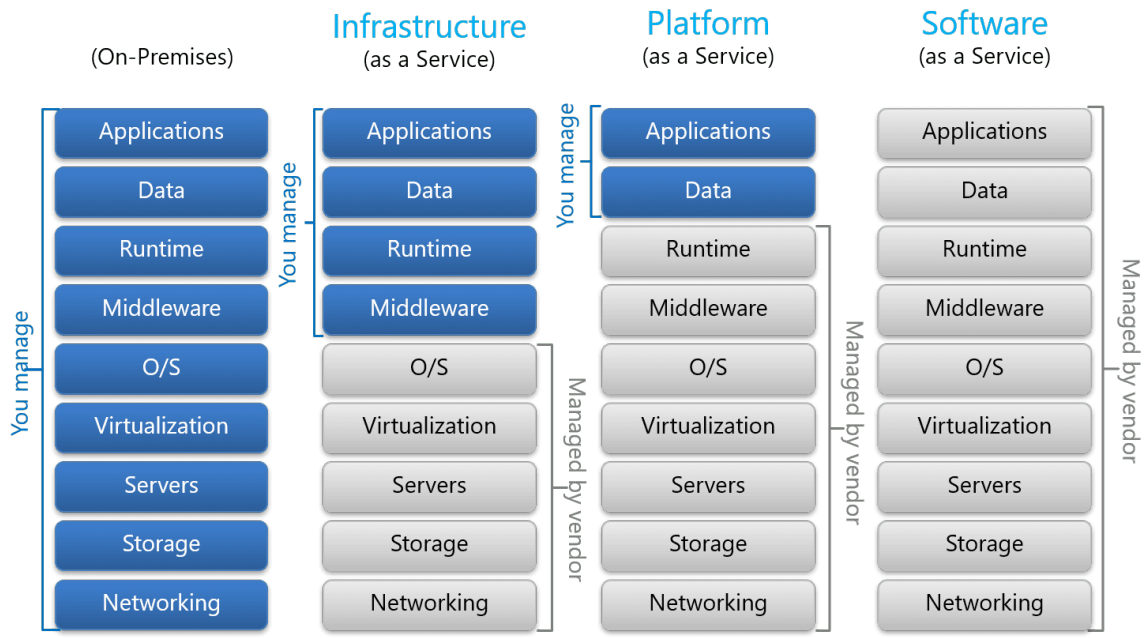


Fig. 2.1: Diagram of the three different cloud computing service models: Infrastructure as a Service, Platform as a Service, Software as a Service [40]. Each of them has different levels of control, flexibility, and management by the users compared to on-premises solutions.

create a "ready-to-use" environment typically comprised of already deployed and configured IT resources, and where pre-packaged products and tools (like libraries) used to support the entire delivery lifecycle of custom applications are already in place. By working within a ready-made platform, the cloud consumer is spared the setting up and maintaining the bare infrastructure IT resources provided via the IaaS model. Conversely, the cloud consumer is granted a lower level of control over the underlying IT resources that host and provision the platform.

- **Software as a Service (SaaS)**. It is a model where everything is managed by the vendor and users have only limited options to configure the software. The applications offered are products of the vendor and are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. Some SaaS examples: Gmail and in general Google Apps, Office365, social media such as Facebook, Twitter, Instagram, etc.

Many specialized variations of the three cloud service models emerged, e.g. Function as a Service (FaaS), Database-as-a-Service (DaaS), Communication-as-a-Service(CaaS), Integration-Platform-as-a-Service(IPaaS), Testing-as-a-Service(TaaS). In particular, in Sec. 2.2.2.1 we focus on the ML as a Service (MLaaS) model.

2.2.2.1 *Machine Learning as a Service*

The amount of data generated is continuously growing from global data sources like social media, web sites, mobile applications, with the risk that these data become useless without proper preparation and processing [42]. Many different ML algorithms are used to extract

valuable knowledge from data (see Ch. 1), e.g. for scientific modeling, consumer behavior, energy consumption forecasting, related article recommendation and user trends. Large companies could have enough resources to invest in their own ML solutions, but small companies, developers and researchers in general have difficulties when try to understand how ML works and when building their own solutions or integrating with third-party ones. In addition, ML can require computational resources with impracticable costs. One way to solve these issues is by using a MLaaS solution. In this situation multiple users will use the same platform, computational resources can be shared or allocated on demand, reducing overall costs. Users can have access to ML solutions efficiently from anywhere, at any time, without worrying about the implementation and computing resources, focusing mainly on the data itself.

MLaaS is a well-known concept in industry: the MLaaS market was valued at USD 2.26 billion in 2021, and it is expected to reach USD 16.7 billion by 2027 [43]. Major IT companies offer such solutions to their customers, e.g Amazon SageMaker, Microsoft Azure ML Studio, Google AI Platform, and IBM Watson ML Studio are prominent implementations of this concept. Usually, MLaaS is used as an umbrella definition of various cloud-based platforms that provide a web service to users interested in ML tasks. These platforms allows to perform ML pipelines, i.e. a sequence of steps typically taken when a user faces a ML problem, like pre-processing the data, identifying the most important features for the task, choosing a ML model, tuning parameters of the model, training the model, evaluating and using it for inference. Depending on the chosen solution a MLaaS system can cover the full spectrum between extreme simplicity (turn-key, nonparametric solutions) and full customization (fully tunable systems) [44]. Some are simple black-box systems where the classifier used is not even revealed, whereas others offer users the choice in everything.

Leading cloud providers offer MLaaS solutions with different interfaces and Application Programming Interfaces (APIs), mostly designed to cover standard use cases, e.g. classification, regression, clustering, anomaly detection, performed in different sectors like natural language processing and computer vision (see Figure 2.2).

These platforms simplify and make ML accessible to even non-experts, ensuring affordability and scalability as these services inherit the strengths of the underlying cloud infrastructure. Moreover, the MLaaS solutions are well integrated with the rest of the provider's portfolio of services which thus offers a complete solution. There are several aspects in which the MLaaS platforms help users [46].

- **Data management**. As many companies are moving data from on-premise storage to cloud storage systems, the need to properly organize data arises. MLaaS platforms are products of cloud providers which offer cloud storage, and at the same time they provide ways to properly access and process data for ML usage.

- **Access to ML tools**. There is a wide range of ML tools that users can try for different use cases. Data scientists do not need to worry about the actual computations of the operations because they are abstracted by MLaaS providers.

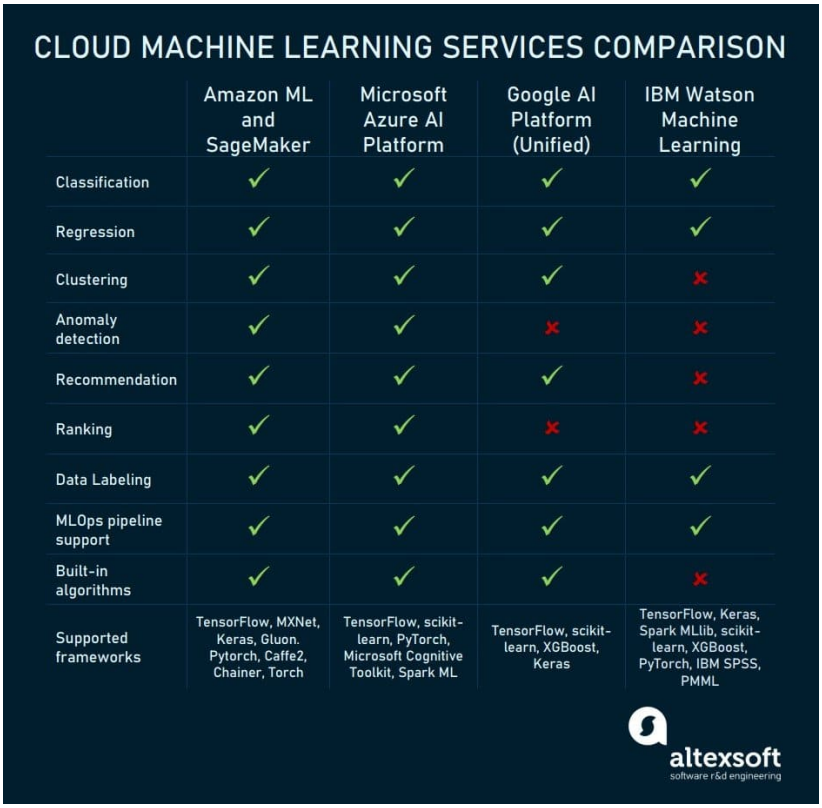


Fig. 2.2: Comparison of some services offered by common MLaaS providers: Amazon, Microsoft Azure, Google Cloud AI, IBM Watson [45].

- **Ease of use**. MLaaS offers data scientists the means to get started quickly with ML without having to take care of software installation processes. With MLaaS the provider's data centers handle the actual computation, without impacting on the user's on-premises resources.

- **Cost efficiency**. MLaaS is beneficial because users only pay for resources when they are actually used.

The popularity of DevOps among the software development community gave birth to the term "MLOps" [45]. DevOps is an approach for software development to optimize software development processes by focusing on short and fast releases, applying a high level of automation to routine tasks. Further information on DevOps is provided in Sec. 2.6. MLOps applies the same principles to ML, which led to the emergence of automated data management, model training/deployment, and monitoring. MLaaS providers offer tools for MLOps practitioners to manage ML pipelines.

In the following we provide a brief overview of some of the most used platforms offering MLaaS solutions.

Amazon Sagemaker

Originally Amazon provided the Amazon ML platform for ML tools but has not updated it anymore since 2021. The service is still functional but does not accept new users. This is because the Amazon SageMaker platform and all of its corresponding services are su-

perior to Amazon ML, and basically deliver the same functionality to users. SageMaker offers an environment that simplifies ML workflows by providing tools for quick model building and deployment. The ML algorithms provided are optimized to run efficiently with extremely large data in a distributed environment, as well as the users can add their own methods and run models leveraging the Sagemaker deployment features [47].

Microsoft Azure ML Studio
Azure ML Studio, is a development environment that creates a resourceful playground both for entry-level and experienced data scientists. Most operations in Azure ML Studio can be completed using a graphical user-friendly drag-and-drop interface, which also allows to visualize each step within the workflow. It also allows to run ML models on Azure, on-premise, or even Edge devices. It integrates well with Visual Studio and Github to make it easy for software engineers to access and track the development. Azure ML offers the AI Gallery, i.e. a hub of ML solutions and data science model templates provided by the Azure community. One of the main benefits of using Azure is the variety of algorithms available to play with.

Google AI Platform
Google AI Platform comprehends tools for ML that previously existed separately, like Google Cloud AutoML. AutoML is a cloud-based ML platform that offers a variety of ML products for beginner data scientists, it is fully integrated with all Google's services and it stores data in the cloud. The trained models can be deployed via the REST API interface. Although many languages and frameworks are supported, its focus is clearly on Tensorflow, Google's product.

IBM Watson ML Studio
IBM Machine Learning platform is organized like the previous providers, but does not support the video analysis APIs. The platform offers two approaches: automated and manual (for expert practitioners). Watson Studio has an AutoAI system that brings a fully automated interface for data processing and model building that needs little to no training.

All the MLaaS solutions described above offer multiple services, many of which are in common. These include the integration with Jupyter notebook to easy access data for exploration and analysis, and to program ML models using popular frameworks. Anyway these solutions differ in terms of available algorithms, of required skill sets, and in tasks they can cover. For this reason there is no best platform at all and the choice depends on the specific use case. Without forgetting that the velocity of change is impressive and could happen that a user chooses one vendor and suddenly another one will roll out something that matches his/her business needs. And using a MLaaS solution may not necessarily be the best choice for a given use case. The user can adopt MLaaS for some part of the workflow and other tools for others. Lastly, when not to use MLaaS [46]?

- When data need to be secure and on-premises;

- when you need a lot of customization (but in some cases MLaaS could still be useful);
- when you need to optimize training or serving costs of complex algorithms and so it is preferable to take the whole infrastructure on-premises.

When to use MLaaS?

- When you are already using services from a cloud provider and integrating their MLaaS services to your system would be a good addition;
- when some/many use cases can be outsourced to an API;
- when you are dealing with a large amount of data and you need to carry out tests frequently on data;
- when you run a microservice-based architecture in your company, MLaaS would help in proper management of some of those services.

2.2.3 *Deployment models*

A cloud deployment model is a specific configuration of environment parameters such as the accessibility and proprietorship of the deployment infrastructure and size. This means that deployment types vary depending on who controls the infrastructure and where it is located.

- **Public cloud**. A public cloud is defined as computing services offered by third-party providers over the public Internet, making them available to anyone who wants to use or purchase them [48]. They may be free or sold on-demand, allowing customers to pay only for the CPU cycles, storage, or bandwidth they consume. Public clouds can save companies from the expensive costs of having to purchase, manage, and maintain on-premises hardware and application infrastructure, of which the cloud provider is held responsible. Public clouds can also be deployed faster than on-premises infrastructures and with an almost infinitely scalable platform.
- **Community cloud**. A community cloud is similar to a public cloud except that its access is limited to a specific community of cloud consumers. It is owned, managed, and operated by one or more organizations in the community, a third party, or a combination of them.
- **Private cloud**. A private cloud is owned by a single organization enabling it to use cloud computing technology as a means of centralizing access to IT resources by different parts, locations, or departments of the organization. It gives businesses many of the benefits of a public cloud, including self-service, scalability and elasticity, with the additional control and customization available from dedicated resources over a computing infrastructure hosted on-premises [49]. Moreover, private clouds deliver a higher level of security and privacy through both company firewalls and internal

hosting to ensure operations and sensitive data is not accessible to third-party providers. One drawback is that the IT department of the company is held responsible for the cost and accountability of managing the private cloud, so private clouds require the same staffing, management and maintenance expenses as traditional data center ownership.

- **Hybrid cloud**. A hybrid cloud technology strategy merges a private cloud or an on-premises infrastructure, or both, with a public cloud environment to create a single cloud computing environment. For instance, a cloud consumer may choose to deploy cloud services processing sensitive data to a private cloud and other, less sensitive cloud services, to a public cloud.

Additional variations of the four base cloud deployment models can exist, like the virtual private cloud, inter-cloud and multi-cloud. The latter involves multiple public cloud computing platforms or providers to handle various business tasks. A business might perform some tasks using AWS, another set of tasks with Google Cloud Platform, and yet other tasks with Microsoft Azure.

2.3 WHY CLOUD COMPUTING?

The cloud is transforming and will continue to transform the computing industry becoming the default place to go. To better understand the reasons of this evolution, we need to focus to what drives the adoption of the cloud [35]. Nowadays for a company the alternative to a cloud solution is to use an on-premises data center. Even companies that have the scale to run a competitive on-premises data center like Netflix have moved to the cloud. Different companies have different priorities, but most will have at least one of the following arguments as a key driver for adopting the cloud over on-premises solutions.

- **Economy**. The economic concerns are generally primary drivers of cloud adoption. The cloud helps users move from CAPEX to OPEX, meaning that costs are moved from capital expenses (CAPEX), that are initial investments, to operational expenses (OPEX), or ongoing costs. This is a very important aspect since one of the key metrics used to evaluate a company by investors and shareholders is the cash flow: the better the cash flow, the better the valuation of the company. Another aspect is that a server is a depreciating asset, meaning that it is something that loses its value over time. Cost is more predictable in the cloud. Typically there is a monthly fee, although that can vary with consumption for some types of cloud services. OPEX based on the cloud are easier to scale with changes in the market. Indeed if a company suddenly sees a loss in the revenue, a cloud model based on paying monthly fees and on consumption is easier to scale than a CAPEX model, where capital has been committed at the beginning of a multiyear period. Moreover, lower levels of CAPEX will inevitably lead to lower levels of debt improving the company credit rating. Consequently, investment in the cloud can be seen as part of a virtuous circle from an economic perspective. Another economic driver is the concept of economy of scale. The basic

idea is that the cost of adding one unit of infrastructure decreases with the number of units already managed.

- **Security**. It is opinion of many that cloud is insecure compared to on-premise, but recent advances in securing the cloud and changes in risk are changing that conception. A company relying on its own data center needs to keep on top of all threats and attacks, 24/7, and this is difficult and costly as the local IT security department needs to be staffed with specialists. In the cloud, some threats are easier to mitigate. Moreover, cloud vendors now offer systems that will run reports, identify places where personally identifiable information may be stored on internal resources, and log who accessed the information and when. They also have the possibility to automatically detect other vulnerabilities in the system configurations that would otherwise have taken an extensive audit to detect. Encryption and Identity and Access Management (IAM), which makes sure that only authenticated and authorized users can access system resources and data, are offered into the cloud by default whereas they should be set up and maintained on-premises.

- **Resilience**. Generally organizations need to make sure that their operations do not cease in the face of a natural or other disaster, so they need to ensure the resilience of their IT operations. The natural disasters like hurricanes, flooding, earthquakes or volcanic eruptions have the potential to completely destroy a data center. For this not to happen a company has to build at least one backup site. This site needs to run exactly the same system setup as the primary one and there has to be automatic failover when the primary site fails. This is both expensive and difficult to implement. In the cloud, this is in some cases virtually reduced to a few clicks that will give users even better geographical redundancy than was feasible for the company alone. Users can do this at the individual server level and can chose to have the failover in a specific geographic region. Moreover, it is possible to specify a template of the VMs that should spin up in the event of a server failure, saving costs and energy for the company. Another example of usage of the cloud for the resilience feature is for backups.

- **Elasticity**. We already described this aspect as one of the five essential characteristics of a cloud solution. From the economic point of view elasticity avoid to make important CAPEX and to pay system resources that initially are not needed, but at the same time when an high demand occurs it is important to integrate more resources fast enough and at the right locations to scale with demand. Then, when the demand decreases again, the resource consumption contracts. This dynamic is difficult to implement in an on-premises data center since resources or licenses that are no longer needed can not simply be returned. Connected to elasticity is the concept of agility that means both the ability to quickly react to changing market conditions and the fact that system resources are available on demand when needed. In the cloud, when a particular database is needed, it is not necessary first to contact sales at the software company, get a quote, purchase it, and install it. All of this is reduced to

searching for it on the website of the cloud provider and then clicking it. With this kind of usage it is easier to support innovation. Indeed an experiment, a proof of concept, can be carried out and once it is no longer needed, it can be deleted and no further payments have to be made. This makes innovation and experimentation much cheaper.

- **Sustainability**. An emerging driver is the cloud sustainability in terms of energy consumption. There are different reasons for that. First of all, a cloud provider may ensure a much better utilization of resources. When a single user turns on his/her own computer, the CPU runs no matter if it is used or not. In a company where people work for eight hours, the computing resources will be used only in that time frame. This means that a traditional server running 24/7 is using three times as much energy as is needed. In the cloud, thanks to the elasticity and pooled resources, it is possible to secure a much better utilization since multiple customers can use the same machines. When utilization drops, machines will turn off and come back up again when usage increases again. Moreover, it is easier to have efficient energy consumption into larger data centers by reducing the energy for cooling, capturing the excess heat and turning it into hot water, setting up solar cells to harvest energy. For example, Microsoft vowed to be carbon negative by 2030, which means that they will produce more clean energy than they consume.

2.4 CLOUD-ENABLING TECHNOLOGIES

Modern-day clouds are supported by a set of primary technology components that were born and matured prior to the advent of cloud computing, although cloud computing advancements helped further evolve these cloud-enabling technologies [37]. In the following subsections we provide details of such technologies.

2.4.1 *Broadband networks and Internet architecture*

All clouds must be connected to a network and this implies a dependency on internetworking. Internetwork, or the Internet, allow for remote provisioning of IT resources and is supportive of network access. Cloud consumers can access the cloud using private and dedicated network links in LANs, but most clouds are Internet-enabled. The Internet topology became a dynamic and complex aggregate of highly interconnected Internet Service Providers (ISPs), where smaller branches extend from these major nodes of interconnection through smaller networks until reaching the Internet-enabled electronic devices. Two fundamental components used to construct the internetworking architecture are connectionless packet switching and router-based interconnectivity. The former is the data transmission process where the data flow (which goes from the sender to the receiver) is divided into packets of a limited size that are received and processed through network switches and routers, then queued and forwarded from one intermediary node to the next. Each packet carries the necessary location information, such as the Internet Protocol (IP) or Me-

dia Access Control (MAC) address. The latter refers to routers that allow to process and forward each packet individually, even when successive packets are part of the same data flow, while maintaining the network topology information that locates the next node on the communication path between the source and destination node.

The communication path that connects a cloud consumer with its cloud provider involve multiple ISP networks determined at runtime, and the interaction with other network technologies:

- physical networks, such as Ethernet, that connect adjacent nodes and are used to transmit IP packets;
- transport layer protocols, such as the Transmission Control Protocol (TCP), that use the IP to provide a communication support for the navigation of data packets across the Internet;
- application layer protocols, such as Hypertext Transfer Protocol (HTTP), that use transport layer protocols to standardize and allow specific transferring methods of data packet over the Internet.

In a traditional on-premise deployment model, applications and various IT solutions are commonly hosted on centralized servers and storage devices that resides in the data center of the organization. End-user devices, like laptops and smartphones, access the IT resources of data center through the corporate network, which provides Internet connectivity. Organizations adopting this deployment model can directly access the network traffic to and from the Internet on which they have complete control, and can safeguard their corporate networks using firewalls and monitoring software.

In a cloud environment, cloud providers configure IT resources to be accessible for both internal and external users through an Internet connection. This internetworking architecture benefits internal users who require ubiquitous access to corporate IT solutions, as well as cloud consumers who want to provide Internet-based services to external users (see Figure 2.3). Moreover, the Internet connectivity offered by cloud providers is superior compared to the connectivity of individual organizations.

Nowadays the most widespread way of accessing VMs in the cloud across every cloud vendor is to use the Secure Shell (SSH) protocol, developed to provide an encrypted way of connecting to a remote computer using a client/server model through a network.

2.4.2 *Web technology*

Web technology is used as both the implementation means and the management interface for cloud services. The World Wide Web is a system of interlinked IT resources that are accessed through the Internet, where the Web browser client and the Web server are the basic components. Other components used to improve Web application characteristics, such as scalability and security, reside between the client and the server, e.g. proxies, caching services, gateways, and load balancers.

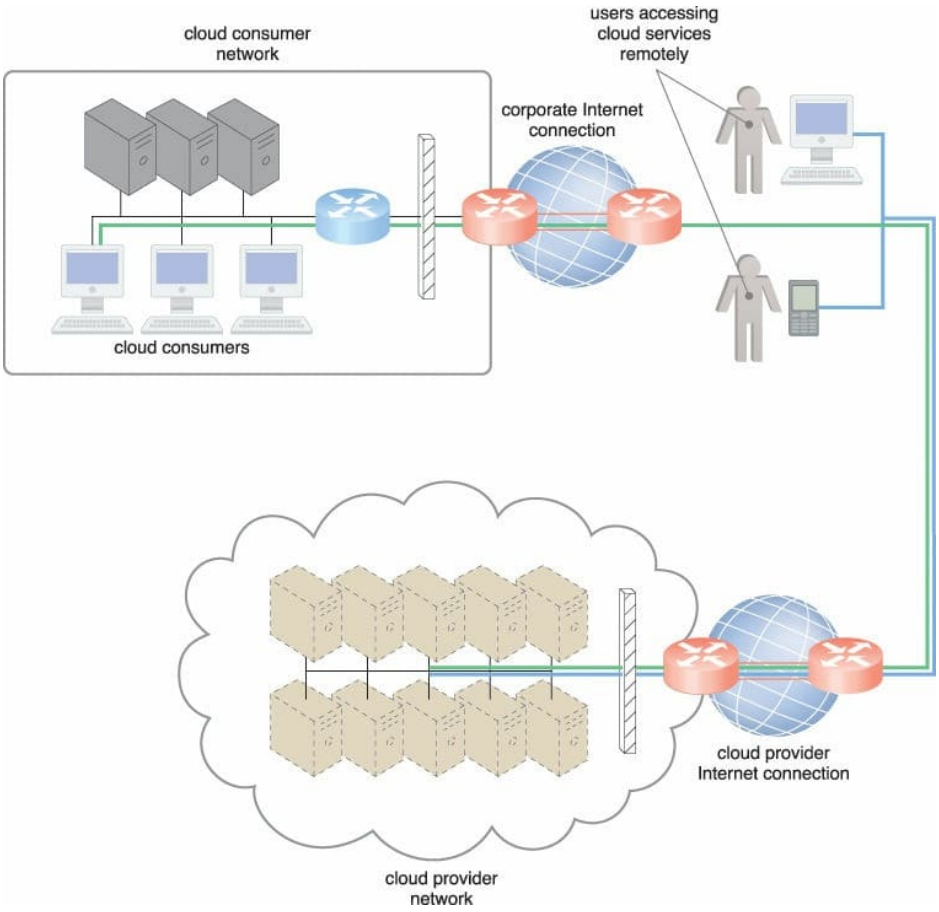


Fig. 2.3: The internetworking architecture of an Internet-based cloud deployment model [37]. Internet connects non-proximate cloud consumers, end-users, and the network of the cloud provider.

Common Web browser operations are performed using three elements: Uniform Resource Locator (URL) as identifier pointing Web-based resources, HTTP as primary communication protocol used to exchange content and data throughout the World Wide Web between client and server, and Markup Languages (e.g. HTML, XML) as lightweight means of expressing Web-centric data and metadata. A Web browser can request to execute an action, like read, write, update, or delete, on a Web resource on the Internet identified by an URL. The request is sent using HTTP to the resource host. The Web server locates the Web resource and performs the requested operation, followed by a response being sent back to the client which may be comprised of content that includes HTML and XML statements.

A distributed application that uses Web-based technologies (and generally that relies on Web browsers for the presentation of user-interfaces) is typically considered a *Web application*. Such applications can be found in all kinds of cloud-based environments thanks to their high accessibility. A common architecture of Web applications is based on three layers: the presentation layer which represents the user-interface, the middle layer which implements the application logic, and the data layer which is comprised of persistent data stores (e.g. databases). PaaS ready-made environments allow cloud consumers to develop and deploy Web applications. Unlike Web applications, Web services do not necessarily have a user interface since it is used as a component in an application.

2.4.2.1 *Overview of HTTP*

HTTP is a protocol that enables a client and server to communicate over a network, and it is the foundation of any data exchange on the Web. A typical flow over HTTP involves a client machine making a request to a server, which then sends a response message.

- Each HTTP request made by a client across the Internet carries with it a series of encoded data containing different types of information [50]. A typical HTTP request contains: the HTTP version type, a URL, an HTTP method, HTTP request headers, and optionally the HTTP body. The URL locates an existing resource on the Internet and includes the information needed to access the resource. Generally a URL for HTTP (or HTTPS) is made up of three or four components: a scheme, i.e. the protocol to be used to access the resource on the Internet (it can be HTTP or HTTPS), a host which identifies the host that holds the resource (e.g. `www.example.com`) and can be followed by a port number, a path which identifies the specific resource in the host, a query string which provides a string of information (made by key-value pairs separated by an &) that the resource can use for some purpose (e.g. parameters for a search or data to be processed). The HTTP method indicates the action that the HTTP request expects from the server. Three of the most common HTTP methods are `GET`, `POST` and `DELETE`. A `GET` request expects some information back in return while a `POST` request typically indicates that the client is submitting information to the web server (such as username and password inserted in a form). Lastly, a `DELETE` request is used to delete a resource. HTTP headers contain text information stored in key-value pairs. They communicate core information, such as what browser the client is using or authentication credentials. The body of a request is the part containing any other information being submitted to the web server.
- An HTTP response is what a web client (often browser) receives from an Internet server as answer to an HTTP request. A typical HTTP response contains: an HTTP status code, HTTP response header,s and optionally the HTTP body. HTTP status codes are 3-digit codes with the following meaning: 1xx Informational, 2xx Success, 3xx Redirection, 4xx Client Error, 5xx Server Error. The "x" refers to a number between 0 and 9. For example, after a client requests a web page, if the request is properly completed the response has the status code `200 OK`. If the response starts with a "4" or a "5" means there is an error and the web page will not be displayed, e.g. `404 NOT FOUND` status code appears when making a typo in the URL. Similarly to an HTTP request, an HTTP response comes with headers that convey information like the language and format of the data being sent in the response body. Successful HTTP response to a `GET` request generally has a body containing the requested information, and in most web requests it is an HTML data which a web browser translates into a web page.

The HTTP protocol was designed in the early 1990s and then it has evolved over time [51]. It is an application layer protocol sent over TCP, or over a TLS-encrypted TCP connection (hence HTTPS, used for secure communications), though theoretically any transport

protocol can be used. Due to its extensibility, it is used to fetch hypertext documents, images and videos or to post content to servers, like with HTML form results.

HTTP is a client-server protocol where requests are sent by a user-agent (or a proxy on behalf of it). The user-agent is any tool that acts on behalf of the user. This role is primarily performed by the Web browser, but it may also be performed by tools used by Web developers, e.g. cURL that allows to send requests by command line. The server appears as only a single machine virtually, but it may actually be a collection of servers that share the load (load balancing), or a complex piece of software that interrogates other computers (like a cache, a DB server, or a e-commerce server), partially or totally generating the document on demand.

Numerous computers and machines transfer the HTTP messages between client and server, and most of them operate at the transport, network, and physical levels, becoming transparent at the HTTP layer but with a potential significant impact on performance. Those operating at the application layers are generally called proxies, which can be transparent, forwarding the requests they receive without altering them, or non-transparent, in which case they will modify the request in some way before passing it to the server. Proxies can perform several functions: caching, filtering, load balancing, authentication, and logging.

The basic features of HTTP can be summarized as the following.

- **HTTP is simple**. HTTP is generally designed to be simple and human readable, even with the added complexity introduced in HTTP/2.
- **HTTP is extensible**. HTTP headers make this protocol easy to extend. New functionality can also be introduced with a simple agreement between client and server on a new header semantics.
- **HTTP is stateless, but not sessionless**. HTTP is stateless, i.e. there is no link between two requests being successively carried out on the same connection. But while the core of HTTP itself is stateless, HTTP cookies allow the use of stateful sessions. Using header extensibility, cookies can be added to the workflow allowing to create a session for each HTTP request to share the same context or the same state.

2.4.2.2 *Application Programming Interfaces*

An API is a set of rules that define how applications or devices can connect to and communicate with each other, creating an interface that provides a specific service to other pieces of software [52]. In practice an API is the messenger that delivers the request of the client to the server, and then delivers the response back to client. Whereas a user interface is designed for use by humans, APIs are designed for use by a computer or application. How do APIs work? A client application makes an API call in order to get the desired information. Then, after receiving a valid request, the API makes a call to an external program or web server, which subsequently will send back a response to the API with the requested information. Lastly, the API transfers the data to the client.

APIs can be classified according to the system and use case they are designed for [53].

- **Database APIs**. They enable communication between an application and a database management system. Developers work with databases by writing queries in order to access data, change tables, etc.
- **Operating systems APIs**. They define how applications use the resources and services of the operating system. Every OS has its set of APIs.
- **Remote APIs**. They are designed to interact through a communications network. Remote means that the resources being manipulated by the API are somewhere outside the computer making the request.
- **Web APIs**. This is the most common class of APIs. They provide machine-readable functionality and data transfer between web-based systems, representing a client-server architecture. Web APIs mainly deliver requests from web applications and responses from servers using HTTP.

In terms of release policies, web APIs can be divided in three main categories: public, private, and partner. Public APIs are open source and are available for any third-party developer. Private APIs are designed to improve solutions and services within an organization, remaining hidden from external users. Partner APIs are openly promoted but shared with business partners who have signed an agreement with the publisher.

API specifications detail the functional and expected behavior of an API, as well as the fundamental design philosophy and supported data types. They contain both documentation and API definitions. Protocols provide users with a set of defined rules for API calls that specify the accepted data types and commands. In the following, some of the most used protocols are described.

- **SOAP**. APIs designed with Simple Object Access Protocol (SOAP) use XML for their message format and receive requests through HTTP or SMTP. SOAP simplifies the sharing of information for apps running in different environments or written in different languages.
- **REST**. Representational State Transfer (REST) is a set of web API architecture principles, meaning that there are no official standards (unlike those with a protocol). To be a REST API (also known as a RESTful API), an API must follow six architectural constraints [54, 55]. Uniform interface, i.e. API requests for the same resource should look the same, no matter where the request comes from. Client-server decoupling, i.e. client and server applications must be completely independent of each other, where the client only should know the Uniform Resource Identifier (URI) of the requested resource and the server application shouldn't modify the client application. Statelessness, i.e. each request must include all the information necessary for processing it, and server applications are not allowed to store any data related to a client request. Cacheability, i.e. resources should be cacheable on the client-side or server-side when it is possible, with the aim to improve performance on the client-side while increasing scalability on the server-side. Layered system architecture, i.e.

the calls and responses go through different layers intended as intermediaries in the communication loop between client and server. Code on demand (optional), i.e. in certain cases responses can contain not only static resources, but also executable code, which should only run on-demand. RESTful systems support multiple formats to store and exchange data (e.g. plain text, XML, HTML, YAML, JSON), while SOAP only allows XML, and this is one of the reasons why REST is a prevailing choice for building public APIs these days.

- **gRPC**. Developed by Google and released for public use in 2015, gRPC is an open-source Remote Procedure Call (RPC) to achieve high-speed communication between microservices. gRPC allows developers to integrate services programmed in different languages. It uses the Protobuf (protocol buffers) messaging format, which is a highly-packed and highly-efficient messaging format for serializing structured data. Firstly, developers need to define the structure of the data they want to serialize, then they use the protocol buffer compiler to generate the data access classes in the programming language of their choice. Lastly, the data is compressed and serialized in binary format at runtime. gRPC is designed for HTTP/2, a major revision of HTTP that provides significant performance benefits over HTTP 1.x.

2.4.3 *Data center technology*

There are many advantages in grouping IT resources in close proximity with one another rather than having them geographically dispersed: power sharing, higher efficiency in the usage of shared IT resources, and improved accessibility for IT personnel, just to mention a few. Modern data centers have specialized IT infrastructures used to house centralized IT resources, such as databases, servers, and networking devices. Data centers are typically characterized by many technologies and components. In the following we provide the description of some of them.

- **Virtualization**. Data centers consist of both physical and virtualized IT resources. Virtualization is the process of converting a physical IT resource into a virtual IT resource, i.e. abstracting the physical computing and networking IT resources as virtualized components that are easier to allocate, operate, and control. Most types of IT resources can be virtualized, including server, storage, network and power. Let's consider the case of creation and deployment of a virtual server[1], performed using a virtualization software. The first step is the allocation of physical IT resources, followed by the installation of an operating system. A virtual server has its own guest operating systems, independent of the operating system in which it was created. Both the guest operating system and the application software running on the virtual server are unaware of the virtualization process. Indeed the virtualized IT resources are installed and executed as if they were in a separate physical server, resulting in an uniformity of execution, i.e. programs run on physical systems as they would on

1 The terms virtual server and VM are used synonymously throughout this thesis.

virtual systems. The virtualization software runs on a physical server called a host or physical host, which make accessible the underlying hardware. This software is sometimes referred to as a Virtual Machine Monitor (VMM), but most commonly known as a hypervisor. Thanks to the hardware independence, a virtual server can be easily moved to another host, automatically resolving hardware-software incompatibility issues. Virtual servers are created as virtual disk images containing binary file copies of hard disk content. These images can be accessed from the host operating system and can be used to replicate, migrate, and back up the virtual server. A key aspect of virtualization is that such technology enables different virtual servers to share one physical server, i.e. multiple virtual servers can be simultaneously created in the same host. This process is called server consolidation and it is commonly used to increase optimization of available IT resources, load balancing and hardware utilization. This fundamental capability unlock common cloud features, e.g. on-demand usage, resource pooling, scalability, elasticity, and resiliency.

- **Standardization and modularity**. Data centers are built using standardized commodity hardware and designed with modular architectures, aggregating multiple identical building blocks of facility infrastructure and equipment to support growth, scalability and speedy hardware replacements. Standardization and modularity are key factors for reducing investment and operational costs since they allow to have economies of scale for the acquisition, deployment, operation, and maintenance processes.

- **Automation**. Data center automation is the process by which routine workflows and processes of a data center (e.g. scheduling, monitoring, maintenance, application delivery) are managed and executed without human administration [56]. Automation increases agility and operational efficiency in a data center, reducing the time IT needs to perform routine tasks and enables to rapidly deliver to users on demand services in a repeatable, automated manner. There are many configuration management tools for a data center automation, representing at the same time a key aspect of a DevOps project, like Ansible, Puppet, and Chef (we provide more details in Sec. 2.6). In this context it is worth mentioning OpenStack as open source cloud computing platform, which enables to control large pools of compute, storage, and networking resources in a data center through a dashboard or through the OpenStack APIs. It helps to build a cloud infrastructure or manage local resources as if they were a cloud by automating the building and management of virtual servers and other virtualized infrastructures.

- **Remote operation and management**. Most of the operational and administrative tasks of IT resources in data centers are controlled using remote consoles and management systems. Technical personnel are not required to enter the rooms of the data center, except to perform highly specific tasks, e.g. equipment handling and cabling or hardware-level installation and maintenance.

- **High availability**. To sustain availability data centers are designed to operate with high levels of redundancy. Data centers usually have redundant, uninterruptable power supplies, cabling, and environmental control subsystems in anticipation of system failures.

2.4.4 *Multi-tenant technology*

The multi-tenant application design is used to allow multiple users (tenants) to access the same application simultaneously. Each tenant has his/her own view of the application, that uses, administers, and customizes it while remaining unaware of other tenants that use the same application. In such applications, tenants do not have access to data and configuration information that is not their own. Tenants can individually customize features of the application, e.g user interface, business process, data model, and access control. In general a multi-tenant application architecture is significantly more complex compared to a single-tenant application one. Indeed, in the former the sharing of various elements (including portals, data schemas, middleware, and databases) by multiple users is needed, while maintaining security levels that segregate individual tenant operational environments.

Multi-tenancy is sometimes mistaken for virtualization but they are different concepts. Virtualization means that multiple virtual copies of the server environment can be hosted by a single physical server and each copy can be provided to different users, can be configured independently, and can contain its own operating system and applications. Multi-tenancy means that a physical or virtual server hosting an application can be used by multiple different users, each of whom is as if they had exclusive use of the application.

2.4.5 *Containerization*

Containerization is a type of virtualization used to deploy and run applications and cloud services without the need to deploy a virtual server for each of them. The multiple isolated user-space instances or isolated runtimes that can coexist in the operating system kernel of a physical or virtual server are mainly known as containers. When a cloud service executes within a container, from its point of view it is as if was running on a real computer, and it can only see the content of the container and the devices attached to the container. On the contrary, a cloud service running on a physical or virtual server operating system can see all of the provided resources, e.g. connected devices, ports, folders, files, network shares, and CPUs.

Containers are an abstraction at the application or service layer which package code and dependencies together, and they can be spun up and retired very quickly according to predefined specifications, much faster than virtual servers. Moreover, an efficient resource utilization is achieved by significantly reducing the CPU, memory and storage usage footprint compared to virtual servers, as the user can restrict the amount of resources each container consumes. Indeed, if VMs are used to run applications independently (one ser-

vice per VM) they are underutilized, and resizing a VM is not an easy task for a production application. On the other hand, containers can run with very minimal CPU and memory requirements, multiple containers can be run inside a VM for application segregation, and resizing a container takes seconds.

A single process of a cloud service is usually deployed in each container, though more than one service/process can be deployed in each. A cloud service deployed in a container normally shares the same lifecycle with the container, i.e. it will start, stop, pause or resume when the container does.

2.4.5.1 *Docker*

Docker is the most common open source platform for building, deploying, and managing containerized applications. It is developed by Dotcloud, it is written in the Go programming language and takes advantage of several features of the Linux kernel, like namespaces and control groups, to create containers. The basic terminology to understand how Docker works, that can be similarly extended to other container solutions, is the following [57].

- **Docker Engine**. It is a specialized software deployed in the operating system of a physical or virtual server to abstract the required resources, and enable the definition and deployment of containers. Docker Engine acts as a client-server application with a server running a long-running daemon process, with APIs which defines interfaces that programs can use to talk to and instruct the Docker daemon, and with a Command Line Interface (CLI) client. The CLI uses the Docker APIs to create and manage Docker objects, e.g images, containers, networks, and volumes [58]. The Docker client and daemon communicate using REST APIs over UNIX sockets or a network interface.

- **Dockerfile**. It is a simple text file containing the list of CLI instructions that Docker Engine will run in order to assemble the Docker image, thus automatizing the deployment.

- **Docker image**. It contains executable application source code as well as all the libraries, tools, and dependencies that the application code needs to run as a container. A Docker image is an immutable (unchangeable) file, and due to their read-only quality it is sometimes referred to as snapshot. A Docker image is organized in a layered fashion and every instruction on a Dockerfile is an additional layer in the image. It is possible to build a Docker image from scratch, but most of the time the image is based on a parent image which can be pull down from common repositories.

- **Docker container**. It is the running instance of a Docker image. While Docker images are read-only files, containers are live, ephemeral, executable content. By default, a container is relatively well isolated from other containers and its host machine, though the user can control how isolated a container storage, network or other underlying subsystems are from other containers or from the host machine [59].

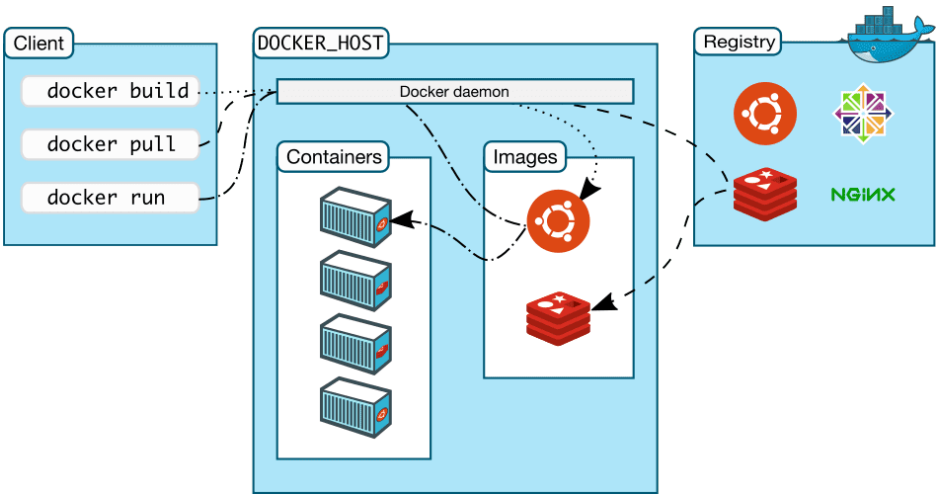


Fig. 2.4: Representation of the client-server architecture adopted by Docker [59]. Through the REST APIs or the CLI the Docker client talks to the Docker daemon, which builds, runs, and distributes Docker containers. The images used to run a container are retrieved from a registry, which is Docker Hub by default.

- **Docker registry**. It is a storage and distribution system for Docker images. A user can define his/her own private registry though Docker is configured to look for images on Docker Hub by default. Docker Hub is the Docker public registry of images with over 100 thousands images provided by commercial software vendors, open source projects, individual developers and Docker itself. The users can share their images using Docker Hub and download the desired image from a specific project, or use one of them as a starting point for any containerization project.

For a schematic representation of the client-server architecture adopted by Docker, see Fig. 2.4.

A container is a process with enough isolation of userspace components to give the feeling of a separate operating system [60]. A real-world analogy could be an apartment building. Even though it is a single big building, each flat is isolated and it is owned by individual households with their own identity. Other people do not have visibility into a flat unless they allowed in. Similarly, this can be related to a single host containing multiple containers. To isolate containers with their CPU, memory, IP address, mount points, and processes two Linux kernel features are needed, which are called namespaces and Control Groups (CGroups).

- Namespaces sets boundaries for the containers, as they are responsible of mount points, users, IP address, processes management, etc. Each container will have its namespace, and the processes running inside that namespace will not have any privileges outside that namespace.

- When starting a container the resources used by it, e.g. CPU, memory, network, and IO resources, can be restricted using CGroups. If no limits are set up, it is the kernel that prioritizes and allocates resources for the services, and when multiple containers are running, a single container might use all the host resources, leaving other containers to crash because of resource unavailability.

If an application is made of multiple processes that are deployed in multiple containers and they all reside on the same host, a user can choose to adopt Docker Compose to manage the application's architecture. The user writes a YAML file where specifies the configuration of the services included in the application, then with a single command he/she can deploy and run containers from the configuration.

2.4.5.2 *Alternatives to Docker*

Although Docker is a robust standalone ecosystem of its own, which provides an extensive tool kit for managing the containerization process, there are alternatives to Docker that offer unique use cases and features, e.g. Podman, Singularity, LXD, Containerd, Buildah, BuildKit, Kaniko and RunC.

Let's take the example of Podman. Podman is an open source project of Red Hat and it is a relative newcomer to the containerization scene, with the first version being released in 2019. For a comparison, the first version of Docker was released in 2013. Docker and Podman have different philosophies and working approaches [61]. Podman relies on "pods", used to organize separate containers under a common denomination to manage them as single unit. Docker is an all-in-one platform, whereas Podman relies on specialized tools for specific duties, e.g it uses Buildah to build container images. Docker is a monolithic and independent tool with all the benefits and drawbacks implied, handling all of the containerization tasks throughout their entire cycle. Docker has a client-server logic mediated by a daemon, while Podman needs another tool (systemd) to manage services and support running containers in background. Recently Docker added rootless mode to its daemon configuration (previously Docker containers run only as root) but Podman used this approach first and promoted it as a fundamental feature. Rootless containers are considered safer than containers with root privileges.

Docker is the most popular and used platform for building, deploying, and managing containerized applications but nowadays its hegemony is no longer so obvious. This can be emphasized by the recent decision of Red Hat to no longer support the docker package for Red Hat Enterprise Linux (RHEL) 8 in favor of Podman, adopted as the preferred, maintained, and supported container runtime.

It is worth briefly describe the Singularity solution compared to Docker. Singularity is a framework that is mostly focused on bare-metal High Performance Computing (HPC) cluster systems. For an overview about what HPC is, see Sec. 2.5.2. Singularity was developed to increase "mobility of compute" in scientific applications [62] by enabling environments to be completely portable via a single image file and allowing for seamless integration with any scientific computational resources [63]. The architecture of Singularity is such that common users can safely run their containers on an HPC cluster system without the possibility of root privilege escalation. Singularity uses a simpler approach than Docker without the daemon process, and it makes use of Linux kernel namespace feature to isolate processes. Singularity can be used to run massively parallel applications which leverage fast InfiniBand interconnections and GPUs. These applications suffer a minimal performance loss since Singularity was designed to run "close to the hardware".

2.4.5.3 *Portability, compatibility and supportability*

Portability, compatibility, supportability are three key aspects of containers [64]. Portability is related to the fact that a container image can be consumed by almost any container engine, like Docker and Podman. Compatibility concerns the content inside the container image and it extends to the processor architecture, the operating system, and its version. For example, running a RHEL 8 container image on a RHEL 4 container host or a Windows container image on a Fedora container host or an x86 container image binaries on a POWER container host can give problems even though the image is portable. Containers do not offer a compatibility guarantee and portability does not guarantee compatibility. Container images are collections of files, e.g. libraries and binaries, specific to the hardware architecture and operating system. When the container image is run, the binaries inside the image run as a process just as they would on a normal operating system. Thus, there must be compatibility between the container image and container host. For this reason a good practice for a developer is to use different base images for different container hosts. The third aspect is the supportability, that is "what vendors can provide, in terms of testing, patching, security, performance, and architecture, as well as ensuring that images and binaries are built in a way that they run correctly on a given set of container hosts (such as processor, operating system, and kernel version)" [64].

2.4.5.4 *Container orchestration*

A single application can be made up of hundreds or even thousands of containers [41]. Containers are a good way to bundle and run applications but, in a production environment, they should be managed by a system which ensure that there is no downtime. For example, if a container goes down, another container needs to start. Containers orchestration platforms are used to streamline processes like the installation, scaling, monitoring, and management of containerized applications. While Docker includes its own orchestration tool (called Docker Swarm), most developers choose Kubernetes instead, the most popular container orchestration platform. Kubernetes (also called "k8s" or "kube" by developers) is an open source platform for managing containerized workloads and services. It was created by Google and in 2014 it became open source. Kubernetes allows to create a cluster, i.e. a set of worker machines called nodes, which run containerized applications. Every cluster has at least one worker node. Kubernetes defines the set of running containers in a cluster as "pods", which are hosted by the worker node(s). Any pod can be composed of multiple, tightly coupled containers (an advanced use case) or just a single container (a more common use case). Containers in the same pod will share the same compute resources and they can communicate between each other. "The control plane manages the worker nodes and the pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability" [65]. Kubernetes uses replication controllers to horizontally scale an application as needed. This means that if a single pod becomes overloaded, Kubernetes can automatically replicate it and deploy it to the cluster.

In addition to supporting healthy functioning during periods of heavy load, Kubernetes pods are also often replicated continuously to provide failure resistance to the system, delivering distributed systems resiliently [66].

The market is monopolized by Kubernetes, which became de facto standard across many businesses because of its functionality, large and growing ecosystem, and portability between on premises, multiple public cloud, environments, and private clouds. Kubernetes is not the only player in the market, since that different competitors and alternatives are also available, e.g. Amazon ECS, Docker Swarm, Nomad, etc.

2.5 COMPUTING PARADIGMS AND THE CLOUD

Many solutions have been developed over the years to efficiently provide computational power and meet the different needs of users, industries, and research institutes. In the following, we briefly describe two different computing environments, High Throughput Computing (HTC) and HPC, and the most recent quantum computing. Lastly, the comparison and connection with cloud computing is presented.

2.5.1 *High Throughput Computing*

HTC is the use of many computing resources over long periods of time to accomplish a computational task, that is easily broken up into smaller, independent components. In HTC sequential jobs can be individually scheduled on many different computing resources. Grid computing has emerged as synonymous to HTC. Grid computing is a group of networked computers working together as a virtual supercomputer to perform large tasks. It is especially useful when different experts need to collaborate on a project but do not have the means to share data and computing resources in a single site [67]. By joining forces despite the geographical distance, the distributed teams are able to leverage their own resources that contribute to a bigger effort. Hence the concept of Virtual Organization (VO) as "dynamic collection of multiple organizations providing coordinated resource sharing" [68]. Virtualization refers to seamless integration of geographically distributed and heterogeneous systems, which allow services provided by the grid to be used in a transparent way. This means that users are not aware of the location of computing resources, and there is just one entry point to the grid system from the users' perspective. They just have to submit their service request at this node, and then it is up to the grid system to locate the available computing resources, needed to serve the users' requests. VOs are meant as multi-institutional entities, and the organizations that form a VO may have heterogeneous resources in terms of hardware, operating system and network bandwidth. Moreover, organizations can join or leave a VO per their requirements and convenience, so a VO is a dynamic entity.

A grid must optimize the resources under its disposal to achieve the maximum possible throughput. Resource management includes the submission of a job remotely, checking its status while it is in progress, and obtaining the output when it has finished the exe-

cution. The selection of resources used to run a job is based on a number of factors, e.g. the location of data. A job should be assigned to a resource located close to data instead of transferring large amounts of data over the network, which can lead to significant performance overheads. Hence the concept of data grid. A data grid can be defined as a grid for managing and sharing a large amount of distributed data, where several copies of data can be created in geographically distributed areas. This can be used to increase the file transfer speed and the overall computational efficiency. Indeed, if a user needs the data for any computational purpose, it can be accessed from the nearest machine hosting the data. Other key factors in the allocation of appropriate resources for a job are the resources availability and the scheduling policy of the grid. Let's suppose that the node executing the job crashes due to some reason. When a failure is detected the grid makes provision for automatic resubmission of jobs to other available resources.

Another important aspect related to the definition of HTC itself is that a user's request can be broken into multiple independent subtasks, each of which could be run on a different machine, then the results from each of these subtasks can be combined to produce the desired output.

2.5.2 *High Performance Computing*

HPC delivers a great number of computing resources to peak computing capability for a short period, allowing to perform computationally intensive operations across shared resources. An important aspect of HPC is related to the networking of computational nodes which uses extremely fast connections so that communicating data does not become a significant bottleneck to completing a large-scale computation. HPC can be run on a single node, but its real power comes from connecting multiple HPC nodes into a cluster or supercomputer with parallel data processing capabilities [69]. HPC clusters are used for simulations, AI inferencing, and data analytics that may not be feasible on a single system. HPC played a central role in academic research for decades, solving complex problems and spurring discoveries and innovations.

2.5.3 *Quantum computing*

Quantum computing is a type of computation in which operations can exploit the phenomena of quantum mechanics, such as superposition, interference, and entanglement [70]. Devices that perform quantum computations are known as quantum computers. There are several models of doing quantum computation, with the most widely used being quantum circuits and are based on the quantum bit, or "qubit", which is analogous to the bit in classical computation. A qubit can be in a 1 or 0 quantum state, or in a superposition of the 1 and 0 states. However, when it is measured it is always 0 or 1 and the probability of either outcome depends on the qubit's quantum state immediately prior to measurement. Quantum computers are built to solve very complex problems that no classical computer could solve in any feasible amount of time, a feat known as "quantum supremacy", that

Google claimed to have reached in 2019. Complex problems are problems with lots of variables interacting in complicated ways, e.g modeling the behavior of single atoms in a molecule because of all the different electrons interact with one another.

2.5.4 *Connection with cloud computing*

There are many differences between cloud and grid computing: the former follows a client-server architecture, where resources are used in a centralized pattern and the users pay for the use, while the latter follows a distributed architecture, where resources are used in a collaborative pattern and the users do not pay for use. Anyway, cloud-based grid computing is possible and it implies the use of computers in a public cloud service, or a hybrid of public cloud and internally owned computers, to collectively accomplish large tasks [71].

And what about HPC and cloud computing? A dedicated HPC cluster has a high-performance tuned setup, where the operating system run on bare-metal hardware and is customized to make full use of the available hardware. In the cloud, PaaS server farms usually run hypervisors as hosts for various types of guest VMs, and the overhead of VMs compared to bare-metal is of course a serious performance disadvantage [62]. Anyway today there are HPC cloud services that can support the most complex and challenging workloads. Throughput for HPC can be accelerated in the cloud, where on-demand availability of computing resources enables jobs to move forward instead of languishing in a queue. An additional advantage of HPC in the cloud is the ability to run workloads on HPC hardware preselected and configured by the cloud service provider [69].

In quantum computing the cloud is the way for researchers, engineers, and other experts to gain access to the prohibitively expensive quantum hardware at an affordable price, also enabling a quantum enthusiast with a laptop in his/her bedroom to test the power of quantum computers in a cloud computing environment.

2.6 DEVOPS

Nowadays the term DevOps is more and more common, especially in companies. It is the contraction of the words “Development” (Dev) and “Operations” (Ops), and it was introduced in 2009 by Patrick Debois. DevOps is a set of practices aimed at reducing the barriers between developers, who want to innovate and deliver faster, and operations teams, who must guarantee the stability of production systems and the quality of the system changes [72]. Before DevOps the two teams had quite different goals and incentives, which often conflicted with each other [73]. Developers tended to focus on shipping new features fast, whereas operations teams cared mostly to make services stable and reliable over the long term. In some cases security policies in place prevented software developers from even having access to the logs or metrics of their own applications running in production, who then had to ask permission from the operations team to debug the application and deploy any fixes. As cloud computing became more popular, the industry changed and the adop-

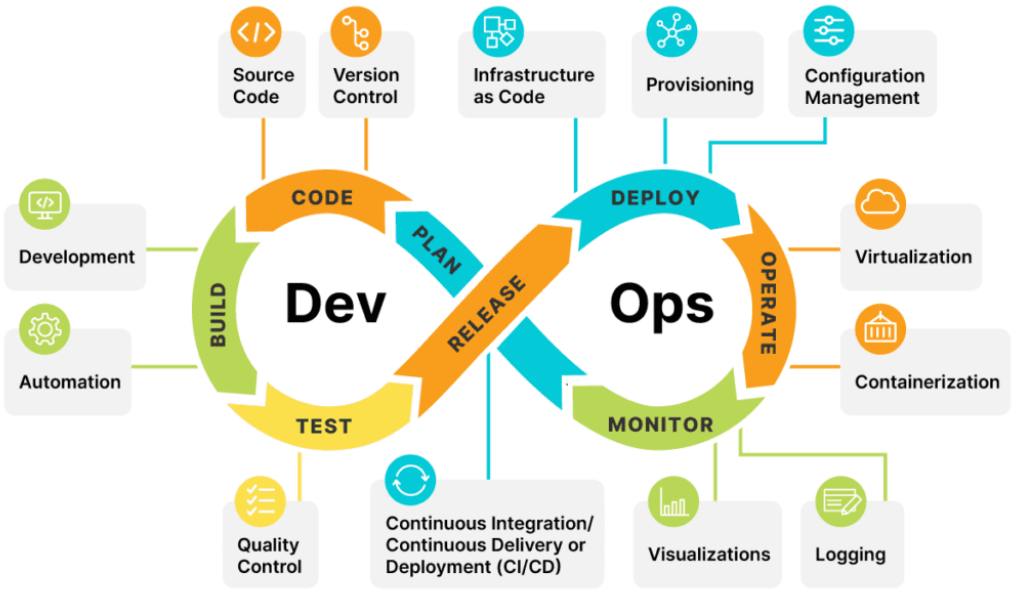


Fig. 2.5: Phases of the DevOps working methodology [74].

tion of DevOps allowed to produce a deployment of higher quality, saving money for the companies.

The DevOps movement is based on three axis.

- **Collaboration**. The idea is not to have specialized teams (one of developers, one of operations, one of testers, and so on) but to reach the multidisciplinarity where people have the same objective, i.e. deliver added value to the final product as quickly as possible.

- **Process**. To expect a rapid deployment, the teams must follow agile development processes with an iterative approach that allows for better functionality, quality, and rapid feedback. The DevOps process includes several phases, which are carried out cyclically and iteratively throughout the life of the project: planning and prioritizing functionalities, Continuous Integration (CI) with Continuous Delivery (CD) or Continuous Deployment, coding, testing, deployment, and continuous monitoring. See Fig 2.5 for a schematic representation of the DevOps phases.

- **Tools**. The choice of tools and products used by teams is an important phase in DevOps, and should avoid communication gaps between the members of the team.

In this scenario, developers need to exploit the monitoring tools used by operations teams to detect performance problems as soon as possible, whereas on the other hand operations team must automate the process of creating and updating the infrastructure integrating the code into a code manager. This process is called *Infrastructure as a Code* (IaC), and can only be done in collaboration with developers who know the infrastructure that is needed for applications. Therefore operations team must be integrated into processes and tools of the application release.

IaC consists in writing the code of the provisioning and configuration of resources that compose an infrastructure, helping in automatizing the deployment in a repeatable and consistent manner. Using IaC has several advantages: standardizing the infrastructure configuration reduces the risk of errors, the code that describes the infrastructure is versioned and controlled in a source code manager and it is integrated into CI/CD pipelines, deployments that make infrastructure changes are faster and more efficient, there is a better management/control and a reduction in infrastructure costs. IaC also brings benefits to a DevOps team by allowing Ops to be more efficient in terms of infrastructure improvement tasks rather than spending time on manual configuration. It gives Dev the possibility to upgrade their infrastructures and make changes without having to ask for more Ops resources. IaC also allows the creation of self-service, ephemeral environments that will give developers and testers more flexibility to test new features in isolation and independently of other environments. The languages and tools that are used to write the configuration of the infrastructure can be of different types, i.e. scripting (e.g. use Bash or PowerShell scripts with Azure CLI or Azure PowerShell), declarative (e.g. Ansible and Puppet), and programmatic (e.g. use TypeScript, Java, Python, or C# in Pulumi or Terraform CDK tools).

Now let's focus on other key steps in the DevOps process, i.e. CI, CD, and continuous deployment.

- **CI** is an automatic process that allows checking the completeness of the application code every time a team member makes a change. This verification must be done as fast as possible. To set up CI two elements are needed: a Source Code Manager (SCM) that will centralize the code of all the team members (e.g. Git), and an automatic build manager (CI server) that supports continuous integration (e.g. Jenkins, GitLab CI, GitHub Actions, and Travis CI). Each team member will work on the application code daily, iteratively, and incrementally, focusing attention on specific tasks/features that should be developed in different branches. Then the members archive or commit their code regularly, preferably with small commits that can be easily fixed in case of error and finally integrated into the code of the application. Integrating all the commits is the starting point of the CI process, which is executed by the CI server, and needs to be automated and triggered at each commit. The server takes the code, builds the application package, and performs unit tests. This CI process should run fast so that developers can obtain quick feedback on the integration of their code and not block the entire team.

- **CD** is the phase in which the application is automatically deployed in one or more non-production environments, also called staging environments. CD often starts with an application package prepared by CI, which is installed following a list of automated tasks. Unlike CI, the CD aims to test the entire application with all of its dependencies. This is very visible in applications composed of many microservices. Here CI will only test the microservices under development, whereas once deployed in a staging environment the entire application can be tested and validated, as well as the microservices that it is composed of. Nowadays it is very common to link CI to CD in an integration environment, where CI at the same time deploys in an

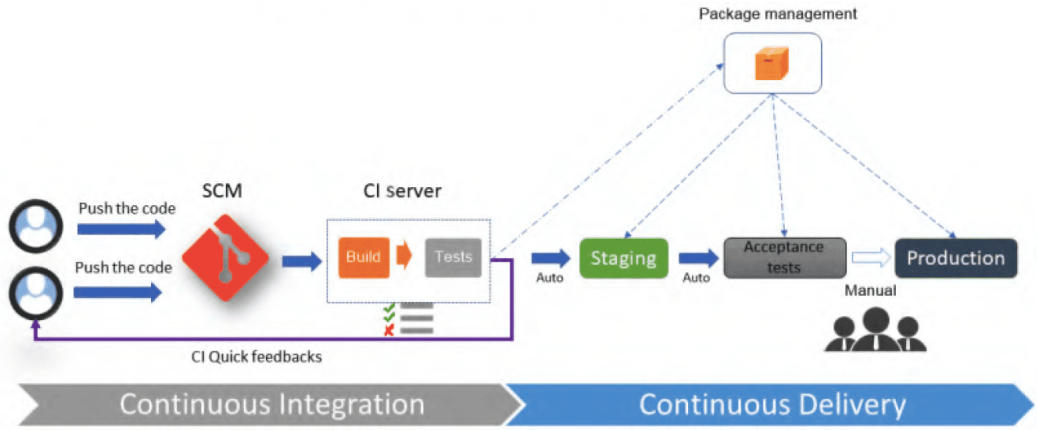


Fig. 2.6: Diagram showing the cyclical steps of CI/CD [72]. In CI the code is pushed into the SCM by the team members, whereas the build and test are executed by the CI server. Then follows the CD process as a continuation of CI. The package generated by CI is stored in a package manager, and afterward it is deployed in different environments.

environment. Here developers can execute not only unit tests, but also verify the application as a whole at each commit. In CD, deploying the application in each staging environment can be triggered automatically (following a successful execution in a previous environment), or manually (e.g. for sensitive environments). In a CD process, the deployment to the production environment, that is for the end user, is triggered manually by approved users. To better understand the cyclical steps of the CI/CD process, see Fig. 2.6.

- **Continuous deployment** is an extension of CD, where the entire CI/CD pipeline is automatized from the developer's commits to the deployment in production through all of the verification steps. It results in an automated end-to-end deployment. This practice is rarely implemented in enterprises since it requires a great variety of tests for the application.

2.6.1 *Source Code Manager*

In Sec. 2.6 we introduced the figure of SCM, which plays an important role in the code development inside a DevOps team. It provides a mechanism to keep track of file changes, enables multiple team members to work simultaneously on the same file, and provides UI- or web-based interfaces for developers to visually see the differences between two versions of the same file [75]. Another feature is the version control safeguard to prevent loss of work due to conflict overwriting. It works by tracking changes coming from each developer, identifying areas of conflict, and preventing overwrites. SCM will then communicate these points of conflict back to the developers so that they can safely review and address them [76]. Once SCM has started tracking all the changes in a project over time, a detailed historical record of the project life is created, that can be used to undo changes to the codebase and revert the codebase back to a previous point in time. Every change over

a project life time is archived by the SCM, which provides valuable record keeping for project release version notes. A clean and maintained history log can be used as release notes offering insight and transparency into the progress of a project and can be shared with end users. The use of an SCM reduces team communication overhead and increases release velocity. With SCM developers can work independently on separate branches of feature development, eventually merging them together.

By far, the SCM and version control system most widely used in the world is Git. Git is an open source project originally developed in 2005 by Linus Torvalds, the creator of the Linux operating system kernel. A staggering number of software projects rely on Git, including commercial as well as open source projects. Github, on the other hand, is the most famous cloud-based hosting service (which is currently owned by Microsoft) of source code that allows users to host repositories in the cloud, enabling Git-based workflows and CI tools. Other common alternatives to GitHub are Bitbucket and GitLab, while other SCM tools alternative to Git are Subversion and CVS.

The best practices in using a SCM can be summarized as the following.

- **Commit often**. Commits are snapshots where the codebase can be reverted to if needed. They are easy to make and should be made frequently to capture small updates in a codebase, giving many opportunities to revert or undo the work.
- **Make sure you are working on the latest version**. SCM enables rapid updates from multiple developers and it is easy to have a local copy of the codebase fall behind the global copy. A good practice is to git pull or fetch (taking the Git nomenclature as an example) the latest code before making updates. This will help to avoid conflicts at merge time.
- **Make detailed notes**. Each commit has a corresponding log entry where developers add a descriptive and explanatory log message, which should explain the "why" and "what" of the commits content. These log messages become the history of the project development and leave a trail for future contributors.
- **Review changes before committing**. SCM offers a "staging area" that can be used to collect a group of edits to manage and review before creating the commit snapshot. In this way, the staging area provides a buffer area to help refine the contents of the commit.
- **Use branches**. Branching is a powerful SCM mechanism that allows users to create and work on a separate line of development, which generally contains different product features. When the development on a branch is complete, it is then merged into the main line of development.

2.7 CLOUD SERVICES

One of the most important benefits businesses gain from cloud computing is the ability to modularize application services so that they can be linked together to create a cohesive

environment, and they can be developed and deployed virtually anywhere [41]. The original concept developed in the early 2000s was the Service Oriented Architecture (SOA). It was intended to create small modular services that could work independently of each other but at the same time could be integrated together, rather than build monolithic applications full of dependencies. In this way, each service can be developed and deployed by small independent teams. SOA was born as a way to make on-premises applications development and deployment more agile.

This way of building services required a new approach to software development and deployment where the code needs to be designed as lightweight code. Once cloud services began to be used more and more, it became clear that this new model of computing needed an even more comprehensive and sophisticated way of developing, deploying, and managing services. The movement toward a service orientation and containerization has brought the development of new management frameworks that simplify and streamline the service and other workloads. This requires a more sophisticated approach to managing APIs as the fundamental means that defines services and provides access to applications [77]. Another key requirement is that application services should be designed for the cloud, hence the term cloud native, where services take advantage of the underlying distributed nature of the cloud to improve agility, flexibility, and modularity.

2.7.1 *Cloud native applications*

Cloud native applications are designed to be modular, distributed, deployed, and managed in an automated way, characteristics that require technologies that go beyond what is typical for the development of traditional software. Even though a monolithic application can be moved to the cloud, there is no gain in terms of costs and flexibility. The best way to gain value from the cloud is to think modular, which is why microservices are so important for cloud computing. The future will be towards a cloud native approach, where cloud applications are meant as a collection of multiple, independent microservices that will help organizations to fully exploit the advantages of the cloud. Microservices are applications with code that is independent of each other and of the underlying developing platform, and are designed to support one discrete, bounded piece of application functionality. Although these microservices are independent, they can be connected in a coordinated fashion to provide all the functionality the application is intended to deliver. Each microservice runs a unique process and it is equipped with well-defined and standardized APIs. All these services are defined in a catalog so that developers can more easily locate the right service and understand the usage rules. The usage of microservices simplifies the deployment of additional instances, the application of changes when they are needed, and the development of the application itself. Microservices are designed to be packaged within containers, and orchestration platforms are often used to streamline processes like the installation, scaling, monitoring, and management of containerized applications.

The idea of cloud native was introduced by the Cloud Native Computing Foundation (CNCF), an organization founded in 2015 whose members are some of the most import-

ant companies in the public and private cloud market, e.g. Google, Twitter, Huawei, Intel, Cisco, IBM, and Docker. A software framework is defined cloud native when it is designed with microservices, containers, dynamic orchestration, and continuous delivery of software. Every part of a cloud native application is housed within its own container and dynamically orchestrated with other containers to optimize how resources are used. Traditionally, developers used VMs to create cloud services, but VMs sit on a layer of software (including the operating system, middleware, and tools) which makes them more complex and slows down the process of continuous integration and rapid development of applications. With the growth of cloud native applications, the use of containers is increasing dramatically. Containers are much smaller than VMs, and can be spun up more quickly. Because a cloud native environment is based on containerization, it is not physically tied to a specific hardware or operating system. Therefore cloud native applications are designed to work in different cloud environments, and to be moved more easily between different cloud environments and between on-premises and the cloud, even because they carry with them all of their dependencies.

A cloud native application exploits the benefits of cloud technology, such as the distributed and scalable architecture the cloud platform provides, in order to offer the highest levels of performance, flexibility, scalability, and reusability. Cloud native applications are built to run on hardware that is modular and automated, enabling them to become both resilient and predictable. Traditional applications do not offer those benefits. The technologies used to create and deploy cloud native applications provide an abstraction layer away from the underlying software and hardware infrastructures so that the developers can focus exclusively on building their applications without the need to deal with dependencies of the underlying infrastructure.

DevOps methodologies are a necessary component of cloud native applications and the adoption of this approach involves the integration of processes, tools, and developers. DevOps creates an environment where software can be written, tested, and released quickly, enabling the CI/CD for a cloud native application as the software modules created can be released continuously and in an automated fashion.

In a highly distributed environment consisting of microservices, the ability to communicate between services becomes critical if the benefits of the cloud are to be fully realized. APIs serve this purpose as they are the mode of communication among microservices and containers. Moreover, at the IaaS level APIs are used to provide control and distribution mechanisms for resources (like provisioning), whereas at the SaaS level APIs furnish the ability to connect applications with the underlying infrastructure and eventually the cloud resources.

2.8 SECURITY

Cloud computing is one of the greatest innovations of modern computing, but with all its many benefits come several responsibilities. One main responsibility is the management of security [78]. Cloud security is a set of procedures and technology designed to address

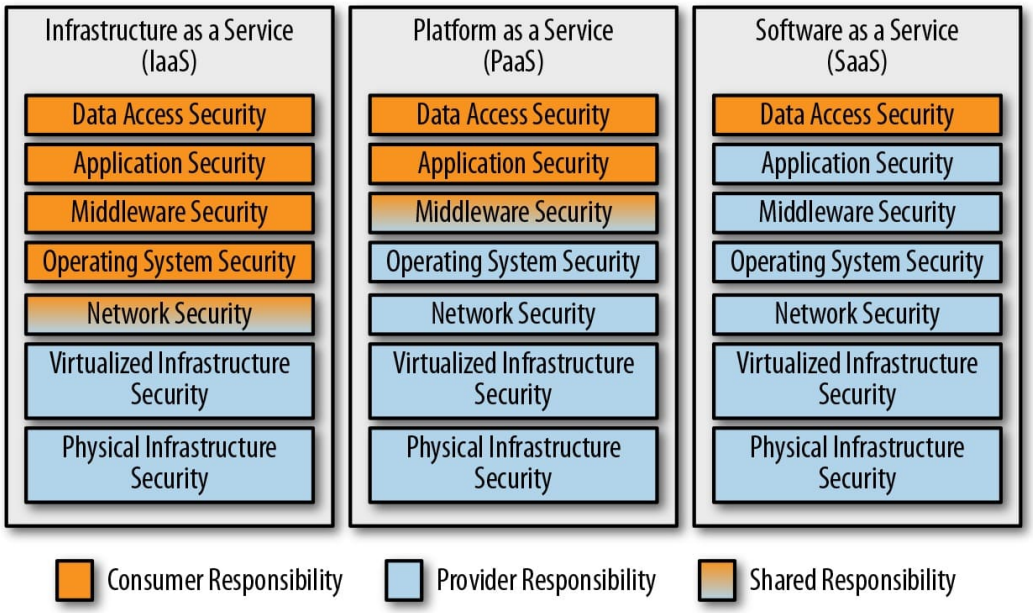


Fig. 2.7: Cloud shared responsibility model [80].

external and internal threats to business security. Organizations need cloud security as they move toward their digital transformation strategy with the integration of cloud-based tools and services as part of their infrastructure [79]. In modern-day enterprises, there has been a growing transition to cloud-based environments and to IaaS, Paas, or SaaS computing models. In this scenario, the understanding of security requirements for keeping data safe has become critical. While third-party cloud computing providers may take on the management of the underlying infrastructure by following best security practices and taking active steps, organizations need to make their own considerations when protecting data, applications, and workloads running on the cloud.

The most basic security question a user of a cloud service must answer is: "What aspects of security am I responsible for?". In an on-premises environment (and in a private cloud), the organization that owns the data center and the resources is the entire responsible for everything. But when you move from an on-premises environment to a cloud environment there is a more complicated shared responsibility model for security (see Fig. 2.7).

Adopting IaaS, PaaS, and SaaS solutions, the cloud provider has complete responsibility for physical infrastructure security, which often involves controls beyond what many companies can reasonably do on-premises, e.g. by integrating biometric access [80]. Likewise, if the provider offers virtualized environments, the virtualized infrastructure security controls keeping the user's virtual environment separate from other virtual environments are responsibility of the provider. Network security is shown as a shared responsibility in the IaaS section of Fig 2.7. There are several layers of networking, and the responsibility for each lies with a different party. The cloud provider has its own network which is its responsibility, but usually there is a virtual network on top, and it is the customer's responsibility to add reasonable security zones and put in the proper rules for access between them. Many implementations use also firewalls, and transport encryption which are the customer's responsibility. Operating system security is customer's responsibility

for IaaS, and it is the provider's responsibility for PaaS and SaaS. Middleware is used in this context as a generic term referred to the software in the middle between the operating system and the application, e.g. databases, application servers, or queuing systems, that are not directly used by end users but are used to develop solutions for end users. Using PaaS, the middleware security is often a shared responsibility: the provider might keep the software up to date but the customer retains the responsibility for security-relevant settings such as encryption. The application layer is what the end user actually uses, and only for the SaaS case vulnerabilities at this layer (such as cross-site scripting or SQL injection) are the provider's responsibility. Even if all of the other layers have high levels of security, a vulnerability at the application security layer can easily expose all of the underlying information. Lastly, data access security is always responsibility of the customer.

The root cause of many security incidents is an assumption that the cloud provider is handling something when it turns out nobody was handling it. Many real-world examples of security incidents stemmed from a poor understanding of the shared responsibility model. In addition to this aspect, there are other elements that represent a challenge in cloud security. For example, it is easy to lose track of how users' data is accessed and by whom since many cloud services are accessed outside of corporate networks and eventually through third parties. Then, as public cloud environments house multiple client infrastructures under the same umbrella, it is possible that users' hosted services can get compromised by malicious attackers as collateral damage when they target other businesses. Moreover, misconfigurations like leaving default administrative passwords in place or not creating appropriate privacy settings are important issues for security, representing the cause of 86% of breached records in 2019.

2.8.1 *Tools for securing the cloud*

Modern cloud vendors offer an increasing number of tools enabling customers to secure their cloud systems. The aim is to guarantee confidentiality, i.e. keep something a secret, integrity, i.e. keep information intact, and accessibility, i.e keep the system available. In the following, we cover the major categories of tools commonly used to secure the cloud and not only [35].

2.8.1.1 *Identity and Access Management*

IAM is a tool which purpose is to have a central directory of identities or users of an organization, and a record of what they are allowed to do in different systems. This area is already standardized around a number of different protocols, and ensures that there are functions like Single Sign-On (SSO) and password reset. The area contains information about users, such as credentials, where they are working, their email and telephone, and defines which users are "valid". To gain access to a system, authentication needs to take place verifying that users are who they say they are and that they are active. There are multiple ways to authenticate, from the simplest username and password to the more advanced two-factor and biometric authentication. Access management systems handle user

authentication. Identity management systems handle the assignment and maintenance of roles for users in different systems, i.e. what that user is authorized to do inside a system.

2.8.1.2 *Traffic management*

A large part of IT solutions are made up of systems or components communicating with other systems or components, and if the flow of information is not strictly controlled, it will be easy to breach the confidentiality and integrity of solutions. For this reason, it is important having a traffic management tool which main function is to specify rules for how communication is allowed to take place within a virtual network, and between it and the Internet. One of the most known tools is the firewall, which specifies not only which IP addresses and ports are accessible by whom, but also what type of traffic is acceptable with the aim to control and detect unwanted traffic.

2.8.1.3 *Encryption*

The purpose of encryption is to keep data confidential using a cryptographic key. There are different types of encryption with different strengths, and the most common are encryption in transit and at rest. The purpose of the former is to prevent wiretapping: even if someone is listening to the transmission, they will not be able to understand the nature of the communication if it is encrypted in transit. The latter aims at encrypting data stored on disk, in a database, on a filesystem, or on another type of storage medium, to avoid unauthorized access to the stored data. The encryption can be done manually, but many services encrypt data simply by turning on a flag when configuring the database, for example. Then there are key-management tools used to store keys and rotate them with a certain frequency.

Now let's focus on encryption in transit. SSL (Secure Sockets Layer) encryption, and its more modern and secure replacement, the TLS encryption, protect the confidentiality and integrity of data sent over the Internet or a computer network. Although SSL was deprecated in 2015 and replaced by TLS, the term "SSL" is still commonly used for this technology. Recent versions of all major web browsers currently support TLS, and it is increasingly common for web servers to support TLS by default. TLS uses a combination of symmetric and asymmetric cryptography, as this provides a good compromise between performance and security when transmitting data securely. Asymmetric encryption is used to establish a secure session between a client and a server, whereas symmetric encryption is used to exchange data within the secured session. To use SSL/TLS encryption a website must have an SSL/TLS certificate. Once installed, the certificate enables the client and server to securely negotiate the level of encryption in the following steps: the client contacts the server using a secure URL (starting with "https://"), the server sends the client its certificate and public key, the client verifies this with a Trusted Root Certification Authority to ensure the certificate is legitimate, the client and server negotiate the strongest type of encryption that each can support, the client encrypts a session (secret) key with the server public key, and sends it back to the server, the server decrypts the client communication with its private key and the session is established, the session key (symmetric encryption)

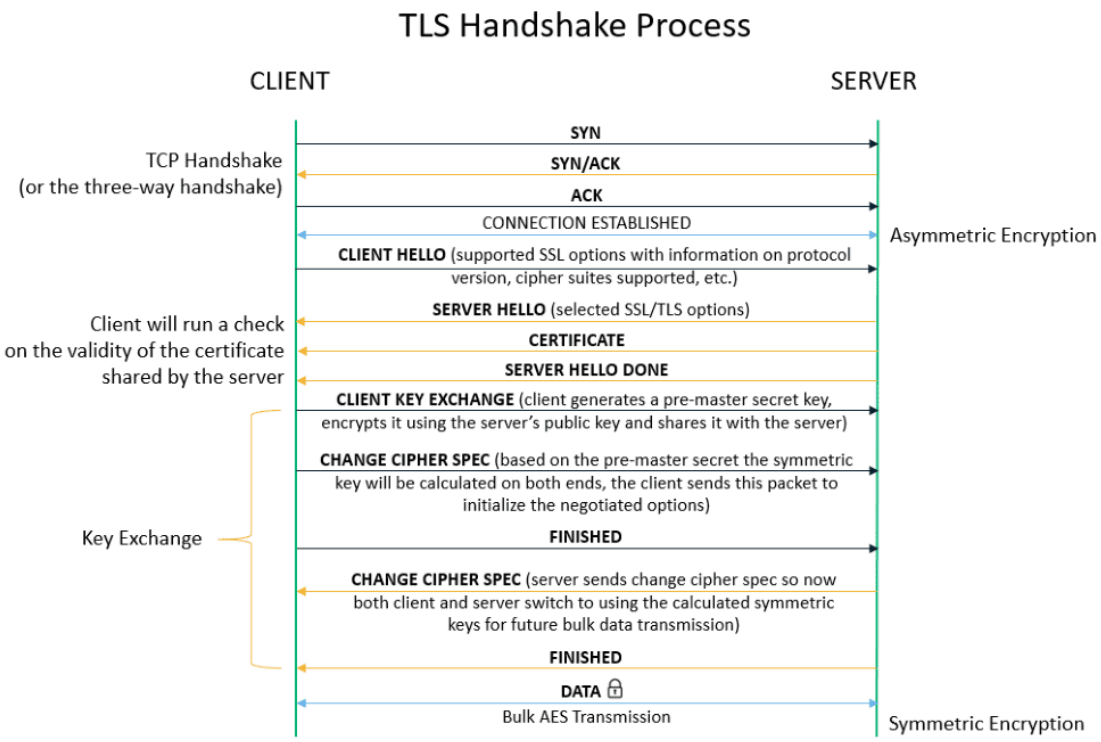


Fig. 2.8: TLS handshake [82].

is subsequently used to encrypt and decrypt data transmitted between the client and server [81]. The process described so far is called TLS handshake and can be visualized in Fig. 2.8.

The most common public key (asymmetric) encryption used in SSL/TLS is the RSA (others are e.g. DSA and ECC), while AES is the most commonly used symmetric encryption algorithm. Even though asymmetric encryption provides more protection to the keys, it is slower compared to symmetric encryption. For this reason, asymmetric encryption is used to exchange the secret key, which can be used to establish symmetric encryption for fast data transfer, making faster the encryption and decryption of data. Symmetric encryption is commonly used to encrypt and decrypt local data, and data at rest.

The SSL/TLS certificate of the server is an X.509 digital certificate issued by a trusted third party known as a Certificate Authority (CA) which asserts the authenticity of the public key [83]. It may happen that a server uses a self-signed certificate which needs to be explicitly trusted by the client (when an untrusted certificate is encountered, the browser should display a warning). A CA acts as a trusted third party that provides clients (known as relying parties) assurance they are connecting to a server managed by a validated entity.

To have more detailed information on how TLS works, see [84].

2.8.1.4 *Security assessment and protection*

One of the benefits of the cloud is that vendors develop a good understanding of common threats and recognize emerging threats faster: they know exactly how security works, much more than any single company would, and provide useful products to the customers.

One type of such products looks at the configuration of the infrastructure in the customers' account and identifies potential issues, e.g. it will look for open ports to the Internet, missing security patches, and elevated privileges. Other security assessment tools focus on data: they can scan for sensitive data that may be out of place in databases or file shares without proper access control. Lastly, there are threat protection products.

2.8.2 *Good security practices for the cloud*

Tools are not sufficient alone to secure the cloud and it is always useful to adopt good security practices. In the following, we provide a list of some of practices to keep in mind in general, but particularly in the cloud.

- **Manage data securely**. The first step of managing data securely is to make an inventory of data you have. A data catalog is a good way to gather insights into available data. Moreover, this will help to identify sensitive data that must be handled satisfying the regulatory requirements, provided for example by the EU General Data Protection Regulation (GDPR) or the US Health Insurance Portability and Accountability Act (HIPAA).

- **Minimize access to system resources**. Rather than allowing unrestricted access to a system resource, a good way to lower risk is to limit access to precisely what is necessary for the user.

- **Implement the least privilege**. A user should only have the required set of permissions to perform the actions for which he/she is authorized, and no more. Permissions can be added as needed, and should be revoked when no longer used.

- **Segregate duties**. Segregation of duties is mandatory in certain industries, and in general it is a good practice. In the context of system development, it means that people developing the systems must be different from those who administer them in production, and this for example allows to limit exposure to sensitive data. Segregation is also used to separate tasks of developers from those testing the finished systems, as the latter are more efficient at discovering defects in the design and function of a system than the developers who created it.

- **Backup data**. Backing up data is a way to make sure that system resources are available even in case of a catastrophic failure.

- **Log and review system access**. Logging does not just allow having a trace of what went wrong when a system crashes. Indeed, access logs can also be used to detect suspicious patterns and proactively counteract, and determine the extent of a breach in order to take efforts to mitigate it. Thus reviewing logs is important but it can be a daunting task. For this, there are solutions that automate the process. Several breaches today are so sophisticated that it can take months before they are discovered, but a proper log review can help for early detection. Then there is the forensic use of

logs when investigations are carried out after a breach or abuse has been discovered or suspected. Moreover, it is important to make sure logs are not altered by the criminal, otherwise their utility greatly diminishes.

- **Maintain systems**. The threat landscape changes continually and this implies that solutions need to be maintained and to keep evolving, especially in the cloud where Internet connectivity makes everything more exposed. A traditional form of maintenance consists of continuous upgrades and patches aimed at addressing identified vulnerabilities.

- **Move towards DevSecOps**. Historically, security practices often have been introduced at the end of the development lifecycle. However, with the rise of more sophisticated cybersecurity attacks and with development teams moving to shorter and more frequent iterations on applications, DevSecOps is now becoming a go-to practice for ensuring application security. DevSecOps is the seamless integration of security testing and protection during the software development and deployment lifecycle. For example, developers can run security tests in the development stage or in production phase in near-real time so they can immediately discover all instances of a vulnerability soon after the vulnerability is announced.

2.8.3 *The life of an identity*

We have already introduced the concept of IAM in Sec. 2.8.1 as a tool used to protect online resources from unauthorized access, playing an important part in a comprehensive security model [85]. It is a set of services supporting the creation, modification, and removal of identities and accounts, as well as the authentication and authorization needed to access resources. In this section, we want to deepen all these aspects related to the life of an identity.

The concepts of identity, identifier, and account are closely related but slightly different. The term "identifier" is used to refer to a single attribute whose purpose is to uniquely identify a person or entity within a specific context, e.g. common identifiers used for people are the name, age, address, email address, and passport number. The term "identity" is used to refer to a collection of attributes related to a specific person or entity in a particular context. These attributes may be used for the authentication and authorization phase as well as to convey information about the identity to applications. A given person may have more than one identity, just as a person might take on different characters in different social contexts. The term "account" is a local construct within a given application or application suite that is used to perform actions within that context, and that has one or more identities associated with it. To summarize, a person logs in to use an account that has various identity attributes associated with it and which enables them to perform actions within a system.

The life of an identity is characterized by a series of events, shown in Fig. 2.9, that we describe in the following.

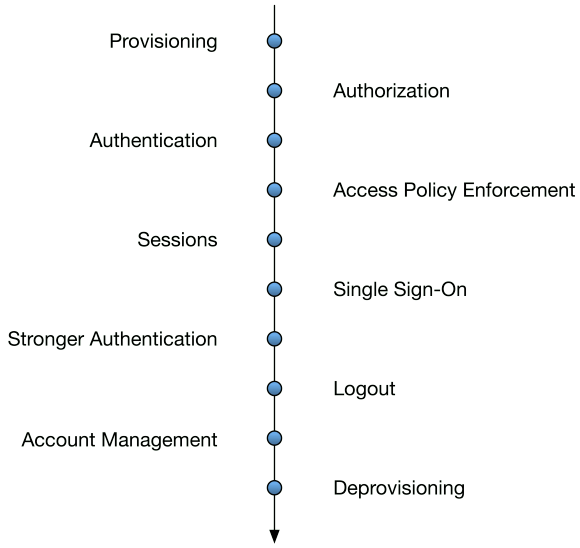


Fig. 2.9: Events in the life of an identity.

- **Provisioning**. The first step in the life of an identity is its creation. The act of creating an account and associating identity profile attributes with the account is often referred to as provisioning. Provisioning can be done by having users register or by leveraging an external identity service. Let's take as an example a user named Ginevra who wants to use some online banking services. The first step is to create an online account at her bank by filling out an account registration form with identity information, including username, password, her name, home address, phone number, and email address. This data will be used to provision an online account at the bank associated with Ginevra's personal identity. Ginevra could create multiple online accounts at her bank for different identities, for example she can create a second identity as a small business owner and use it for a second online account.

- **Authorization**. We use the term authorization to mean granting privileges that govern what an account is allowed to do. When Ginevra creates her online account, the bank authorizes her account to access the application to view her financial situation, whereas she is not authorized to view account information for other customers of the bank. Typically the authorization for an account is done at the time an account is created and can be updated over time.

- **Authentication**. To access an online content that is not publicly available a user needs to authenticate, and this implies the use of credentials. The credentials may involve something the user knows (i.e. a password), something the user has (e.g. a numeric code generated from a previously registered device, such as a mobile phone), and/or something the user is (i.e. biometric information, such as a fingerprint). After Ginevra establishes her online identity and account at the bank, she can access the bank online services, e.g. using the username and password established during the registration step. The username indicates the account she wants to use, while the knowledge of the password demonstrates her right to use that account.

- **Access Policy Enforcement**. Whereas the authorization specifies what a user or entity is allowed to do, the access policy enforcement checks that the user's requested actions are allowed by the privileges he/she has been authorized to use. If Ginevra attempts to access the stock trading services from the bank online application but she does not have the permissions, this would be denied as she is not authorized to access those services, and the application might display a message indicating she is not allowed to view that service, possibly with information on how to sign up for it.

- **Sessions**. Once a user has been authenticated and authorized, he/she can perform various actions within an application. Anyway, typically Web applications only allow a user to remain active for a limited period of time before requiring the user to authenticate again. This is done by creating a session for the user, which tracks information such as whether the user has been authenticated and when the authentication occurred, which enables an application to know when the user should be prompted to re-authenticate. The length of time a user can remain active before re-authentication is known as session limit or session timeout, and it typically varies by the sensitivity of the data in the application. Session limit help to protect against users who walk away from their screen without logging off, and provides a means of checking that he/she is still the legitimate user at the keyboard.

- **Single Sign-On**. After a user accesses an application, he/she may wish to do something else involving another application. SSO is the ability to log in once and then access additional applications or protected resources with the same authentication requirements without the need to reenter the credentials. SSO is possible when a set of applications delegates the authentication to the same entity. An authenticated session used to access multiple resources via SSO is commonly called an SSO session. The term "federated identity" is commonly used in SSO terminology to mean that an identity is trusted across multiple IT systems or even organizations. When Ginevra accesses her bank web site, SSO would provide an easy access to multiple banking services. If Ginevra signed up for the investment newsletter service at her bank, she could log in to access first the bank online application to view her account balance, and then access the investment newsletter without having to sign in again.

- **Stronger Authentication**. Forms of authentication, such as with username and password, are considered relatively weak because they involve a single factor (the password) which can be captured and easily used by others. Stronger forms of authentication, such as step-up and multi-factor authentication, involve other factors, e.g. something the user has and/or something the user is. Authentication that requires multiple factors is known as multi-factor authentication: typically it involves a password as well as the possession of a device, such as a laptop or mobile phone, or possibly a biometric factor, such as a fingerprint or voiceprint. Instead, step-up authentication elevates an existing authentication session to a higher level of assurance by authenticating with a stronger form of authentication. For example, Ginevra might initially log in with a username and password to view her account balance on the

bank web site. If at a later time she attempts to transfer a large amount of money out of her account, she might enter a stronger authentication factor, such as a special one-time use code generated by an application on her phone. This elevates her session to a higher level of authentication assurance, providing a higher degree of confidence that the user who requests access or performs a transaction is the legitimate account owner.

- **Logout**. When a user is done with an application, he/she terminates the session by logging out and this terminates the user's application session. If the user returns to the application, he/she has to authenticate again before being granted access. In situations where SSO is used, there may be multiple sessions to terminate and it is a design decision which sessions should be terminated when the user logs out of the application.

- **Account Management and Recovery**. During the course of life of an identity, various attributes of the user profile for the identity could change. For example, a user may need to update the email address or phone number, may periodically change the password or mobile device used in the authentication process. Account management consists of features which allow users and administrators to view and update user profile attributes associated with an identity. It may happen that a user forgets his/her password or lose the device that is required for the authentication process, and therefore the user needs to establish new credentials. Account recovery is a mechanism to validate a user is the legitimate owner of an account through secondary means and then allow the user to create new credentials, e.g. an account recovery link can be sent to the user's email that will enable him/her to reset the credentials.

- **Deprovisioning**. There may come a time when a user has to close his/her account, which means that the user's account and associated identity information must be deprovisioned so that they can no longer be used. Deprovisioning can take the form of completely deleting the account and the associated identity information, or simply disabling the account.

2.8.4 *OAuth 2.0 and API authorization*

Modern applications are often designed around APIs, which provide access to valuable data or services. Typically in the past, a user often had to share his/her credentials with the application to enable an API call on his/her behalf. This gave the application an unnecessary amount of access and the responsibility of safeguarding the credential. Let's see how the OAuth 2.0 protocol/framework changed this scenario providing a better solution for authorizing applications to call APIs.

The OAuth 2.0 authorization framework was published in 2012 and was designed to enable an application to obtain authorization to call third-party APIs. Using OAuth 2.0 an application can obtain a user's consent to call an API on his/her behalf without the need of his/her credentials. Moreover, an application can obtain authorization to call an API on

its own behalf if it owns the content to be accessed. OAuth 2.0 introduces four actors in a typical OAuth flow.

- **Resource owner**. A user, or other entity, who owns protected resources at the resource server.
- **Resource server**. A service, with an API, storing protected resources to be accessed by an application.
- **Client**. An application which has to access resources at the resource server on the resource owner's behalf or on its own behalf. In the following, the term application is used interchangeably with the term client.
- **Authorization server**. A service trusted by the resource server that authorizes the client to call the resource server. It authenticates the client and requests consent from the resource owner when the client makes requests on the resource owner's behalf. With OAuth 2.0, the authorization server and resource server may be operated by the same entity.

With OAuth 2.0, when a client calls an API on behalf of a user, it sends an authorization request to an authorization server. The authorization server handles the access request for the API and returns a token that can be used by the client to access the API. In the authorization request, the client provides an indication, known as `scope`, of what it wants from the API. The authorization server evaluates the request and, in case the client is authorized, returns a token to the client in which the `scope` refers to the access that is actually granted. Moreover, if the client asks for a content owned by the user, the authorization server authenticates the user and then asks him/her to give the consent for the client to access the requested data. This authentication step ensures that the user giving the consent is the owner of the resource being accessed. If the user agrees to the requested access, the client receives a token to call the API on behalf of the user. This token is called access token and it allows the client to make API requests within the `scope` of what the user authorized when he/she gave the consent for the request. This approach eliminates the need for the user to share credentials with the application and gives the user more control over what the application can access. For a schematic representation of the authorization process with OAuth 2.0, see Fig. 2.10

The OAuth 2.0 framework defines four methods by which a client obtains authorization to call an API. To represent the authorization each method uses a different type of credential, known as authorization grant, and the choice depends on the use case and type of application. The four authorization grant types are: authorization code grant, implicit grant, resource owner password credentials grant, and client credentials grant. For a full description of how the four authorization grant types work, see [85].

OAuth 2.0 defines two security tokens and an intermediary authorization code.

- **Access token**. It is a token that represents the authorization obtained by the client to call an API and it is used by the client to access the API. Access tokens have an expiration.

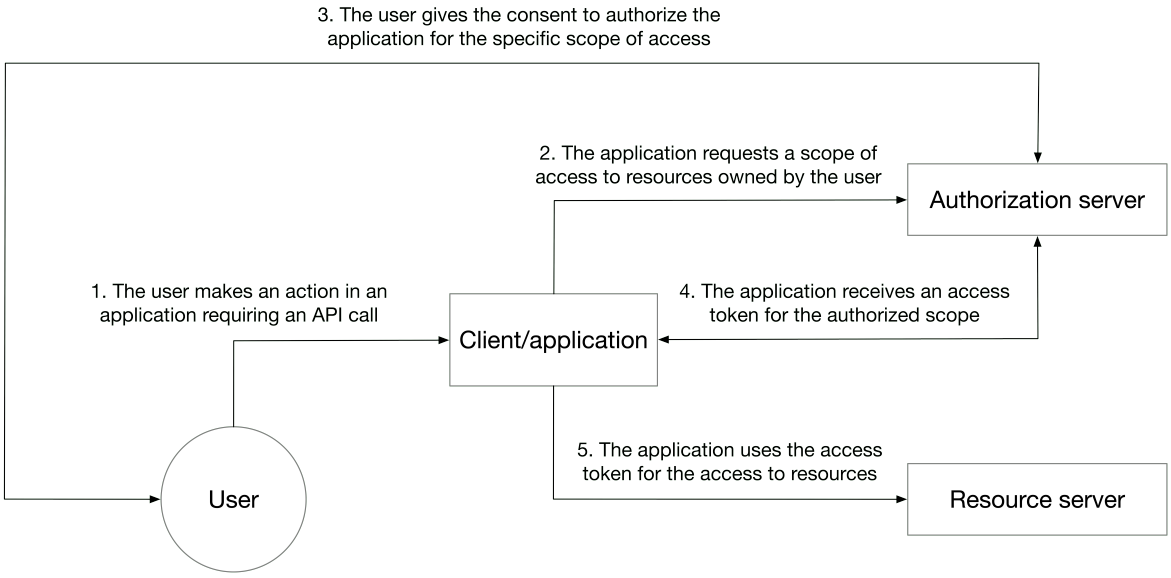


Fig. 2.10: Authorization process with OAuth 2.0.

- **Refresh token**. It is an optional token that provides a convenient way for clients to obtain a new access token when the previous one has expired, minimizing the risk if an access token is compromised. The handling of refresh tokens may vary across individual authorization servers: some of them release refresh tokens automatically, and others expect an application that explicitly requests a refresh token.
- **Authorization code**. It is an intermediary code returned to an application and used once to obtain an access token, and optionally a refresh token.

OAuth 2.0 has two main token profiles: OAuth 2.0 Bearer Token Profile and OAuth 2.0 MAC Token Profile. The former is the most popular and almost all OAuth 2.0 deployments today are based on it. As the name suggests, anyone who "bears" the token can use it, so it is essential not to lose it. Bearer tokens should always be used over TLS to avoid losing them in transit. Once the bearer access token is obtained from the authorization server, the client can use it to talk to the resource server in three ways. The most popular way is to include the access token in the HTTP authorization header, i.e. adding `Authorization: Bearer <access_token>`. Otherwise, the access token can be included as a query parameter or can be sent as a form-encoded body parameter. A bearer token can be a reference token or a self-contained token: the former is an arbitrary string, and the latter is a JSON Web Token (JWT). We describe the format of JWTs in depth in Sec. 2.8.4.1. When the access token is a reference token, the resource server needs to validate the token by talking to the authorization server (or the token issuer). Whereas when the access token is a JWT, the resource server can validate the token by itself by verifying the signature of the JWT.

The OAuth 2.0 access token is only intended for API access, for authorizing API calls, and not for passing information about the authentication event or the user (at least in absence of any proprietary additions to the base protocol that some providers have imple-

mented). OpenID Connect (OIDC), described in Sec. 2.8.5, can be used to authenticate a user to an application.

2.8.4.1 *JSON Web Token*

JWT is an open standard that defines a self-contained and compact means to securely transmit information between parties as a JSON object [86]. Again, JWT is a standard [87], meaning that all JWTs are tokens but not vice versa. Using JWTs offers several advantages over other types of tokens, e.g. Simple Web Tokens (SWTs) and Security Assertion Markup Language (SAML) tokens.

- **JWTs are more compact**. JSON is less verbose than XML, so after it is encoded, a JWT is smaller than a SAML token (which is based on the XML format).
- **JWTs are more secure**. JWTs can use an X.509 certificate (containing a public/private key pair) for signing, or they can be symmetrically signed by a shared secret using the HMAC algorithm. Thus the information contained within the JSON object can be verified and trusted as it is digitally signed.
- **JWTs are more common**. JSON parsers are common in most programming languages as they map directly to objects.
- **JWTs are easier to process**. JWTs are used at the Internet scale and are easier to process on users' devices, especially mobile.

JWTs can be used for various purposes.

- **Authentication**. When a user successfully logs in using his/her credentials, an ID token is returned, which is always a JWT according to the OIDC specs.
- **Authorization**. When an application requests to access services or resources (e.g. APIs) on behalf of a user, in every request it must pass an access token, which can be in the form of a JWT. SSO widely uses JWT due to the small overhead of the format, and its ability to be easily used across different domains.
- **Information Exchange**. JWTs are a good way to securely convey information between parties as they can be signed, which means you can be confident that the senders are who they say they are, and the structure of a JWT allows you to verify that the content has not been tampered with.

There are some common best practices for a proper usage of tokens, and therefore also of JWTs: keep them secret and safe, do not add sensitive data to the payload, give tokens an expiration, send them over HTTPS connections, use the stored token for future calls until it expires rather than requesting a new token.

JWTs consist of three concatenated Base64URL-encoded strings, separated by dots: the header, containing metadata about the type of token and the cryptographic algorithms used to protect the token content, the payload, as a set of statements (called claims) about

the entity (typically the user) and the permissions granted, the signature, used to validate that the token is trustworthy and has not been tampered with.

The JWTs claims are pieces of information asserted about a subject and appear as name/-value pair, where the name is always a string and the value can be any JSON value. Typically, when we talk about a claim in the context of a JWT, we mean the name (or key). There are two types of JWT claims. Custom claims, which are non-registered public or private claims. Registered claims, which are standard claims registered with the Internet Assigned Numbers Authority (IANA) and defined by the JWT specification. There are seven registered claims that are not mandatory, but are recommended in order to ensure interoperability with third-party or external applications. These are: `iss` (issuer), the issuer of the JWT; `sub` (subject), the subject of the JWT, i.e. the user; `aud` (audience), the recipient of the JWT; `exp` (expiration time), the time after which the JWT expires; `nbf` (not before time), the time before which the JWT must not be accepted for processing; `iat` (issued at time), the time at which the JWT was issued and can be used to determine the age of the JWT; `jti` (JWT ID), the unique identifier that can be used to prevent the JWT from being replayed, and allows a token to be used only once.

A useful tool to decode, verify and generate JWTs can be found here [88].

2.8.5 *OpenID Connect and user authentication*

As described in Sec. 2.8.4, OAuth 2.0 provides a framework for authorizing applications to call APIs, but it is not designed for authenticating users to applications. The OIDC protocol provides an identity service layer on top of OAuth 2.0 to delegate user authentication to an OAuth 2.0 authorization server, and return to the application claims about the authenticated user and authentication event in a standard format. [85]. OIDC is the third generation of OpenID, has become standard since 2014, and currently is the most popular Identity Federation protocol. Most of the applications developed in the last few years are supporting OIDC [84]. OIDC is very attractive, as it is much easier to use than SAML, and does not require the heavy XML handling that SAML does [89]. SAML is a mature technology dating back to 2005 and supports a wide range of identity functionality. It is an open standard for authentication (and, if required, authorization) which provides SSO access to web applications through identity federation. Anyway, now that OIDC and OAuth 2.0 exist, modern applications using APIs will benefit from implementing these newer protocols.

Now let's see how OIDC works. When a user accesses an application, it redirects the user's browser to an authorization server that implements OIDC to authenticate the user. OIDC calls such an authorization server (also known as identity provider) an OpenID Provider. After the authentication, the user's browser is redirected back to the application. Moreover, the application request that the claims about the authenticated user be returned in a security token called ID token. Alternatively, the application can request an OAuth 2.0 access token and then use it to call the UserInfo endpoint of the OpenID Provider to obtain the claims.

Since OIDC is a layer on top of OAuth 2.0, an application can use an OpenID Provider for both user authentication and API authorization. The provider can issue to the API an access token with custom claims containing information about the user, so that an access policy at the level of individual users can be enforced, ensuring that a user only accesses the articles he/she is allowed to view.

ID tokens are encoded in the JWT format. The payload section of the JWT contains claims about the user and the authentication event. The OIDC specification defines a set of claims for ID tokens (we have already seen and described some of them in Sec. 2.8.4.1) and additional standard claims can be added, e.g. user's name, email, and picture (see [90] for further details). Custom claims can be also defined and added by the OpenID Provider. A comparison between an example of ID token and access token is shown in Fig. 2.11, both in the form of JWT. We recall that ID tokens are always JWTs, whereas access tokens can be a string of any structure, such as a JWT.

OIDC defines three different flows by which an application can interact with an OpenID Provider to make an authentication request: authorization code flow, implicit flow, and hybrid flow. See [84] for more details on how they work.

Fig. 2.11: Comparison between an example of ID token and access token, both in the form of JWT [91]. We can notice that the payload in the ID token contains information about the user, such as the name, whereas the access token contains the `scope` field where it is defined which access is granted to the application.

3

HIGH ENERGY PHYSICS AT THE LARGE HADRON COLLIDER

Elementary particle physics or High Energy Physics (HEP) is the study of fundamental particles and forces that constitute matter and radiation. It is called "high energy" because experimentally very high energy probes are needed for such study. Progresses in HEP requires complex accelerators to create very high-energy collisions to uncover evidence of new particles and forces, as well as very intense lower-energy beams for precision measurements that can reveal subtle inconsistencies between experiments and theory.

The European Organization for Nuclear Research, known as CERN, is an European research organization that hosts the largest particle physics laboratory in the world, and it is based in a northwest suburb of Geneva on the Franco-Swiss border. The CERN convention was signed in 1953 by 12 founding states (including Italy) and entered into force on 29 September 1954. Today CERN has 23 member states. In 2021, the CERN community counted 16,200 members of which almost 2700 were scientific, technical, and administrative staff members, and more than 11,100 were users of 115 different nationalities from institutes in 77 countries who contributes to CERN's scientific mission [92].

The main function of CERN is to provide particle accelerators and other infrastructure needed for HEP research, and for this reason several experiments have been built at CERN through international collaborations. CERN hosts the largest and highest-energy particle collider in the world, called Large Hadron Collider (LHC). Collectively, LHC experiments operations produce about 200 PB of data each year that must be stored, processed, and analyzed. In order to allow physicists spread all over the world to have access to computing power and storage needed to conduct their research activities, CERN exploits a grid-based computer network infrastructure called Worldwide LHC Computing Grid (WLCG). Such distributed infrastructure and the computing software play a vital role in reconstructing and simulating collision data, and analyzing both in order to unlock the secrets hidden in the vast quantity of highly complex data produced by the experiments at CERN.

In this chapter, an overview of the Standard Model of particle physics is given, with additional insights on three physics analyses taken as use cases for the MLaaS4HEP framework (see Ch. 4). A description of the LHC experiments (and particularly CMS) is also provided. Afterwards, details on the CMS computing model (and its evolution towards HL-LHC) and the tools used for HEP analysis (in particular those related to data science) are provided. Lastly, Lastly, an overview of ML applications in HEP is given.

3.1 STANDARD MODEL AND TOP QUARK

The Standard Model [93, 94] explains how the basic building blocks of the matter interact, governed by fundamental forces [95]. The basic building blocks of matter are the irreducibly smallest detectable particles, called elementary particles. Elementary particles are classified as bosons and fermions based on their spin (integer and half-integer respectively), and are grouped based on the fundamental interactions they are subject to.

Fermions are 12, divided into 6 quarks and 6 leptons, in addition to other 12 that are the anti-particle counterparts. There are three generations of elementary particles, and each generation contains two types of leptons and two types of quarks. The lightest and most stable particles are in the first generation, whereas the heavier and less stable particles belong to the second, and third generations. All stable matter in the universe is made up of particles that belong to the first generation. The six quarks are: up (u), down (d), charm (c), strange (s), top (t) and bottom (b). Quarks come in three different "colors" and only mix in ways that form colorless objects. The six leptons are: electron (e), electron neutrino (ν_e), muon (μ), muon neutrino (ν_μ), tau (τ), and tau neutrino (ν_τ). The electron, the muon, and the tau all have an electric charge and a sizeable mass, whereas the neutrinos are electrically neutral and have a very tiny mass.

In the universe, there are four fundamental forces (strong, weak, electromagnetic, and gravitational) that work over different ranges, and have different strengths. The gravitational force is the weakest but it has an infinite range. The electromagnetic force also has an infinite range, but it is stronger than gravity. The weak and strong forces are effective only over a very short range and dominate only at the level of subatomic particles. Despite its name, the weak force is much stronger than gravity but is the weakest among the other three. As the name suggests the strong force is the strongest of all four fundamental interactions. Except for gravitational, fundamental forces result from the exchange of force-carrying particles, which are bosons: particles of matter transfer discrete amounts of energy by exchanging bosons with each other. Each fundamental force has its corresponding boson: the strong force is carried by the gluons (g), the electromagnetic force by the photon (γ), and the weak force by the W and Z bosons. Although not yet found, the "graviton" should be the corresponding force-carrying particle of the gravitational force. The Standard Model includes the electromagnetic, strong, and weak forces, and all their carrier particles, explaining extremely well how these forces act on all particles.

However, gravity is not part of the Standard Model, and trying to fit gravity into this framework proved to be a difficult challenge. The quantum theory (used to describe the micro world) and the general theory of relativity (used to describe the macro world) are difficult to fit into a single framework, and no one has managed to make them mathematically compatible in the context of the Standard Model. But fortunately when it comes to the tiny scale of particles, the effect of gravity is so weak that is negligible. Only when the matter is in bulk, at the scale of the human body or of the planets, the effect of gravity dominates. So the Standard Model still works well despite the exclusion of one of the fundamental forces.

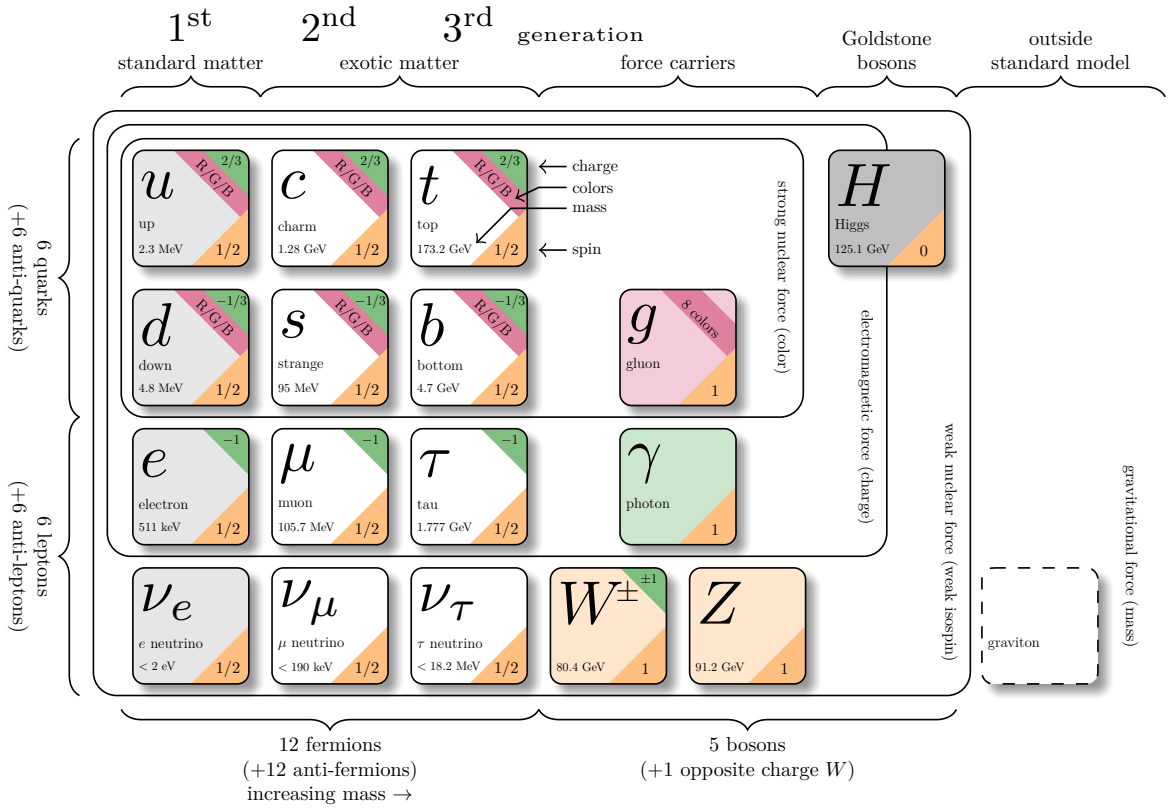


Fig. 3.1: Schematic view of the particles belonging to the Standard Model [96].

The Standard Model includes also the Higgs boson, the particle which explains why the other elementary particles, except photon and gluons, are massive. See Fig. 3.1 for a schematic view of the elementary particles belonging to the Standard Model.

In the following, an overview of the physics goal of three HEP analyses is given. Such analyses are then chosen as use cases for the MLaaS4HEP framework, as described in Ch. 4.

3.1.1 *Top quark production and fully hadronic channel*

The top quark is the most massive of all observed elementary particles (approximately 173 GeV). The large value of its mass makes the top quark contribution dominant in loop corrections to many observables, like the W boson mass. Furthermore, precise measurements of the W boson and the top quark masses are related to the mass of the Higgs boson and are used to evaluate the self-consistency of the SM. Its existence was postulated in 1973 by Makoto Kobayashi and Toshihide Maskawa, and was discovered in 1995 by the CDF and D0 experiments operating at Tevatron [97]. Kobayashi and Maskawa won the 2008 Nobel Prize in Physics for the prediction of the top and bottom quark.

At the LHC, top quarks are mostly produced in pairs via the strong interaction with a production cross section $\sigma_{t\bar{t}} \approx 830$ pb for $\sqrt{s} = 13$ TeV[1]. However, there is a significant number of top quarks produced singly via the weak interaction ($\sigma_t \approx 300$ pb for $\sqrt{s} = 13$

1 $\sqrt{s}$ is the center-of-mass energy.

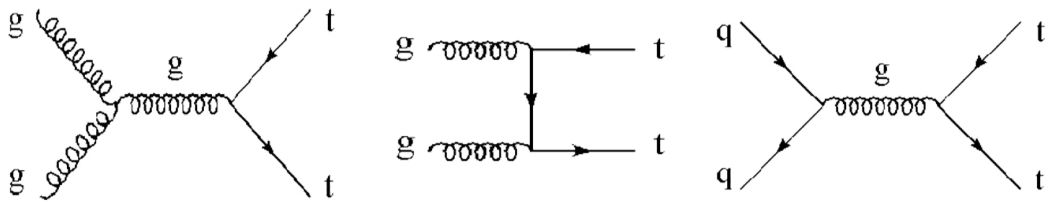


Fig. 3.2: Feynman diagrams for $t\bar{t}$ production at LHC (gluon-gluon fusion in the left and central diagrams, whereas $q\bar{q}$ annihilation in the right diagram).

TeV). At leading order, there are only a few processes that describe the production of $t\bar{t}$: the dominant production mechanism is from the gluon-gluon fusion ($\approx$ 90%), whereas $q\bar{q}$ annihilation accounts for about 10%. See Fig. 3.2 for the Feynman diagrams of such processes.

The top quark decays through the weak interaction almost exclusively to a W boson and a bottom quark. The final states of $t\bar{t}$ are classified by the decay products of the W boson, which can decay to either leptons or quarks. Three final states are distinguished as follows.

- **Dilepton channel**. Both W bosons decay to a lepton-neutrino doublet (branching ratio BR $\approx$ 5%):

$$t\bar{t} \;\rightarrow\; W^+ b\, W^- \bar{b} \;\rightarrow\; \bar{l}\nu_l b \;\; l'\bar{\nu}_{l'}\bar{b} \qquad \text{with } \; l = e, \mu$$

- **Single-lepton channel**. Only one W decays to a lepton-neutrino doublet (BR $\approx$ 30%):

$$t\bar{t} \;\rightarrow\; W^+ b\, W^- \bar{b} \;\rightarrow\; \bar{l}\nu_l b \;\; q\bar{q}'\bar{b}$$

  or:

$$t\bar{t} \;\rightarrow\; W^+ b\, W^- \bar{b} \;\rightarrow\; q\bar{q}'\bar{b} \;\; l\nu_l b$$

- **All-jets (or fully hadronic) channel**. Both W bosons decay hadronically, as represented in Figure 3.3 (BR $\approx$ 46%):

$$t\bar{t} \;\rightarrow\; W^+ b\, W^- \bar{b} \;\rightarrow\; q'\bar{q}b \;\; q'\bar{q}\bar{b} \;\; \rightarrow \;\; j_1 j_2 j_3 \;\; j_4 j_5 j_6,$$

  producing 6 jets[2] ("j_i" above) in the final state.

In the fully hadronic channel, the decay products would be six distinct (i.e. "resolved") jets, but if the top quarks have large transverse momentum (p_T), the particles coming from the decay $t \rightarrow W^+ b$ and those coming from the decay $\bar{t} \rightarrow W^- \bar{b}$ will receive a large boost and emerge quite collimated, giving two distinct wide jets. Wide jets coming from particles with a relevant Lorentz boost are thus called "boosted jets". During the LHC Run 2 of proton-proton (pp) collisions at 13 TeV, jets with p_T up to few TeV can be observed, and at such high p_T the decay products of top quarks can be so collimated that standard reconstruction methods start to be less efficient.

2 A jet is a physics object defined as a spray of collimated particles produced by the fragmentation and hadronization of quarks and gluons originated by a hard collision.

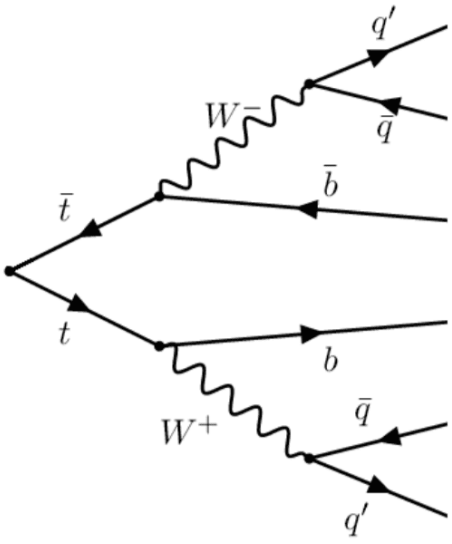


Fig. 3.3: Feynman diagram for $t\bar{t}$ all-jets decay.

The study of such boosted jet allows us to test the Standard Model under extreme conditions, but it may also provide observations of Beyond the Standard Model (BSM) signals. In the CMS experiment, an official analysis was conducted on the fully hadronic channel, where also people from the University of Bologna were involved [98]. In such analysis, the signal events are the $t\bar{t}$ pairs that decay in the fully hadronic channel, while the background is composed of Quantum ChromoDynamics (QCD)[3] multi-jet production that is largely produced at LHC.

3.1.2 *Higgs boson and challenging analyses*

The Higgs boson is considered the most relevant discovery of the last few years in HEP. After almost 50 years from its prediction (occurred in 1964 by Peter Higgs and other five scientists), it was discovered by the ATLAS and CMS collaborations in 2012 at the CERN LHC [99, 100]. Since then, many analyses have been performed to measure its properties with higher precision. At LHC, the Standard Model Higgs boson can mainly be created in four different processes or production modes: gluon-gluon fusion (ggH), vector boson fusion (VBF), Higgs-strahlung (VH), and top-antitop-Higgs ($t\bar{t}H$) associated production (see Fig. 3.4). The first mode is the most probable and the last one is the rarest (see Fig. 3.5 for the different production modes of the Higgs boson and the corresponding cross sections).

In the Standard Model, the Higgs boson is predicted to couple with fermions via Yukawa-like interaction which gives the mass to fermions proportionally to the coupling. To verify the Standard Model, it is important to measure the coupling of the Higgs boson to other particles and check their consistency with the prediction of the Standard Model [101]. In the original discovery, the Higgs boson was seen decaying into $\gamma\gamma$, WW, and

3 QCD is the theory of the strong interaction between quarks mediated by gluons.

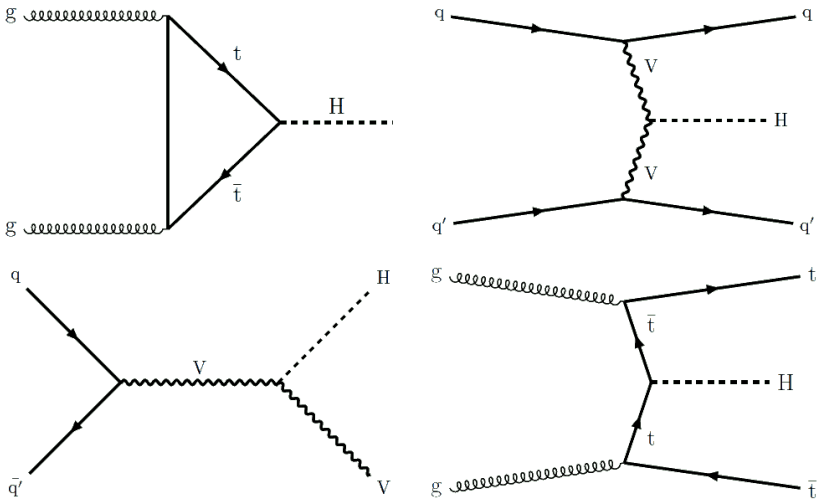


Fig. 3.4: Leading order Feynman diagrams for the main production modes of the Standard Model Higgs boson at LHC: ggH (upper left), VBF (upper right), VH (lower left), and $t\bar{t}H$ (lower right).

Production Mode	Cross Section [pb]
ggH	$48.58\ ^{+2.93}_{-3.77}$
VBF	3.78 ± 0.08
VH	2.26
W^+H	0.84 ± 0.02
W^-H	0.533 ± 0.011
ZH	$0.884\ ^{+0.036}_{-0.031}$
$t\bar{t}H$	$0.51\ ^{+0.03}_{-0.05}$

Decay Channel	Branching Ratio [%]
$H\rightarrow b\bar{b}$	$58.24\ ^{+0.72}_{-0.74}$
$H\rightarrow \tau\tau$	6.27 ± 0.10
$H\rightarrow c\bar{c}$	$2.89\ ^{+0.16}_{-0.06}$
$H\rightarrow \mu\mu$	$(2.18 \pm 0.04)\cdot 10^{-2}$
$H\rightarrow WW$	$21.37\ ^{+0.03}_{-0.05}$
$H\rightarrow gg$	8.19 ± 0.42
$H\rightarrow ZZ$	2.62 ± 0.04
$H\rightarrow \gamma\gamma$	0.227 ± 0.005
$H\rightarrow Z\gamma$	0.153 ± 0.009

Fig. 3.5: In the table on the left, the cross sections for all possible production modes of the Higgs boson are shown. In the table of the right, the branching ratios for all possible decay channels of the Higgs boson are reported. All these values are obtained considering $\sqrt{s} =$ 13 TeV and considering a Higgs boson of 125 GeV [102].

ZZ, that are all boson pairs. Anyway, the dominant decay channel for the Higgs boson is predicted to be $H \rightarrow b\bar{b}$, with a branching ratio of about 58.4% (see Fig. 3.5 for branching ratio of all possible decay channels of the Higgs boson).

3.1.2.1 *Higgs boson Machine Learning challenge*

The Higgs Boson ML challenge was a competition held in 2014, organized by a group of ATLAS physicists and data scientists, and hosted by the Kaggle platform. The Higgs decay channel chosen in this challenge was $H \rightarrow \tau\tau$, which is challenging for two main reasons. First, since neutrinos cannot be directly detected, their presence in the final state makes it difficult to evaluate the mass of the Higgs candidate. Second, the Z boson can also decay

in two taus, and this happens much more frequently than the Higgs decay. Moreover, since the mass of a Z (91 GeV) is not very far from the mass of the Higgs (125 GeV), the two decays produce similar events which are difficult to separate. In the challenge, a specific topology among the many possible ones is chosen, i.e. events where one tau decays into an electron or a muon and two neutrinos, and the other tau decays in hadrons and a neutrino.

The signal and background samples are Monte Carlo simulated events using the official ATLAS full detector simulator. The signal sample contains events in which Higgs bosons were produced. The background sample was generated by other known processes which can produce events that mimic the signal. For simplicity, only three background processes are used in the challenge. The first set of events comes from the decay of the Z boson in two taus. The second set contains events with a pair of top quarks, which can have hadronic tau and lepton in their decay. The third set involves the W boson decay, where one electron or muon and a hadronic tau can appear simultaneously only through imperfections in the procedure of particle identification.

In the challenge, the Approximate Median Significance (AMS) metric [101] is used to evaluate the quality of the classifier. The AMS metric is defined by Eq. 3.1, where s and b represent the estimated number of signal and background events respectively, whereas b_{reg} is a regularization term set to 10 for the challenge.

$$AMS = \sqrt{2\left(\left(s+b+b_{reg}\right) ln\left(1+\frac{s}{b+b_{reg}}\right)-s\right)} \tag{3.1}$$

3.1.2.2 $t\bar{t}H(b\bar{b})$ *in boosted, fully-hadronic final states*

The $t\bar{t}H$ production is very important in the study of the top-Higgs Yukawa coupling, as other production mechanisms (e.g. gluon-gluon fusion) involve loop-level diagrams where contributions from BSM physics could enter the loops unnoticed. The highest branching ratio ($\approx 25\%$) is given by the all-hadronic decay channel with $H \to b\bar{b}$ and all-hadronic $t\bar{t}$ (see Fig. 3.6). In the final state, there are at least eight partons[4] (more might arise from the initial and final state radiation) where four of them are bottom quarks. Despite the highest branching ratio, the all-jets final state is very challenging. There are large uncertainties due to the presence of many jets and it is dominated by the large QCD multi-jet production at LHC, but at the same time, it represents the unique possibility to fully reconstruct the $t\bar{t}H$ system as all decay products are observable.

In the CMS experiment, an official analysis (where also people from the University of Bologna were involved) was conducted on the $t\bar{t}H(b\bar{b})$ channel in the all-jets final state where at least one of the jets is a boosted jet, and where the Higgs boson decays in a pair of well resolved jets identified as a result of the hadronization of bottom quarks. For the identification of $t\bar{t}H(b\bar{b})$ events containing a resolved-Higgs decay a ML model based on Boosted Decision Tree was used in this CMS analysis [96, 103, 104] and the training was done within the TMVA framework (see Sec. 3.4). The Monte Carlo simulation gives

4 The name "parton" was proposed by Richard Feynman in 1969 referring to any particle constituent within the proton, neutron, and other hadrons. Today these particles are referred to as quarks and gluons.

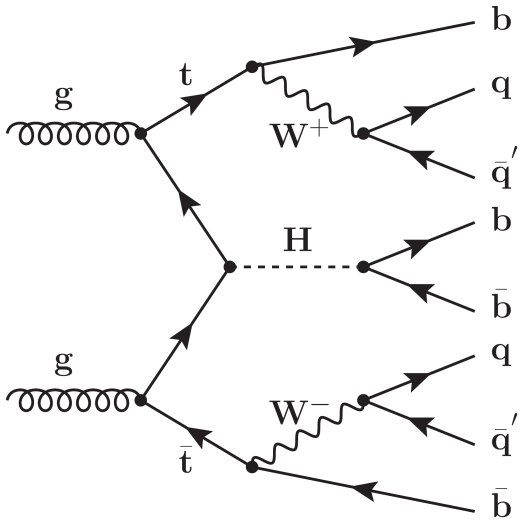


Fig. 3.6: Feynman diagram for the $t\bar{t}H(b\bar{b})$ decay

the events used for training, and the events are selected among the $t\bar{t}H$ sample and the two dominant background samples, namely QCD and $t\bar{t}$, respectively. The $t\bar{t}H$ events with the resolved Higgs boson matching to the system of two b-tagged jets are considered signal events, whereas unmatched $t\bar{t}H$ events, all the QCD, and $t\bar{t}$ events are considered background events. Both signal and background events are required to pass some selection criteria, e.g. to pass the signal trigger, to have at least a boosted jet, to contain no leptons, etc. This selection is aimed to select boosted all-jets-like events.

3.2 LARGE HADRON COLLIDER AND ITS EXPERIMENTS

LHC is a particle accelerator and collider located in a circular tunnel 27 km long and around 100 m underground, the same tunnel that housed the Large Electron-Positron (LEP) collider in the 80s and 90s (see Fig. 3.7). Inside LHC two high-energy particle beams travel at close to the speed of light before they are made to collide. The beams travel in opposite directions in separate beam pipes, which are two tubes kept at an ultrahigh vacuum [105]. The beams are guided in the accelerator ring by a strong magnetic field maintained by superconducting electromagnets. LHC is mainly used to make bunches of protons collide, but for a short period of the year, heavy ions are used.

LHC is the last stage in the CERN's accelerator complex (see Fig. 3.8). The accelerator complex at CERN is a succession of machines that accelerate particles to ever higher energies. Each machine increases the energy of a beam of particles before injecting it into the next machine. In LHC, particle beams are accelerated up to the energy of 6.8 TeV per beam.

From the very beginning of the project conception, LHC has had a detailed plan of upgrades based on specific research objectives and technology improvements, which have gradually led the machine to be more and more powerful [108]. However, over the years, there have been some adjustments and delays on the originally planned upgrades due to various obstacles encountered, e.g. in the supply of materials, in the installation of

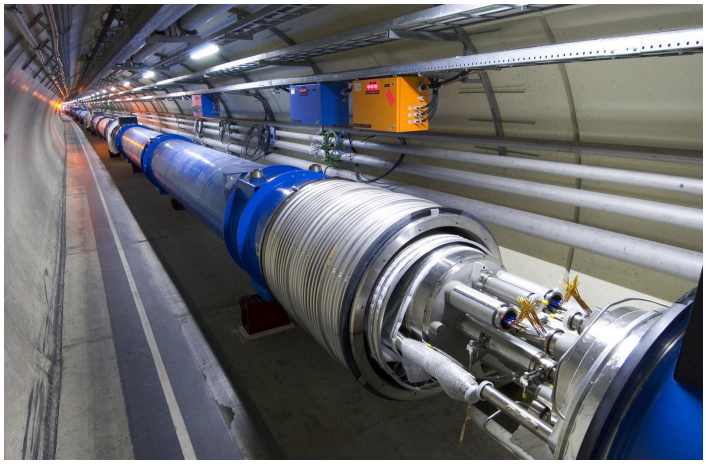


Fig. 3.7: The LHC tunnel [106].

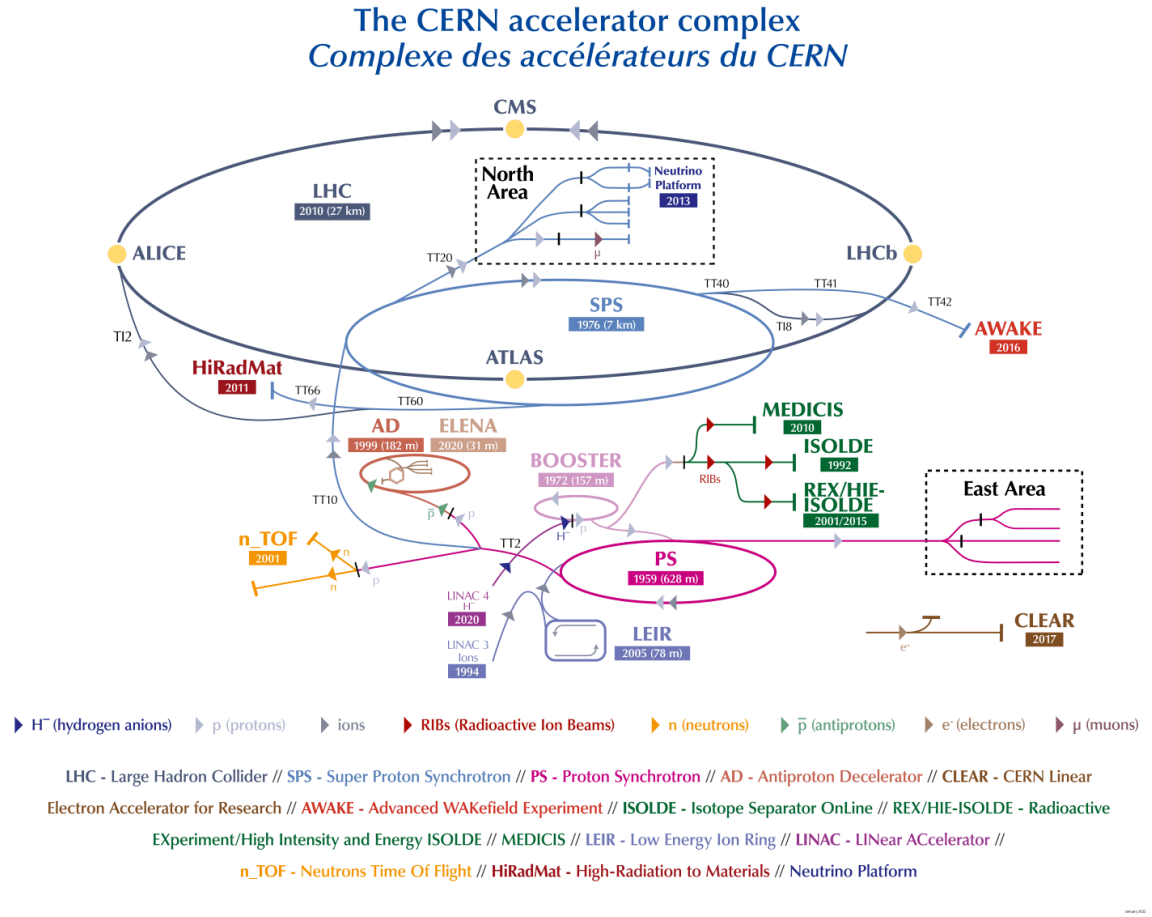


Fig. 3.8: The LHC accelerator complex at CERN [107].

detectors, and during the operations of data-taking. The LHC plan is divided into three different data taking period, called Runs, with two Long Shutdowns (LSs) connecting the different Runs (see Fig. 3.9). During a LS, all planned upgrades on LHC and detectors are integrated. We are currently at the beginning of Run 3, which will last for almost three years. Subsequently, another LS will lead to a new phase called High-Luminosity LHC (HL-LHC). In the HL-LHC phase, the expected luminosity will be a factor of 5 and 7.5 the

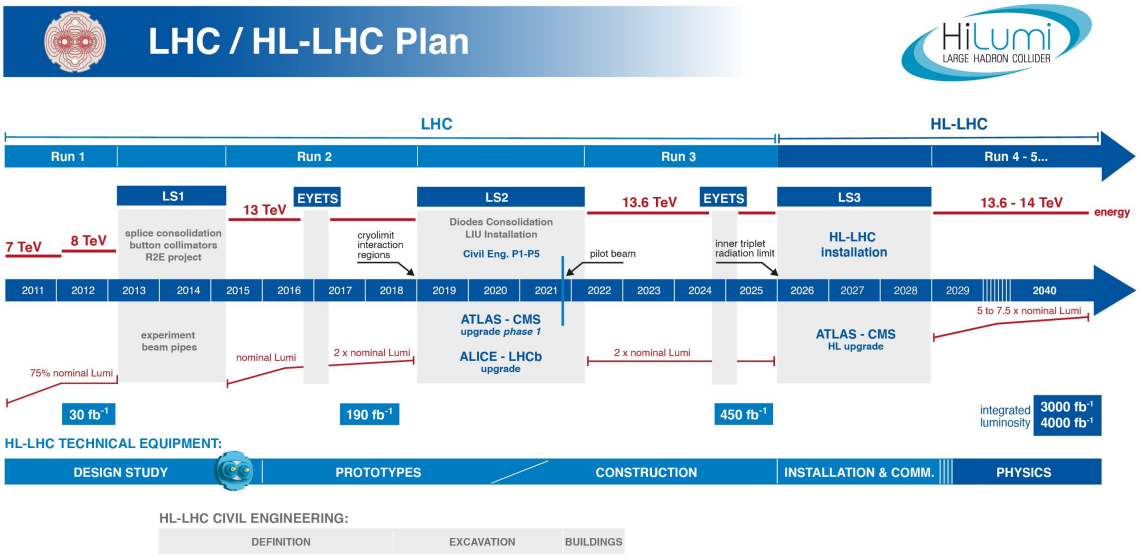


Fig. 3.9: Current LHC and HL-LHC schedule [109].

nominal luminosity[5] during Run 4 and Run 5, respectively. This means that the number of events produced at HL-LHC will considerably increase, and at the same time also the data to be stored and analyzed will increase, which will have also a higher complexity.

The two beams inside LHC are brought into collision inside four detectors: A Large Ion Collider Experiment (ALICE), A Toroidal LHC ApparatuS (ATLAS), Compact Muon Solenoid (CMS), and LHC beauty (LHCb).

ALICE is a detector of 10,000 tonnes, 26 m long, 16 m high, and 16 m wide, dedicated to heavy-ion physics at the LHC [110]. For part of each year, the LHC provides collisions between lead ions, recreating conditions similar to those just after the Big Bang. Collisions in the LHC generate temperatures more than 100,000 times hotter than the center of the Sun, and under such conditions protons and neutrons "melt" freeing the quarks from their bonds with the gluons, creating the so-called quark-gluon plasma. The existence of this plasma and its properties are key issues in the theory of QCD, for understanding the phenomenon of confinement and the problem of chiral-symmetry restoration.

ATLAS is a detector of 7,000 tonnes, 46 m long, 25 m high, and 25 m wide, and it is the largest volume particle detector ever constructed [111]. It is a general-purpose detector which investigates a wide range of physics, from the search for extra dimensions to Higgs boson and particles that could make up dark matter.

Regarding the CMS detector, we provide a more detailed description in Sec. 3.2.1.

LHCb is a detector of 5,600 tonnes, 21 m long, 10 m high, and 13 m wide, and it is specialized in investigating the slight differences between matter and antimatter by studying the beauty quark [112]. Instead of surrounding the entire collision point with an enclosed detector as ATLAS and CMS do, the LHCb experiment uses a series of subdetectors to

5 Luminosity is the ratio of the number of events detected in a certain period of time to the cross section. The ATLAS and CMS experiments have a designed luminosity of $10^{34}\text{cm}^{-2}\text{s}^{-1}$.

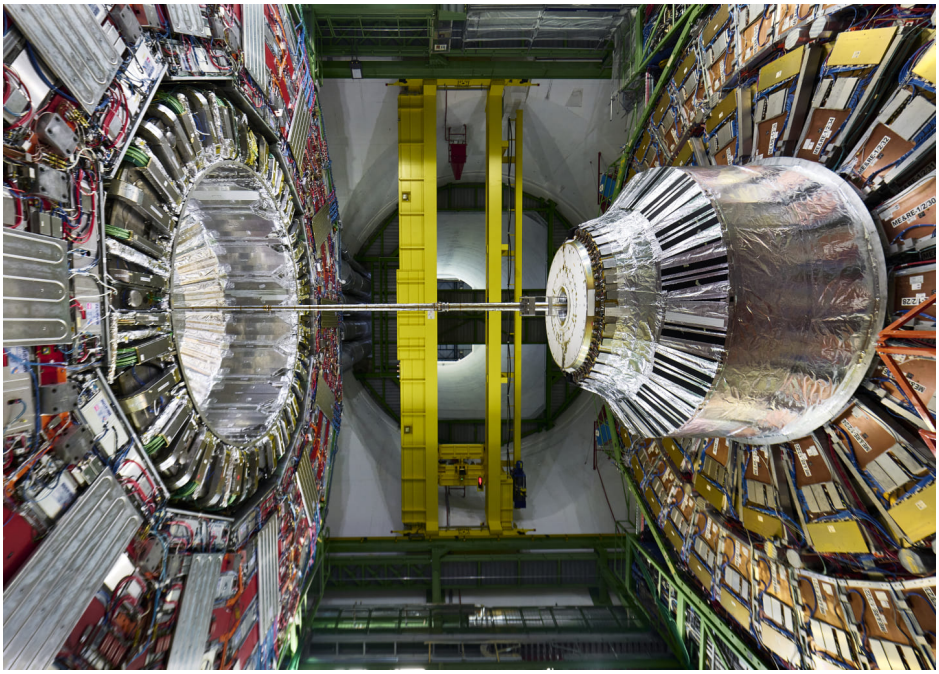


Fig. 3.10: The CMS detector [113].

mainly detect forward particles, i.e. those produced forwards by the collision in one direction. The first subdetector is mounted close to the collision point, whereas the others follow one behind the other for a length of 20 m.

The LHC serves other smaller experiments: TOTEM, LHCf, MoEDAL, and FASER. There is a diverse and rich experimental program connected to the accelerator complex, not only related to LHC experiments: most of the other accelerators in the chain have their own experimental halls, where beams are used for experiments at lower energies.

3.2.1 *The Compact Muon Solenoid experiment*

CMS is a detector of 14,000 tonnes, 28.7 m long, 15 m wide, and 15 m high [113] (see Fig. 3.10). It is a general-purpose detector covering a broad physics program like ATLAS, but unlike ATLAS it uses different technical solutions and a different design of the magnet-system.

CMS is built around a huge superconducting solenoid (which generates a magnetic field of about 4 T) and is composed of several layers of detectors that identify the different particles and that allow building a picture of the collision events. The detector acts as a giant filter where each layer is designed to stop or track different types of particles coming from pp and heavy-ion collisions. To work correctly (i.e. detect and identify particles), CMS needs the following systems (starting from the collision point and going outwards):

- a high-resolution vertex detector, to give an accurate reconstruction of the decay vertex of the heaviest particles;
- a high-quality central tracking system (tracker), to give accurate momentum measurements of charged particles;

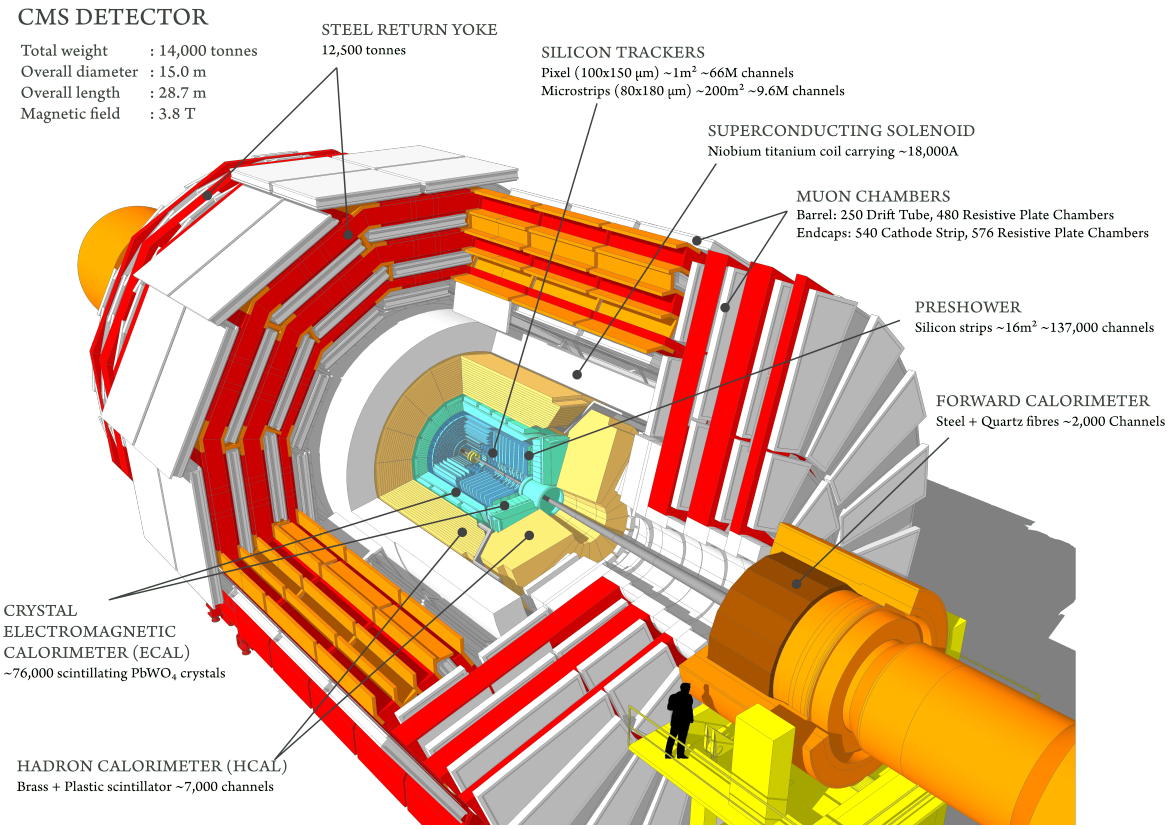


Fig. 3.11: Schematic representation of the systems that make up CMS [114].

- a high-resolution method, to measure the energy of electrons and photons, called electromagnetic calorimeter (ECAL);
- a hadron calorimeter (HCAL), to measure the energy of particles coming from the hadronic shower;
- a high-performance system to detect and measure muons.

See Fig. 3.11 for a schematic representation of the systems that make up CMS.

When there are collisions at LHC, about one billion of pp interactions occur every second in the CMS detector [95, 108, 115]. Indeed, LHC beams cross with a frequency of 40 MHz (every 25 ns), and at each crossing 20-60 pp interactions happen, called pileup. Among the huge number of collisions, potentially interesting events are selected through a trigger system. The CMS trigger system is organized into two levels, represented by the Level-1 (L1) trigger and the High-Level Trigger (HLT). The first one is based on custom electronics and performs data selection in a fast and automatic way by selecting data according to the physics of interest. The L1 trigger uses coarsely segmented data from the muon system and the calorimeters while it keeps the high-resolution data in pipeline memories in the front-end electronics. With the L1 trigger, the frequency of acquired data is reduced to $\approx$ 100 kHz. Then, the events are passed to the next trigger system. HLT is a software system implemented on a farm of thousands of commercial processors. The HLT consists of a streamlined version of the offline reconstruction algorithms, that are optimized to comply with the strict time requirements of the online selection. The HLT contains many trigger

paths, each corresponding to a dedicated trigger, consisting of several steps (i.e. software modules). Each module performs a well-defined task, such as the reconstruction of physics objects (e.g. electrons, muons, jets, and missing transverse energy). The HLT reduces the rate of events to ≈ 1 kHz.

3.3 CMS COMPUTING MODEL

When there are collisions at LHC, the experiments produce a huge amount of data that must be stored and later analyzed by scientists. For example, in CMS during Run 3, the HLT throughput can be even higher than 2 GB/s. In addition to raw data coming from the detector, there is data obtained from processing raw data, data from Monte Carlo simulations, and data produced by the analyses of the users. To manage all this data, a complex computing infrastructure has been designed and deployed, composed of computing centers distributed worldwide, known as WLCG. Currently, almost 1.5 EB of data is stored in disks and tapes of the WLCG [116].

Moreover, each LHC experiment must be equipped with a solid computing model. It includes the description of all the functionalities, the needed software, middleware, and hardware that must be available to enable the collection, distribution, and access of the data in order to streamline the analysis of this data and the production of the desired physics output [95]. All the interactions among each specific component of the overall system, and their management through a certain number of tools and services in real time are also part of the computing model of each LHC experiment.

On top of a set of components that constitute the so-called "middleware layer"[6] (common across experiments), each experiment may also decide to add application layer software solutions that are experiment-specific and perform very specific functionalities that cannot be offered by the middleware layer. While common components are designed and developed by international projects to coherently and smoothly operate on WLCG resources, application layer components are developed by software architects and developers of each experiment collaboration. It is the responsibility of the experiment to make sure that these components can be operated in computing centers around the world in an efficient way, and not disruptive of the activities of other experiments. All these very complex set-up and operations are driven by the WLCG management in close collaboration with the software/computing management of each LHC experiment.

3.3.1 *Grid technologies and WLCG*

The WLCG project is a global collaboration for the building and maintaining of the data storage and analysis infrastructure required by the experiments at the LHC. The main purpose of this infrastructure is to provide computing resources to store, distribute and ana-

6 The software that allows users to access computers distributed across the globe over high-performance networks is called "middleware", as it sits between the operating systems of the computers and the physics application software.

lyze the data produced by the LHC to all the collaboration users regardless of where they are. The idea of a shared computing infrastructure is the basis of the concept of Grid. The WLCG cooperates with several Grid projects, e.g. the European Grid Infrastructure (EGI) and Open Science Grid (OSG). As previously mentioned, middleware projects provide the software layers on top of which the experiments add their own (and different) application layer. At the middleware layer, the main building blocks that make up the infrastructure, i.e. the logical elements of a Grid site, are the following.

- **Computing Element** (CE). It manages the user's requests for computational power in a Grid site. This power is provided by the use of computer clusters organized in farms and managed by software tools. The CE manages the processing requests (called "jobs") submitted by the user and the interactions with the services of the Grid.
- **Worker Node** (WN). This is where the computation actually takes place on a site farm. Here, scripts can be run to configure the environment.
- **Storage Element** (SE). It gives access to storage and data on a site. Data is stored on tapes and disks: tapes are used as long-term secure storage media, while disks are used for quick access to data for analysis. The Storage Resource Manager (SRM) service offers a common interface to access data remotely.
- **User Interface** (UI). It is the machine on which a user interacts with the Grid. Through the UI a user can access remote computing resources.
- **Central services** to help users access computing resources. Examples are information systems, data catalogs, workload management systems, and data transfer solutions.

A job submitted by a user may include requests like storage, processing capacity, availability of analysis software, etc. The computing Grid establishes the user's identity, checks his/her credentials, and searches for available sites that can provide the resources requested by the user. User does not have to worry about where computing resources are located as long as he/she needs are served.

The security on the Grid is based on X.509 proxies which provide authentication for both users and services. The user is endowed with a Grid certificate (issued by the CERN Grid Certification Authority) which is used to get a proxy needed to access the requested services. The authorization is based on the Virtual Organization Management System (VOMS), which contains the list of all users of the Grid registered to the recognized VOs (e.g. CMS) and the tasks they can execute on the Grid itself. Since 2017, the WLCG authorization working group started planning the evolution towards JWTs as means for authentication/authorization on the WLCG [117]. The transition will be gradual and the migration of all the services is expected to be completed in 2026.

3.3.1.1 *WLCG Tiers*

The WLCG is composed of centers belonging to a few levels, called "Tiers", labeled with 0, 1, 2, and 3 (see Fig. 3.12). Each Tier is a computer center made up of computing resources

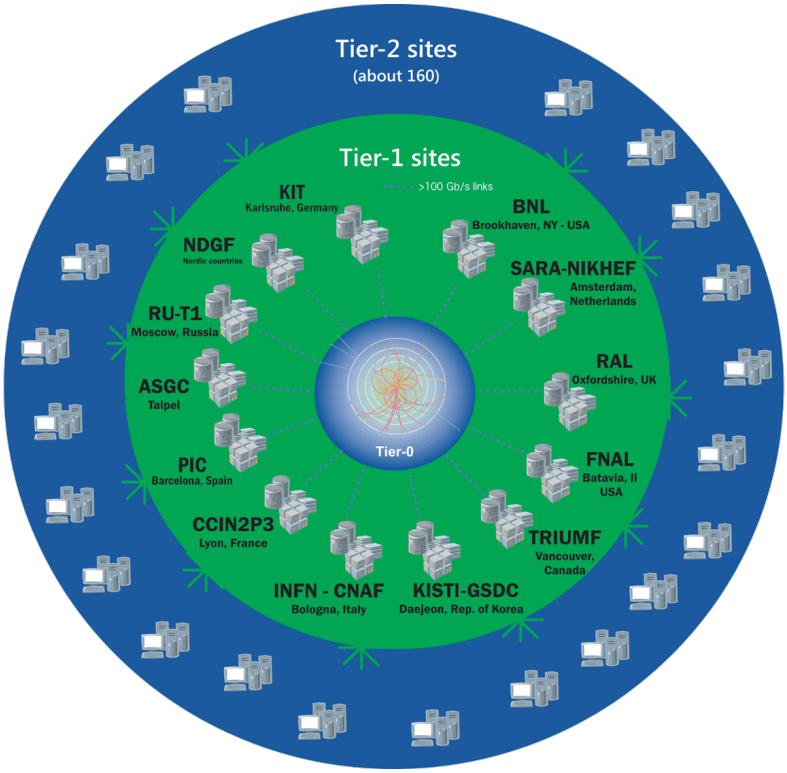


Fig. 3.12: The Tier structure of WLCG [118].

(storage capacity, CPU power, and network connectivity) and provides a specific set of services, different from one level to another.

The structure made by Tiers was formalized by the MONARC model, which provided for a rigid hierarchy. According to the MONARC model, the numeric label associated with the Tier is intended to indicate the quantity and level of functionalities and services offered by the Tier to the community (and also, ultimately, its "size"): the lower the number the higher the quantity and level of services. For example, a Tier-1 implements and offers more storage, CPU, network, additional services, and related availability and support levels compared to a Tier-2. Moreover, a computer center that takes part in WLCG for one or more LHC experiments could theoretically cover different roles and perform different functions for each experiment. For instance, a Tier could have the function of Tier-1 for CMS and Tier-2 for LHCb. Currently, the CMS distributed computing has overcome the original MONARC model, blurring the roles of Tier-1s and Tier-2s, and enabling greater flexibility in the type of workflows that can be executed at each Tier level [119]. CMS has reduced the functional differences between Tiers, while maintaining some operational characteristics, e.g. the site support level.

In the following, more details about the different levels of Tiers are provided.

Tier-0

The Tier-0 is located at CERN, and since 2012 it was extended via a link with the Wigner Research Centre for Physics in Budapest. The latter guarantees greater availability, thus allowing the Tier-0 to be operational even in case of CERN issues. The role of the Tier-0 is to receive raw data from detectors and store them on tapes, perform prompt reconstruction,

as well as distributes the raw data and the reconstructed output to Tier-1s. Despite all data from the LHC passes through this central hub, it provides less than 20% of the Grid total computing capacity.

In CMS, part of the data is processed offline within 48 hours at the Tier-0 via the so-called prompt reconstruction, while the rest is copied to tape (or "parked") for later processing [120]. The idea behind partial prompt reconstruction is to reconstruct promptly at the Tier-0 only a small fraction of the events recorded (an amount which is just enough to ensure data quality and the preparation of analyses), parking the rest on tape to then reconstruct all events at the end of the year, when improved calibrations are available. Data is stored in two copies, one at CERN (the so-called cold copy) and one distributed to Tier-1s (the so-called hot copy).

In order to offer secure and efficient transfers from the Tier-0 to the Tier-1s, a dedicated high-performance network infrastructure based on fiber optic, the so-called LHC Optical Private Network (LHCOPN), was designed and deployed. Although it was initially conceived for Tier-0 to Tier-1 connections only, it was later expanded to cover also all the Tier-1 to Tier-1 connections, adding a crucial redundancy to the overall system. CMS transfer rates from CERN to Tier-1s exceeded 16 Gbps at their peaks during 2018.

Tier-1

The Tier-1 pool consists of 13 computer centers located in different parts of the world, of which 7 were available to CMS during Run-2 (KIT in Germany, PIC in Spain, CCIN2P3 in France, INFN-CNAF in Italy, ASGC in Taiwan, RAL in the UK, and FNAL in the USA). They are responsible to store and reprocess raw and reconstructed data, distribute data to Tier 2s, and safe-keep a part of simulated data produced at these Tier 2s. In Tier-1 sites, the "hot copies" of real and simulated data are stored on both disk caches (for performant access) and tapes (for persistent archive).

Tier-2 and Tier-3

Tier-2s are typically located at laboratories, universities, and other scientific institutes that can store sufficient data and provide adequate computing power for specific analysis tasks. They manage a proportional share of the production and reconstruction of simulated events. There are around 160 Tier-2 sites around the world, of which about 50 are used by CMS. Individual scientists can access the Grid through local computing resources (e.g. Tier-3), which can be local clusters in a university department or even (in principle) an individual PC. Unlike Tier-2 centers, a Tier-3 site does not sign any Memorandum of Understanding with WLCG, meaning that there is no formal engagement between WLCG and Tier-3 resources. This implies no check on availability and therefore a great flexibility in site management. A Tier-3 is a good resource for the local community of physics end-users and analysts.

3.3.1.2 *Opportunistic resources, cloud resources, and HPCs*

The WLCG experiments benefit from a relatively large amount of opportunistic resources, with large fluctuations between experiments (on average 20% beyond the CPU pledge level) [121]. Opportunistic resources today are largely provided in the form of Grid resources at WLCG sites and Tier-3s, but in part they are made accessible through non-Grid interfaces, e.g. cloud and HPC facilities.

Commercial cloud facilities can be an opportunity for their elastic capabilities, although it is not obvious whether this flexibility is necessary for the expected needs of HL-LHC (it is likely to be specific to the experiment). Given the costs of today's cloud providers, commercial cloud resources are not economically a good alternative to on-premise. So far, cloud resources have been integrated in some cases as an extension of WLCG facilities, and in other cases directly with the workload management systems of the experiments.

Today, HPCs are used in production by CMS for all kinds of central workflows, e.g. event generation, simulation, digitization, reconstruction, and creation of analysis formats [120]. In recent years, HPC machines contributed significantly to the processing of Run 2 data and the generation of the related Monte Carlo samples: between 5 and 10% of the total used computing power dedicated to these activities came from HPCs. Italy was the second largest provider of HPC resources to CMS during 2019 and 2020 with the Marconi A2 machine at INFN-CINECA. Substantial national and supra-national investments have been made in this area and the nascent exascale machines are on the horizon, which will be part of the future scientific computing infrastructure.

3.3.2 *CMS data types*

To extract useful information for a HEP analysis, a physicist must combine a variety of blocks, e.g. reconstructed information from raw data of the detector (specified by a combination of trigger paths and possibly further selected by cuts on the reconstructed quantities), Monte Carlo samples which simulate the physics signal under study, and background samples (specified by the simulated physics process). This information is stored in event collections and datasets.

An *event collection* may correspond to the event data from a particular trigger selection from a given "run" (corresponding to a single bunch crossing). This could very easily be any type of user-defined "ntuple". A *dataset* is defined as any set of event collections that are grouped and analyzed together depending on physics attributes, e.g. their trigger path or Monte Carlo physics generator, or on the fact that they are a particular object model representation of those events. An event collection is the smallest unit within a dataset that a user can select, and generally, the reconstructed information needed for the analysis is all contained in one or a few event collection(s).

Data is stored in ROOT files (for more details see Sec. 3.4.1), and the smallest unit in computing space is called "file block" which corresponds to a group of ROOT files likely to be accessed together. This needs a mapping from the physics abstraction of the event (event collection or dataset) to the file location.

Starting from raw data produced by the online system, successive stages of processing refine this data, apply calibrations, and create higher-level physics objects [122]. The event information from each step in the simulation and reconstruction chain is logically grouped into the so-called data tier, where each one contains different levels of information about the same event. Every bit of data in an event must be written in a supported data format. A data format is essentially a C++ class that defines a data structure (a data type with data members). The term data format can be used to mean the format of data written using the class (intended as a sort of template), or to the instantiated class object itself. CMS uses a number of event data formats with varying degrees of detail, size, and refinement. A data tier may contain multiple data formats.

The three main data tiers in CMS are the following.

- **RAW**. It contains the raw detector information (e.g. detector particles hits). In the vast majority of cases, RAW data is not used directly for analysis.
- **RECO** (RECOnstructed data). It is the output of the first pass of processing at the Tier-0. This layer contains reconstructed physics objects, but still contains many details about the event. RECO data can be used for analysis, but in general, its size is too big for frequent use.
- **AOD** (Analysis Object Data). It is a "distilled" version of the RECO event information, which provides a trade-off between event size and the complexity of available information in order to optimize flexibility and speed for analyses. The AOD contains enough information about the event to support all the physics analyses.

Since in Run 2 the data produced would have increased by a factor of ≈ 10 compared to Run 1, and to avoid the intermediate ntuples that were produced by analysis groups from AOD files, a new data format was introduced in 2014 for the beginning of Run 2. The motivation for the MiniAOD format is to have a small and quickly derived data format on which the majority of CMS analyses can run. This format was intended to have enough information to serve about 80% of CMS analyses, while dramatically simplifying the disk and I/O resources needed for the analyses. The MiniAOD size is approximately 10% of the size of the AOD format (see Fig. 3.13).

In 2018, CMS introduced a new compact event data tier, named NanoAOD, intending to serve the needs of a substantial fraction of its physics analyses with a per-event payload of about 1-2 kB, ≈ 20 times smaller than MiniAOD [124, 125]. NanoAOD achieves such a strong data reduction by keeping only high-level information about physics objects, e.g. jets and leptons, dropping their individual constituents, and reducing the precision of the stored variables. The content of a NanoAOD file takes the form of a flat ROOT TTree with no specific dictionary needed for I/O, miming the typical format of user ntuples. Conventions on column (branches) names are used to group information belonging to the same high-level object: branches from the same object have all the same length in an event and no further array nesting within branch elements is allowed. The current content of NanoAOD consists of all physics objects, including jets, electrons, muons, photons, trigger

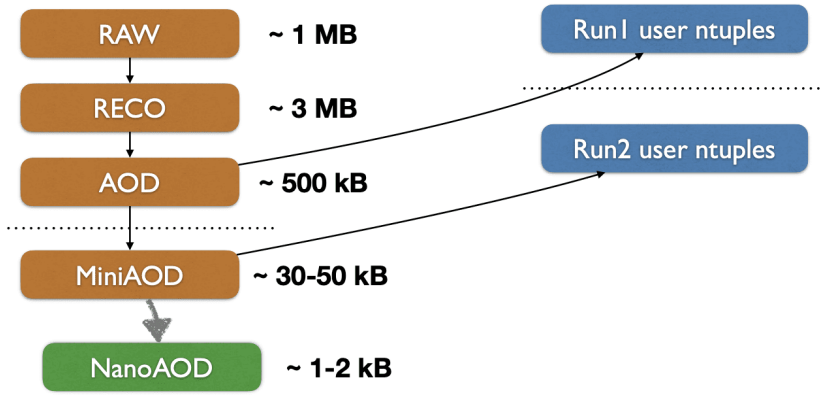


Fig. 3.13: Evolution of data formats and their respective event sizes [123].

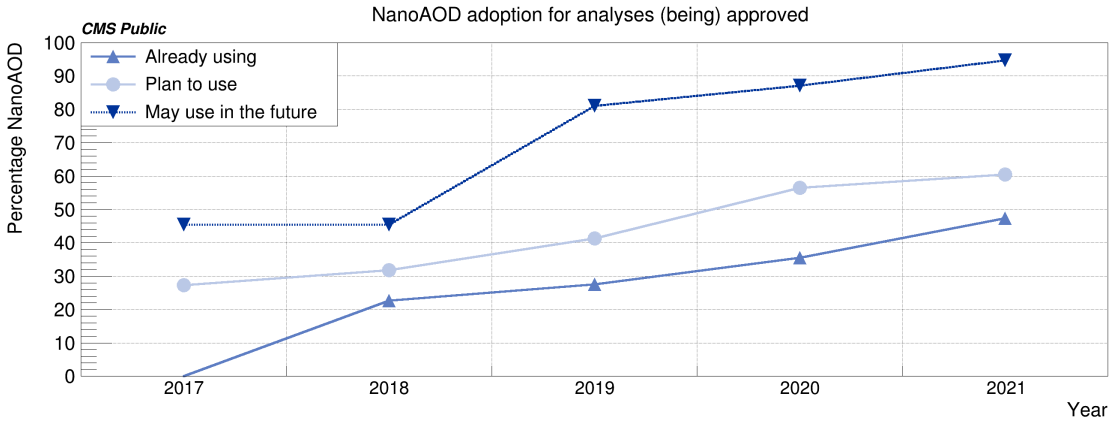


Fig. 3.14: Results of surveys of adoption of the NanoAOD data tier, or plans to do so in the near or far future, by physics analyses since 2017 [120].

information, missing transverse energy, and more. The NanoAOD event content undergoes a continuous process of tuning, based on the experience accumulated with analyses adopting it. For a detailed description of the NanoAOD data tier content, see [126]. In the last years, the portion of the analyses using NanoAOD has increased substantially (see Fig. 3.14): currently, the percentage of analyses that have adopted NanoAOD is 30%.

3.3.3 *CMS data processing tasks*

CMS executes a variety of tasks on its distributed computing infrastructure for reconstructing and simulating collision data and analyzing both.

- The *generation* task is the first step in simulating collision data. It uses Monte Carlo event generators, which are software packages provided by the theoretical particle physics community, to generate events from theoretical principles. CMS relies on several event generators, e.g. Pythia, POWHEG, and MadGraph.
- The *simulation* task takes the output of the generation task and models the energy deposition of the particles interacting with the material in the CMS detector. Two types of simulation exist in two separate frameworks: Geant4-based and a parameterized

fast Monte Carlo chain, known as "fastsim" in CMS. The fastsim is a full Monte Carlo processing chain, which is combined with a parameterized digitization and reconstruction (steps described below), relying on a simplified geometry. This type of data processing chain is much faster than the regular one: the datasets obtained with it give the output of the standard simulation and reconstruction at about the level of 10% approximately 6 times faster.

- The *digitization* task consists in modeling the electronic processing of the signal produced by the energy depositions. Moreover, it simulates the effect of additional pp interactions within the same (in-time) or neighboring (out-of-time) bunch crossings (pileup).
- The *reconstruction* task executes all the algorithms necessary to interpret signals as due to the interaction of identifiable particles with the detector. Reconstruction output comes in several different formats designed for the different needs of the analyses tasks, i.e. RECO, AOD, MiniAOD, and NanoAOD.
- The *analysis* task reduces the size of the reconstruction output by filtering events and eliminating the information not required for a particular physics measurement. Properties of specific events are calculated and information is reduced to graphs or histograms, building a statistical interpretation of the results.
- With the increasing success of deep learning methods to address HEP challenges, CMS is investigating and developing ML-based applications at all levels. Before being deployed and embedded within the data processing steps, models need to be trained: this additional data processing step is called *ML training*.

3.3.4 *CMS services and operations*

As already mentioned in Sec. 3.3, the integration of resources at CMS computing centers into a single and coherent system is based on the Grid middleware which presents a standardized interface to storage and CPU facilities at each WLCG site. At the same time, however, a number of CMS-specific distributed computing services operate above the generic Grid layer, enabling higher-level data and workload management functions. These services, in some cases, require CMS-specific software agents to run at the sites, in addition to generic Grid services. An overview of the CMS computing services components is shown in Fig. 3.15. In the next two paragraphs, some of the most important services will be briefly presented, with the caveat that this is only a general high-level overview, and some existing additional services (which could rapidly evolve over the years) are not mentioned.

3.3.4.1 *Data management*

CMS requires tools to catalog existing data, track the location of corresponding physical data files on-site storage systems, and manage and monitor the flow of data between

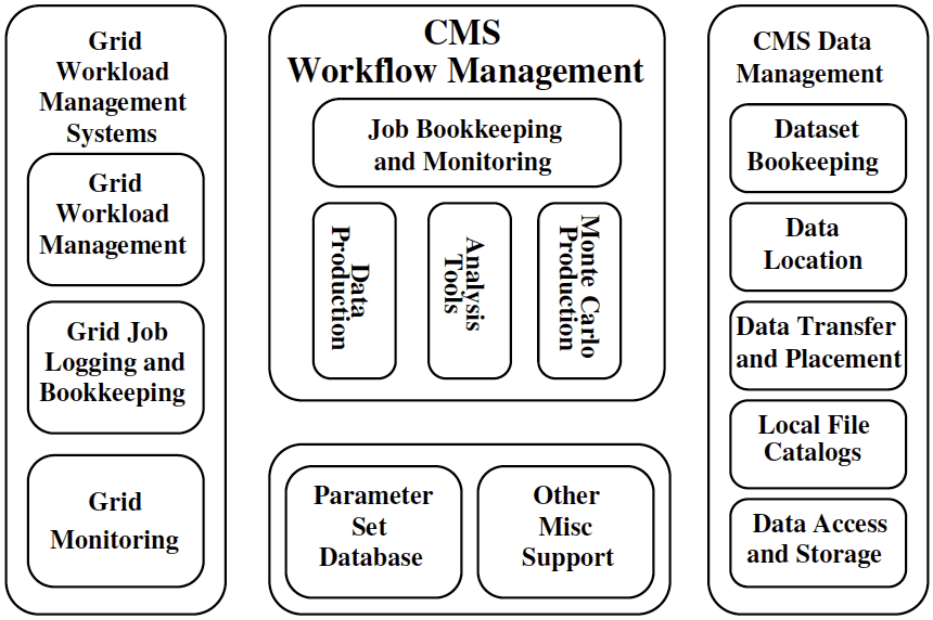


Fig. 3.15: Overview of the CMS computing services [127].

sites. A multi-tiered catalog system, the Dataset Bookkeeping System (DBS), is used to provide the connection between abstract datasets or event collections and physical files. It provides a standardized and queryable means of cataloging and describing event data. It is the primary means of data discovery for the user, which answers the question "what data of this type exists in the system?". In particular, it must express the relationships between datasets and event collections as well as their mapping to the packaging units of file blocks and files. A second functionality of the catalog system is needed to provide the mapping between file blocks to the particular sites at which they are located, including the possibility of replicas at multiple sites. For this, Local File Catalog at each site are used to map logical files onto physical files in local storage.

The Data Aggregation Service (DAS) is a service used in CMS to allow a data search through a query language. The Physics Experiment Data Export (PhEDEx), is a reliable and scalable dataset replication system in production for CMS since 2004, then replaced by Rucio in 2020 to meet the technical and scale needs of CMS.

3.3.4.2 *Workflow management*

The management of grid jobs is handled by a complex system, which nowadays has completely migrated towards the use of "pilots" on an HTCondor infrastructure. The goal is to schedule jobs onto resources based on CMS policy and priorities, to assist in monitoring the status of those jobs, and to ensure that site-local services can be accurately discovered by the application once it starts executing in a batch at the site. All these issues should be invisible to the user.

The centrally-steered production activities are very different, in terms of concepts and tools used, from the distributed analysis. In the former case, the actual management of the whole task is performed by a WMCore/WMAgent infrastructure, while in the latter

case, the preparation, submission, and monitoring of the job are done via the CMS CRAB distributed analysis toolkit. The distributed analysis comprises many jobs running on the Grid that are intrinsically independent of each other. This fact allows for the distribution of the workload based on the capabilities of the computing center and the location of the input data. When the analysis has been assigned to a specific computing center, a CRAB job wrapper performs the environment configuration, then runs the user analysis on the local dataset, and delivers the requested data to the user. CRAB takes care of interfacing with the user environment, provides data-discovery and data-location services, manages the status reporting, monitoring, and user job output that can be saved on a user-selected storage element. Through a simple configuration file, a physicist can thus access data available on remote sites as easily as he/she can access local data: all the complexities of the infrastructure are hidden users as much as possible. A client-server architecture is also available, so the job is not directly submitted to the Grid but to a dedicated CRAB server, which, in turn, manages the job on behalf of the user, interacting with the services of the Grid.

3.3.5 *Computing towards HL-LHC*

Over the next few years, many challenges will need to be addressed from the computing perspective in order to be prepared for the upcoming HL-LHC phase [121].

- **Fitting within a restricted cost envelope for computing**. It has been clear for several years that the budget outlook for the HL-LHC computing will be limited, with the key message from funding agencies that the LHC community must keep computing costs within a "flat-budget" scenario in long term. This is a real challenge as a number of factors conspire against it. The overall level of requirements for computing resources is significantly higher in Run 4 than in Run 3, for several reasons. The naive cost improvements we were previously used to, following the Moore's Law[7], are no longer applicable. Currently, the affordability of computing on the HL-LHC timescale is almost impossible to predict with any certainty: for many components, it is not technology that drives the prices, but market forces, lack of availability, and uncertainty about the future of some technologies.

- **Managing Exabyte scale data**. The expectation from both ATLAS and CMS during Run 4 is close to 1 EB of data to be collected each year, to which derived and simulated data are added. Thus each experiment will be in a multi-Exabyte per year regime of data to be managed, representing a significant challenge compared to the current situation. There are several strategies focused on addressing this challenge, that can be grouped into: efficient and more cost effective data management, reduction in the size of derived data that is to be distributed, and management of operations costs. In the future, data management must be able to serve data to heterogeneous compute resources, from a pool of large scale data stores with automated

7 The number of transistors in a dense integrated circuit doubles about every two years.

and policy-driven replication and load balancing within and between the stores (the so-called "Data Lake"). This will enable the optimization of managed data in larger sites with large storage systems and experienced staff, as well as a more effective provision of compute at both traditional grid sites, or opportunistic use of clouds and HPC sites without the need to install special data services in them. This should result in better use of available funds for compute services at many sites, without the need for expensive storage systems or additional staff. In this scenario, there is a change of strategy which moves away from replicating data "everywhere" to serving data when and where it is needed. The latter is more efficient as only data that is actually being processed will be transferred. Moreover, another way to reduce the amount of data transfers and management, and the need for large managed storage systems is the adoption of small analysis datasets, e.g. based on NanoAOD in CMS.

- **Heterogeneous computing and portability**. In recent years, the era of "x86-only" processor dominance has begun to change. Even within the "x86" processors, the introduction of vector units, multi-threading, and other forms of parallelism support has meant that most of HEP software does not use all available processing at maximum efficiency. Furthermore, the introduction of new types of processors (e.g. non-x86-64 architectures), and the ubiquity of coprocessors (e.g. GPUs) has made it clear that the software must be adapted to make use of the available processing, and also to be able to port it to a more parallel environment. To do that effectively, in most cases, re-engineering of the software and algorithms is required. There is a common software portability challenge that needs to be addressed, regardless of the type of facility used (Grid, HPC, or cloud).

- **Common software tools and services**. Another aspect of the software challenge is to continue to build common software solutions. During Run 1 and Run 2, commonalities arose, in addition to the long standing examples of Geant4 and ROOT: there are examples of common event generators, run-time libraries, use of CVMFS for software delivery, and the birth of a project towards common data management service. It is very important to continue to identify the commonalities and benefit from them, since the effort for many competing but similar solutions are expensive. Additionally, it is important to note that other related sciences are now facing similar challenges to HL-LHC and the commonalities between LHC experiments can bring benefits, and eventually build a stronger overall community.

3.3.5.1 *Evolution of the CMS computing model*

The increase in granularity of the CMS detector and the higher complexity of the collision events that there will be in the HL-LHC phase pose challenges in the areas of data acquisition, processing, simulation, and analysis [120]. These challenges cannot be solved only by increasing the computing resources available to CMS, but they must be followed by major improvements in the computing model and computing software tools, as well as data processing software and common software tools. In CMS, there are various R&D

activities with the aim to reduce resource needs in the HL-LHC era, or mitigate risks, and enable the efficient use of HPCs and accelerators. In the following, we provide details of some of them.

- In order to reduce storage requirements, one focus is on reducing the size of the RAW events in collaboration with CMS sub-detectors experts. The overall impact of this R&D would be a 15% reduction in the RAW data size. Concerning the other main data formats (AOD, MiniAOD, and NanoAOD), advances in the ROOT columnar storage point to a 20% reduction in size for each of these tiers. This reduction is possible thanks to the adoption of the new columnar format RNtuple (see Sec. 3.4.1). Moreover, a goal of CMS is the largest possible adoption of the slimmest analysis format, NanoAOD, aiming to reach 50% of the analyses by the end of Run 3.

- In order to reduce the requirements in terms of CPU, there are several R&D lines covering, e.g. generators code performance improvements, Geant4 improvements, optimization of reconstruction code, and adoption of the mkFit tracking tool.

- Following the successful deployment of GPUs in HLT for Run 3, CMS aims to use accelerators, e.g. GPUs, for part of its offline data processing chain. The goal is to take advantage of heterogeneous architectures, at HPCs or elsewhere, for instance at future WLCG centers. The main elements necessary to achieve this goal are the support of heterogeneity in the data processing framework, an adequate programming model, and an enabling computing infrastructure. Then, once the applications can efficiently offload calculations on accelerators and a diverse distributed system (also made up of heterogeneous platforms) is under control, the focus can shift to maximizing the amount of code that can be offloaded.

- It is very likely that HPCs will be a substantial component of the future HEP distributed computing infrastructure. In this direction, examples of R&D activities cover the evolution of the workload management system, and the scalability of the submission infrastructure.

CMS developed a mathematical model that can be used to predict the computing resource needs of the experiment given a small number of input parameters. It is used for annual resource requests, as well as for projecting the needs into the HL-LHC era. The expected results of ongoing development activities are incorporated into the model to assess their impact on overall resource needs. Two scenarios for the projections are considered. The first one, called *Baseline Scenario*, is the extrapolation of the current computing model, practices and software performance to the HL-LHC era, without including the benefits from current developments. The second one, called *Weighted Probable Scenario*, is the scenario that includes the weighted probable impact of the R&D activities. The results aggregate the resources of Tier-0, Tier-1s, and Tier-2s.

Fig. 3.16 shows the projections of the CPU, disk and tape needs of CMS in Run 4 and Run 5, according to the LHC schedule. In all projections plots, the gray band represents the foreseen capacity of resources assuming a flat budget (with a 10% to 20% annual resource

increase). The Weighted Probable Scenario represents a possible solution to the HL-LHC computing challenge and is already very promising (as can be seen from plots of Fig. 3.16), even only considering today's R&D. However, projections are affected by uncertainties (particularly those relating to market trends of computer hardware), and approaching the lower part of the availability band does not necessarily imply solving the HL-LHC computation challenge.

Fig. 3.17 shows the breakdown of CPU, disk and tape usage in 2031 for the Baseline Scenario. Reconstruction is the biggest CPU consumer, tape usage is driven by RAW data, whereas most disk is used for Monte Carlo simulations.

3.4 TOOLS FOR ANALYSIS IN HEP

Each LHC experiment has its collection of software used to cover many different tasks. For example, the CMS experiment collection of software is called CMSSW, which contains the needed tools for simulation, calibration, and alignment, as well as the reconstruction modules that process event data so that physicists can do analysis [129]. A key ingredient for a HEP analysis is represented by the ROOT framework, which provides the data format commonly used to store HEP data, as well as tools to access and analyze such data. In the last years, other tools have been added to the HEP analysis landscape, to be traced back to the exponential growth of data science. In Sec. 3.4.1 we provide an overview about the ROOT framework, in Sec. 3.4.2 about common data science tools used in HEP analysis, and in Sec. 3.4.3 about the future of HEP analysis tools.

3.4.1 *The ROOT framework*

ROOT is a framework born at CERN (the first public release was at the end of 1995), mainly written in C++, and is a fundamental ingredient of most HEP workflows, in areas such as data persistency, modeling, graphics, and analysis [130]. The ROOT project has a key role in the HEP community with excellent and active connections with the experiments including direct investment from them. The main characteristics of the ROOT framework are reported below [131, 132].

- ROOT is used to save data (and any C++ object) in a compressed binary form in a ROOT file. In the same file, the object format is also saved, therefore the code for the C++ classes corresponding to all objects saved in the file can be generated automatically (this means that ROOT is a self-descriptive file format). In a ROOT file, plain tables (named ntuples, as in mathematics), TTree data, RNTuple data, and non-event data (such as histograms) can be stored. TTree provides a *tree* data structure, which is extremely powerful for fast access to huge amounts of data, orders of magnitude faster than accessing a normal file (considering HEP use cases, ROOT I/O is faster than potential alternatives such as HDF5 or Parquet). TTree represents a columnar dataset, consisting of a list of independent columns called *branches*, represented by the TBranch class. Branches can store simple variables or more complex

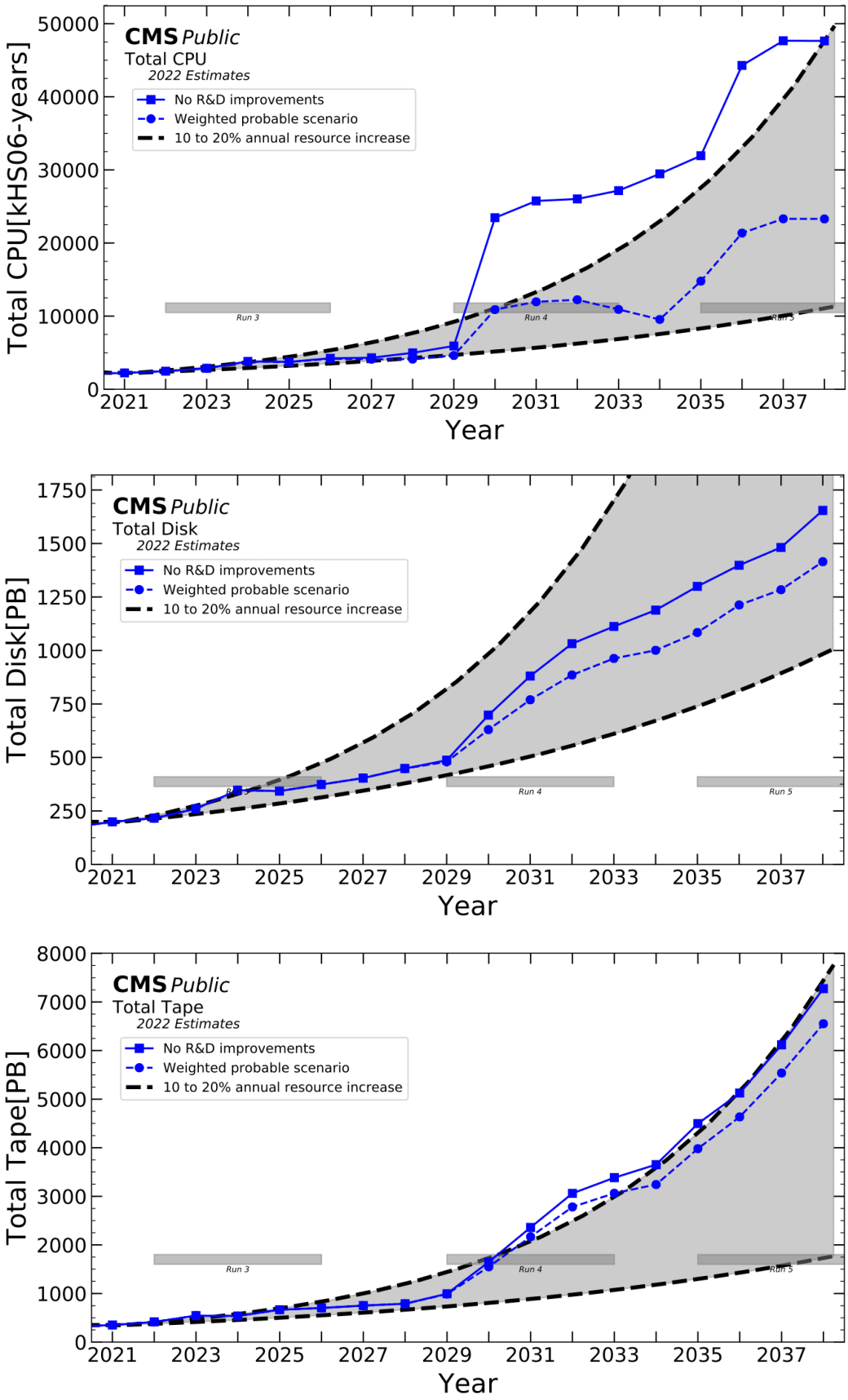


Fig. 3.16: Updated projections of CPU, disk, and tape needs into HL-LHC [128].

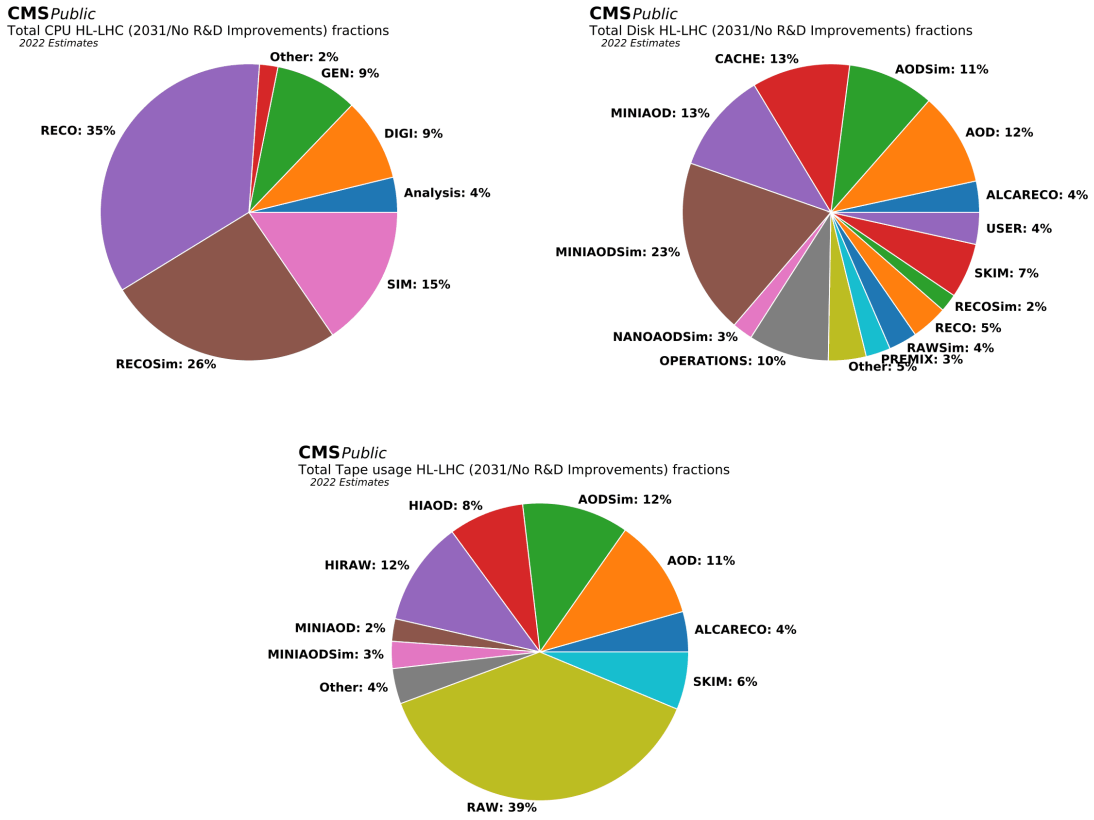


Fig. 3.17: Breakdown of CPU, disk, and tape usage in 2031 for the Baseline scenario. Reconstruction is the biggest CPU consumer, tape usage is driven by RAW data, whereas most disk is used for Monte Carlo simulations. [128].

objects, including other trees. The RNTuple class is the designated successor of the TTree class, which introduces a new data format and new data access APIs, that are backward incompatible with TTree. This break in compatibility allows to reach space savings, increase the read/write speed, and improve the robustness of the APIs for the decades to come. The RNTuple development started in 2018 and the training of physicists is planned for 2024.

- The ROOT data format allows for storing experiment-specific and data management specific metadata, and integrates well with the experiment computing models.

- You can access the data saved into one or several ROOT files from the PC, from the web, or from large-scale file delivery systems, e.g. the WLCG. ROOT trees spread across several files can be concatenated and accessed as a single object, allowing for loops across huge amounts of data.

- The ROOT core I/O integrates with 3rd party components which are important for the I/O of LHC experiments. As remote access protocol, it integrates XRootD (crucial for production workflows), whereas Davix for HTTP access (used in particular for training, outreach, and open science). As authentication plugins, it uses X.509 certificates and SciTokens.

- Several mathematical and statistical tools are provided to operate on data. The potential offered by a C++ application and the parallel processing are available for any type of data manipulation.
- Results can be displayed with histograms, scatter plots, and fitting functions. ROOT graphics is easy to use and can be adjusted real time with a few clicks.
- ROOT provides a set of bindings so that it can be seamlessly integrated with existing languages such as Python and R.
- ROOT provides the TMVA library that offers interfaces and implementations of supervised ML techniques. The package includes for example MLPs, Boosted/Bagged decision trees, and SVMs.
- ROOT's RDataFrame provides a modern and high-level interface for the analysis of data stored in TTree, CSV, and other data formats, both in C++ and Python [133]. RDataFrame allows to apply filters and creating custom columns starting from an input dataset. Users can exploit all the resources available on their machines transparently, thanks to the multi-threading and other low-level optimizations that RDataFrame offers.

3.4.2 *Data science tools for analysis*

The HEP analysis software landscape is changing [134]. Although physicists have used data science tools like NumPy, SciPy, Pandas, Matplotlib, Spark, and Hadoop over the past 15-20 years, these tools have only become an important part of the HEP analysis ecosystem in the past 4-5 years. The reasons for this change can be summarized in two major influences. Externally the HEP world, data analysis tools have focused on Python and array-oriented programming. Internally the HEP world, some organizations have mediated the spread of data science software, helping physicist-developers to find each other, reduce duplication, and use the tools developed by others. These organizations are actively supporting the development of such software, especially for the core components that enable specialized tools.

The first of these organizations is the HEP Software Foundation (HSF), conceived in 2015, that in 2017 produced a roadmap white paper (published as an article in 2019 [135]) on the software and computing challenges that will be faced during the 2020s. The HSF's primary mission is communication: helping project developers to work towards a common vision, particularly across experiment boundaries. Then, there is the Python in HEP (PyHEP) working group, which brings together a community of developers and users of Python in particle physics (embracing a broad community, from HEP to the Astroparticle and Intensity Frontier communities), with the purpose of improving the sharing of knowledge and expertise. Activities-wise, it organizes workshops and meetings on both HEP domain-specific and data science packages. Another organization is DIANA/HEP and its successor IRIS-HEP, funded by the NSF in 2015 and 2018, respectively, to support the

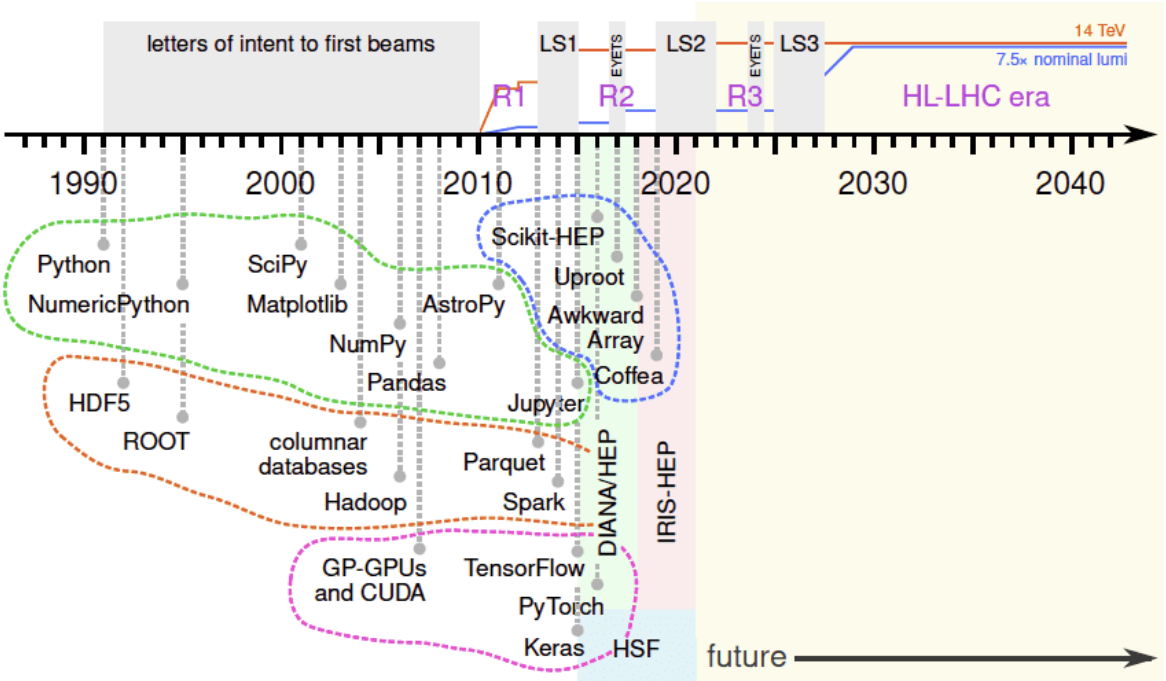


Fig. 3.18: Timeline of developments in scientific Python (dashed green line), GPUs and ML (magenta), big data (orange), and data science in HEP (blue), overlaid on the HSF, DIANA/HEP, IRIS-HEP, LHC, and HL-LHC timelines [134].

development of HEP software by funding software developers, creating an intellectual hub for sharing knowledge, and offering education and training. The main directly funded software products include Awkward Array [136] and Uproot [137]. DIANA/HEP and IRIS-HEP organize meetings held by physicists and occasionally draw data scientists from industry to share techniques and tools. It is worth mentioning other two organizations, SWIFT-HEP and Scikit-HEP, and a few projects, e.g. Coffea, FAST-HEP, and zfit.

Major developments in the data science landscape (comprehensive of scientific Python, big data, GPUs, and ML) roughly coincide with the development and Run 1 of LHC, from the late 1990s to 2015. The HSF, DIANA/HEP, and IRIS-HEP that contributed to the development of data science-oriented software in HEP were active after 2015 (see Fig. 3.18).

Moreover, the usage of Python in HEP ramped up in the years following 2015, and in particular, it overcame C++ for CMS users in 2019. This can be seen from Fig. 3.19, where CMS users are identified as those who fork the GitHub repository of CMSSW, i.e. `cms-sw/cmssw`. This way of identifying CMS users is certainly reductive and it does not actually consider the entire collaboration, but it shows a trend that can be generalized with proper caution. Moreover, here each repository is considered either entirely Python or entirely C++, but in reality, most are mixed.

HEP analysis culture is in the midst of a transition, which is confirmed by other aspects, such as the following.

- HEP data analysts are becoming averse to monolithic frameworks. A framework (e.g. ROOT) provides all the essentials in one package but requires users to fit their workflows into it. On the contrary, data science libraries and toolkits have a very modular mindset.

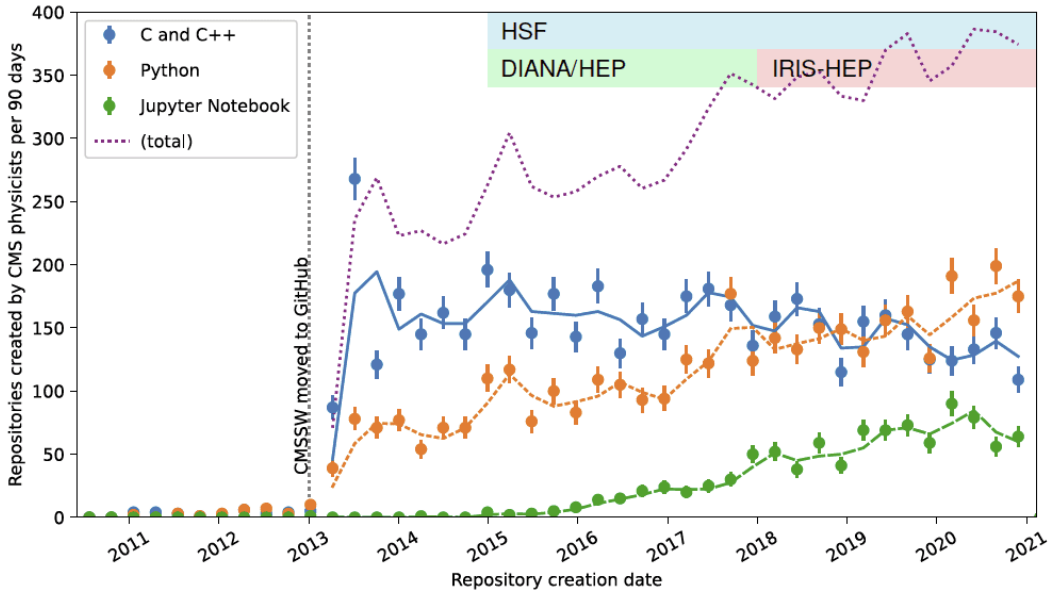


Fig. 3.19: Number of GitHub repositories created by CMS physicists by language (exclusive categories), that shows the rise of Python and Jupyter notebooks. [134].

- In HEP analyses, a columnar approach to data (typical of R and Pandas DataFrame model) is more and more visible.
- It has become common for HEP developers to describe their tools as "declarative". In computer science, a declarative language (e.g. SQL) describes calculations regardless of the order in which they would be calculated. In HEP, it could mean a separation of the event loop from the calculations performed on each event, as in ROOT's RDataFrame, or on a partition of events, as in Coffea's Processor.

Among data science tools, key players are ML frameworks and libraries. The use of ML in HEP analyses has become common over the past two decades [135]. In such period, the majority of HEP analyses that exploited ML have used the implementation of algorithms offered by the TMVA package included in ROOT, but recently many HEP analysts have started using non-HEP ML packages (e.g. Scikit-learn, XGBoost, Tensorflow, Keras, and PyTorch), confirmed by the growing number of published results based on externally developed tools. The use of non-HEP solutions leads to the necessity of adapting HEP data, commonly stored in ROOT files, to the external software, breaking analysis workflows and introducing difficulties in the analysis software development. To solve this issue two approaches were followed: both converters have been written to integrate some externally trained models into HEP tools (e.g. [138]), and interfaces between HEP and external tools have been developed [139].

A great advantage of using external ML tools is the size of the community that uses and supports them, being able to easily keep up with industry advances and profit from the cutting edge of ML research. Moreover, the industrial efforts to develop and maintain ML tools rely on resources far beyond that of basic research. As disadvantages, there is no guaranteed support for external tools over the lifetime of HEP experiments, and it can be difficult to adapt such tools to HEP-specific requirements that may not be among the

	TMVA	TensorFlow	Theano	Scikit Learn	R	Spark ML	VW	libFM	RGF	Torch
ROOT [**T, C**]	✓	through conversion into other formats								
CSV [**F**]		✓	✓	✓	✓	✓	×	×	×	✓
libSVM [**M**]							×	✓	×	
VW [**M**]							✓			
RGF [**M**]									✓	
NumPy [**R**]	×	✓	✓	✓	✓	✓	×	×	×	✓
Avro [**S, R**]					✓	✓				
Parquet [**S, C**]					✓	✓				
HDF5 [**S**]		×	×	×						✓
R df [**R**]					✓					

Fig. 3.20: The table lists various ML tools (columns) and data formats (rows). The × indicates that a native solution exists, while ✓ indicates that the conversion from one data-format to another is possible. The notation used for data-formats is the following: **T** trees, **F** flat tables, **M** sparse matrices, **C** column-wise arrays, **R** row-wise arrays, **S** static data structures. The table is updated to 2019 [139].

priorities of the ML community. Instead, the use of internal tools allows to keep decisions about long-term support within the community, and the tools can be adapted to the specific needs of HEP. On the other side, to include any new algorithm or idea it should be ported before it can be used and evaluated.

An approach for HEP users is to use TMVA, also because also interfaces between TMVA and common external ML tools have been built. This solution has the advantage of allowing the use of a homogeneous interface which requires a little training overhead for those who already have knowledge in TMVA. However, many researchers prefer to convert their data into the formats used by external tools and work exclusively with them (see Fig. 3.20 for the file formats that common ML tools support). This has the advantage of working as close as possible with the tools of the ML community and their documentation. Middleware solutions have been developed to provide such conversion, e.g. PyROOT, root_numpy, Uproot, and root2hdf5.

The training of ML algorithms takes a considerable amount of time and parallelization at various levels is desired, such as parallelizing computations within a single model. Another type of parallelism is data parallelism, which targets the processing phase of the training with data partitioning and model training using distributed workers. Frameworks like Apache Spark and ideas such as batch training offer good solutions for these tasks. Moreover, CERN developed its interactive framework called Service for Web Based Analysis (SWAN), which may play an important role in the adoption of ML tools in HEP workflows. It allows for rapid prototype development and testing of ML tools, and provides a straightforward means to visualize models and data.

3.4.2.1 *Towards Machine Learning as a Service for High Energy Physics*

MLaaS solutions are not yet widely used in HEP, although various R&D activities are underway within HEP aimed at providing HEP analysts with tools or services to accomplish ML tasks [140].

For instance, the hls4ml project targets ML inference on FPGAs, whereas the SonicCMS project is designed to provide Services for Optimal Network Inference on Co-processors.

Both aim at optimizing only the inference phase rather than the entire ML pipeline, i.e. from reading data to training models to serving predictions.

Another solution uses the Spark platform for data processing and ML training [141]. Although it seems very promising, it needs data ingestion into the CERN EOS file system or HDFS/Spark infrastructure, therefore data located at WLCG sites or outside of such dedicated infrastructure cannot be easily accessed. Moreover, Spark- based libraries (BigDL and Analytics Zoo) may be required on top of Keras APIs, thus limiting the flexibility of choosing the ML framework on the user side.

In a CMS work [142], a DNN used for a jet tagging algorithm relies on the TensorFlow queuing system with a custom operation kernel for reading ROOT trees and feeding them into a NN written in Keras. It represents an interesting approach for a specific use case, but it is not an "as a Service" solution.

In 2021, a CERN project made available a solution for training and serving ML workloads with Kubeflow, which runs in production in the private cloud of CERN based on OpenStack [143, 144]. This solution is based on Kubeflow, an open source ML platform built on top of Kubernetes, that provides components for all the required ML steps: data loading and pre-processing, distributed model training, storage and versioning, and lastly model serving. Each component is containerized and managed as a microservice.

3.4.3 *The future of HEP analysis tools*

The rise of the tools described in Sect. 3.4.2 shows that there has been a change in the software used by the HEP world in recent years. However, this does not mean that the trend will continue to rise: there is good reason to believe it will smooth out, with physicists using Python as an interface whereas C++ for performance and accelerator access (e.g. GPUs), freely mixing HEP-specific ROOT routines with external ML solutions and other tools from the data science world. In the following, we discuss what changes are expected for different sectors.

3.4.3.1 *File formats*

History teaches us that file formats tend to be conservative. Thus, it is expected that ROOT files will continue to play a major role in HEP. For many years, ROOT TTrees had been unique in their ability to efficiently store (usually columnar) nested, variable-length data structures with a direct interpretation in C++. Now, there are also Apache Arrow and Parquet that can efficiently represent (always columnar) nested data structures in memory, in a language-independent way. As non-domain-specific formats, they are also recognized in all scientific fields and are supported by interdisciplinary libraries like Scikit-Learn, TensorFlow, and Pandas. Looking at the HEP world, ALICE's O2 analysis framework uses Arrow as its primary data model for Run 3.

This does not mean that all HEP experiments will switch from ROOT to Parquet, but only that analysts prefer to mix ROOT, Arrow, Parquet, and HDF5 in their analysis workflows, and so a lightweight software is needed to accommodate that. Uproot is a Python

implementation of ROOT I/O well established among HEP physicists, capable of reading and writing TTree data that are sufficiently independent of C++ and allowing conversion to Parquet/Arrow format. Uproot, by using Awkward Array, avoid the creation of Python objects for every entry of a TTree during conversion, which affects the conversion rate by a factor of hundreds [145].

Moreover, ROOT data will not always be TTrees. Indeed, as we said in Sec. 3.4.1, the ROOT team is developing RNTuple as a replacement for the TTree class, which addresses TTree's shortcomings compared to Parquet. To guarantee the same fluency in the conversion between RNTuples and Arrow/Parquet as we currently have between TTrees and Arrow Parquet, more effort will be needed in this area.

3.4.3.2 *Databases*

Typically, HEP collaborations centrally produce datasets with basic physical quantities stored as primitive types. Many analyses share some subsets of such quantities, but each analysis may also require some specific variables that are not computed by default. A database would allow to store and access each variable independently, eliminating any duplication of disk space, processing, and effort. Each collaboration could centralize its analyses data into a global, federated, and extensible database, with automatic replication, provenance tracking, versioning, caching, and improved metadata handling. Some projects are working in this direction, e.g. Striped, ServiceX, SkyhookDM, and Coffea.

3.4.3.3 *Distributed computing*

Data analytics software products of the industry tackle the problem of scaling to large datasets, aka big data, and common examples are Apache Spark and Hadoop. In HEP, the Coffea developers experimented with big data scale-out mechanisms, including Spark and Dask. Of these, Dask has been the most successful so far, with more analyses that have opted to use Dask as backend. RDataFrame is a good interface to organize and distribute tasks written in C++ (on Spark and Dask).

High-throughput data processing problem should be tackled with open source industry tools. Three IRIS-HEP projects (i.e. ServiceX, SkyhookDM, and coffea-casa) use generic data science tools to build HEP-specific workflows. They mix Docker, Kubernetes, Helm, Flask, Kafka, and other tools alongside ROOT, XCache, Rucio, and Uproot to deliver columns of data to the analyses as Arrow or Awkward Array buffers, Parquet, or ROOT files. A similar solution is the one adopted by the R&D project INFN Analysis Facility, which uses Dask to distribute RDataFrame payloads, and offers the possibility to be used as an HTCondor batch system or through a JupyterHub interface[146].

3.4.3.4 *Acceleration*

It might surprise that Python emerged as a language used also for large-scale data analysis, particularly ML, despite having many language features that prevent fast computation. Developers have learned to split programs into a fast and simple part in a compiled extension,

and a slow and complex part in pure Python. However, data analysts in general need low barriers to optimizing their code. Numba provides the lowest barrier to compilation, as it just-in-time compiles the Python code for CPU and GPU backends. Many HEP packages use Numba to accelerate Python code, including Scikit-HEP's Awkward Array. Nevertheless, Numba has some drawbacks which do not always make it an optimal choice. The Julia language would be ideal for HEP (small groups of physicists investigated Julia for years), as it combines Python's dynamism with C++'s speed, but lacks the very large worldwide community that Python and C++ have. This situation could result in the development of a HEP language on the HL-LHC's timescale.

3.5 MACHINE LEARNING IN HEP

The data collected by HEP experiments is complex and high dimensional. Typically in HEP data analyses, a sequence of boolean decisions is performed to select the data of interest, followed by a statistical analysis [147]. These operations are done by plotting the distributions of single observed quantities and are motivated by physics considerations, that cannot be easily extended to higher dimensions. For several decades, physicists have tried to improve their analyses by exploiting algorithms that utilize multiple variables simultaneously. In HEP, this approach is often referred to as multivariate analysis (MVA), but however outside of HEP this is considered an example of ML. Physicists have used different ML algorithms, nevertheless the Boosted Decision Trees implemented in the software package TMVA have been the most common choice. These tools provided an important solution for many data analysis tasks (just think that the analyses that led to the discovery of the Higgs boson by the CMS and ATLAS collaborations used Boosted Decision Trees), but it was understood that their capabilities were limited: they often failed to match the performance of traditional solutions, especially when the dimensionality of data gets large. In recent years, there has been an increasing use of ML techniques in HEP analyses, thanks to the availability of ML tools beyond TMVA and the rise of deep learning. Indeed, it was possible to enable the training of very large NNs that greatly outperformed the previous state of the art, in order to handle higher-dimensional and more complex problems than previously possible. There are different types of DNN used in HEP: MLP, CNN, RNN, and Graph NN (GNN) [139]. Additionally, NNs are used in generative models, where a NN is trained to reproduce the multidimensional distribution of the instances in the training set. Variational AutoEncoders (VAE) and Generative Adversarial Networks (GAN) are two examples of such kind of models used in HEP.

3.5.1 *Machine Learning applications in HEP*

A useful collection of papers where ML approaches have been applied in experimental, phenomenological, or theoretical analyses is provided by the "Living Review of Machine Learning for Particle Physics" [148]. The breakdown of papers reported in this collection

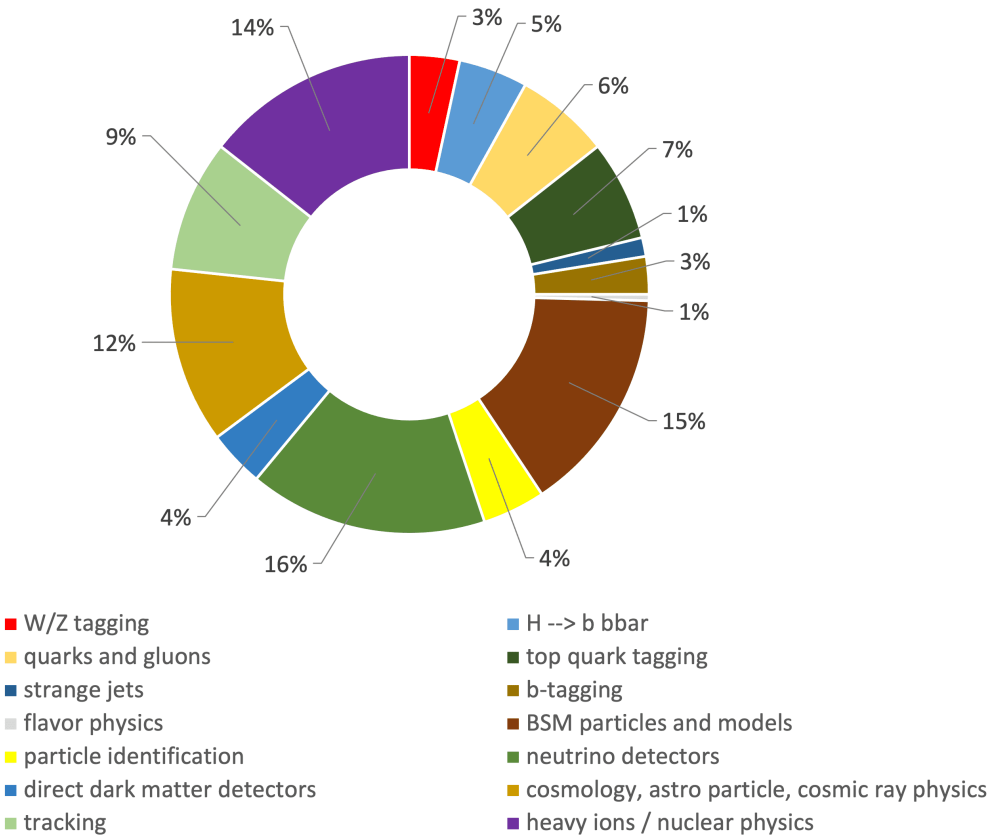


Fig. 3.21: Plot showing the breakdown of 236 papers collected in the "Living Review of ML for Particle Physics" [148] for different areas of data analysis, where ML techniques have been used to tackle classification problems.

for different areas of data analysis about classification and regression tasks is shown in Fig. 3.21 and Fig. 3.22 respectively.

In the following, we provide details about some areas of application of ML techniques in HEP, and where they can play a significant role in advancing the current state of the art.

3.5.1.1 *Event selection*

ML techniques for classification purposes have a great application in extracting small signals with complex topologies from huge backgrounds (event selection) as happens at the LHC experiments. In the past few years, many studies have shown that traditional shallow networks (consisting of 1 or 2 hidden layers) using physics-inspired engineered ("high-level") features, are outperformed by DNNs based on the higher-dimensional features which receive less pre-processing ("lower-level"). Before the advent of deep learning, such pre-processing was necessary as shallow networks performance with low-level features fell short.

An early study [149] compared the performance of shallow networks and DNNs in distinguishing a cascading decay of new exotic Higgs bosons from the dominant background. This study used a dataset in which a large set of basic low-level features were reduced to a smaller set of physics-inspired high-level engineered features. It was found that DNNs using the lower-level features significantly outperformed shallow networks

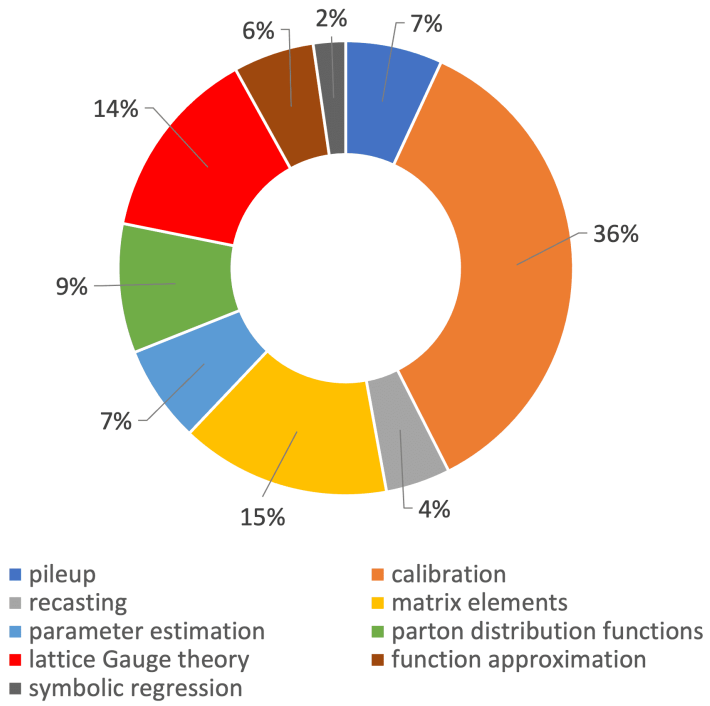


Fig. 3.22: Plot showing the breakdown of 87 papers collected in the "Living Review of ML for Particle Physics" [148] for different areas of data analysis, where ML techniques have been used to tackle regression problems.

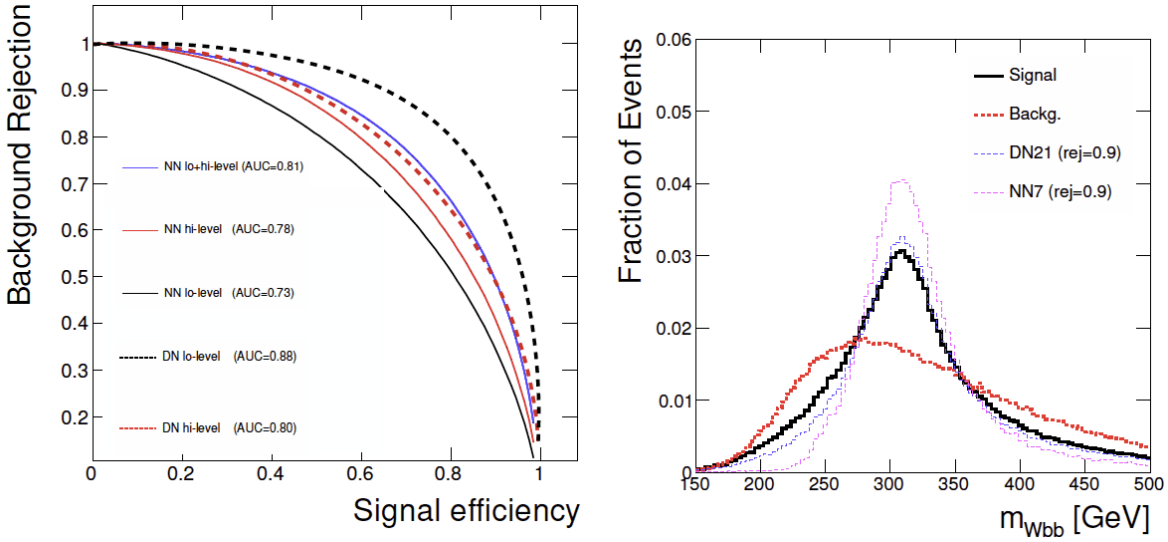


Fig. 3.23: In the plot on the left, DNNs performance in signal-background classification compared to shallow networks (NN) with a variety of low- and high-level features is shown. In the plot on the right, the comparison of the distributions of invariant mass of events selected by a DNN (DN21) using only object momentum to a shallow network (NN7) trained using this feature at equivalent background rejection, is shown [149].

that used physics-inspired features such as reconstructed invariant masses (see Fig. 3.23). The high-level engineered features captured real insights, but at the same time sacrificed some useful information.

Another approach can be found in [150], where a GNN was implemented for classification and reconstruction in the IceCube detector. Here the events were represented as point

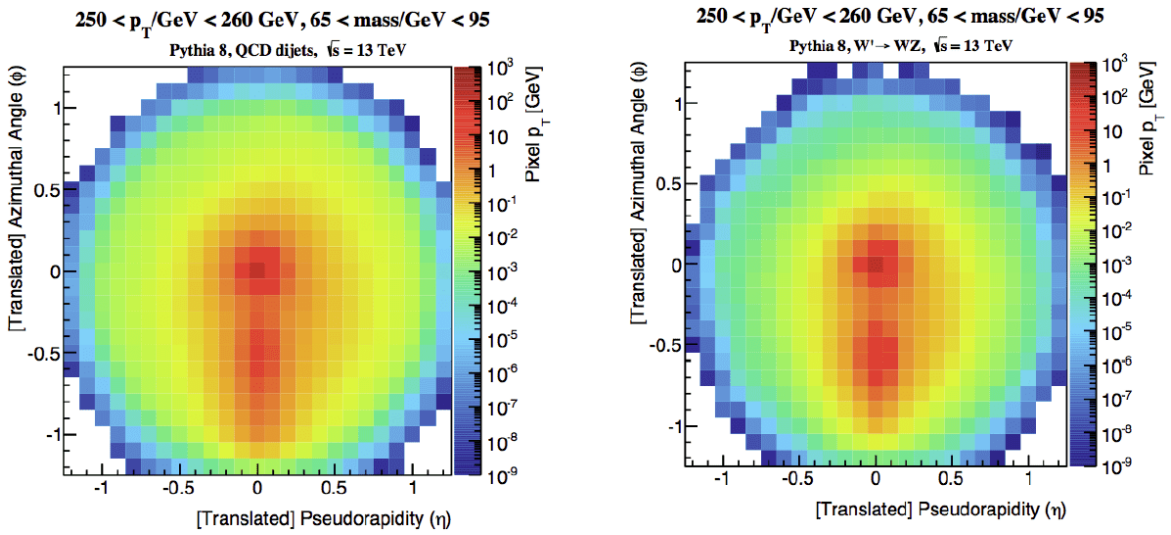


Fig. 3.24: Examples of jet image inputs from the jet substructure classification problem described in [151]. The background jets (plot on the left) are characterized by a large central core of energy deposited by a single hard hadronic parton, whereas the signal jets (plot on the right) tend to have a subtle secondary deposition due to the two-prong hadronic decay of a vector boson with high-p_T. The use of image-analysis techniques, such as CNNs, allows for powerful analysis of this high-dimensional input data.

cloud graphs and the GNN was capable to distinguish neutrino events from cosmic-ray backgrounds, classify different neutrino event types, and reconstruct the deposited energy, direction, and interaction vertex. For neutrino event classification, the GNN increased the signal efficiency by 18% at a fixed background rate, compared to current IceCube methods that use BDTs.

3.5.1.2 *Jet classification*

ML has been applied to a wide range of jet classification problems, in order to identify jets from heavy (c, b, t) or light (u, d, s) quarks, gluons, and W, Z, and H bosons. Traditionally these classification problems have been grouped into flavor tagging (which discriminates between b, c, and light quarks), jet substructure tagging (which discriminates between jets from W, Z, t, and H), and quark-gluon tagging. In 2014, a study [151] recognized that the projective tower structure of calorimeters present in almost all modern HEP detectors was similar to the pixels of an image (see Fig. 3.24). This representation of data allowed physicists to leverage advances in image classification such as CNN. While the image-based approach has been successful, the actual detector geometry is not perfectly regular. Therefore, some pre-processing is required to represent the jet as an image. Both ATLAS and CMS have since commissioned flavor-tagging NNs that rely on individual tracks or, in the CMS case, particle-flow candidates. For example, CMS's DEEPFLAVOR first embeds each flow candidate with a shared transformation, and then combines the high-level variable candidates in a single dense network.

3.5.1.3 *Track and event reconstruction*

Track-reconstruction algorithms are among the most CPU and data intensive of all low-level reconstruction tasks. The initial stage of track reconstruction involves finding hits (or points) where some charge is deposited on a sensing element. In the pixel sensors that form the innermost layer of the detector, neighboring hits are clustered into pixel clusters which then form track seeds. These seeds form a starting point for a Kalman filter, which extends the seeds into full tracks that spread up to the calorimeters. The whole procedure can be viewed as a sequence of clustering algorithms. In cases where multiple tracks pass through the same pixel cluster, ATLAS uses NNs to return a measurement for each track rather than assigning each to the cluster center.

Thanks to these algorithms and careful tuning, track reconstruction is nearly 100% efficient and incorrectly reconstructed tracks are rare, meaning that the clustering aspect of tracking is largely solved. However, reducing the CPU overhead remains a significant problem, especially within high-level trigger farms. Within ATLAS and CMS, these are clusters of $\mathcal{O}\left(10^4\right)$ processors that must reconstruct $\mathcal{O}\left(10^5\right)$ events per second. To keep CPU costs manageable, the experiments reconstruct tracks only in limited regions of the detector. Such regions are selected on the basis of their proximity to muons or to calorimeter energy deposits which are consistent with relatively rare physical signatures, e.g leptons or high-p_T jets.

In the CMS experiment, a Particle Flow (PF) event-reconstruction algorithm [152] has been deployed to identify and reconstruct individually each particle arising from the LHC pp collision by combining the information (i.e. tracks and calorimeter clusters) from all the subdetectors. In general, a given particle gives rise to several PF elements in the various subdetectors, and the reconstruction of a particle proceeds with a link algorithm that connects all the PF elements. The link algorithm is limited in terms of CPU time and is thus restricted to the nearest neighbors (tracks and clusters). ML-based reconstruction approaches using GNNs have been proposed for PF reconstruction. For example, in [153] a novel end-to-end trainable, machine-learned particle-flow algorithm (MLPF) has been introduced. It is based on a parallelizable, computationally efficient, and scalable GNN optimized using a multi-task objective on simulated events. Details about the implementation of a similar MLPF algorithm for CMS can be found in [154]. A good correspondence between the MLPF algorithm and the PF was found both at the particle and object level. A positive aspect of the MLPF model is that it runs natively on a GPU and has an approximately linear scaling of runtime and memory with increasing particle multiplicity.

3.5.1.4 *Fast inference on designed hardware*

With the start of the HL-LHC phase, the instantaneous luminosity of the LHC is expected to increase up to $\approx 7.5{\cdot}10^{34}\mathrm{cm}^{-2}\mathrm{s}^{-1}$. Therefore, new strategies for data acquisition and processing will be needed, in preparation for the higher number of signals produced within the detectors [155]. Following the rapid rise of ML through deep learning algorithms, the study of processing technologies and strategies to accelerate deep learning and inference

is well underway. In particular, much effort has been made to convert learning models into specific firmware code [156], capable of running on fast hardware like Field Programmable Gate Arrays (FPGAs).

In CMS, new reconstruction algorithms are being developed, aimed at better performance. For example, regarding the tracking of muons in the muon trigger, one of the figures that is being improved is the accuracy of the p_T measurement [157]. The implementation of ML models on FPGAs is beneficial for two main reasons. Firstly, these models are able to predict with improved precision the p_T, as they exploit much more information collected by the detector. Secondly, FPGA hardware promises lower latency than traditional inference algorithms running on CPU, which is an important aspect of a trigger system.

A study from the ATLAS collaboration concerns the implementation of a CNN in an FPGA to identify significant energy deposits in the Liquid-Argon (LAr) calorimeters [158]: in this way, the fast inference enabled by the hardware solution might be able to handle the enormous amount of signals coming from the HL LHC increased pileup.

3.5.1.5 *Fast simulation*

Simulation is the most intensive CPU operation in HEP. Therefore, a fast simulation is really valuable as the full simulators (which faithfully describe the low-level interactions of particles with matter) are very computationally intensive and consume a significant fraction of the computing budgets of experimental collaborations. A promising approach is based on GANs. The training of such a generative model (G) is accomplished through the competition with an adversary network (A). The task of G is to generate simulated samples, whereas A has the task to determine whether a given sample is from G or from the full simulator. The two networks are put against each other: A attempts to identify differences between the traditional samples and those generated by G, whereas G attempts to fool A into accepting its events and in doing so learns to mimic the original sample generation. However, the stability of a such training solution can be difficult to achieve and expert knowledge is usually required to construct an effective network. Currently, some GAN approaches are used in the simulation of electromagnetic showers in a calorimeter [159], giving computational speed-ups while achieving a reasonable energy deposition model. Similar approaches are also applied in the simulation of jet images [160].

3.5.1.6 *Monitoring of detectors and data quality*

LHC systems and detectors are complex machines that are equipped with monitoring systems. They constantly check that every parameter is in an acceptable range: from voltages to reconstructed masses from known decays. A major challenge for monitoring systems is that they must be able to recognize changes in data caused by equipment malfunctions. If an observed variable distribution is different from the corresponding reference, the operator investigates the discrepancy; if it has not been marked as previously addressed, the incident is recorded and the relevant expert is contacted. Additionally, the system is equipped with automatic alarms that go off when any large discrepancy is detected. False alarms occur when the reference is not updated in time and the discrepancy is caused

by a legitimate change in data taking conditions. A whole class of ML algorithms, called anomaly detection, can be useful to monitor the conditions of the detector and also predict eventual future anomalies. Some efforts have been already made for the CMS Data Quality Monitoring system of different subdetectors, using unsupervised learning methods, e.g. VAEs [161].

3.5.1.7 *Computing operations*

Data operations are one of the main challenges for the upcoming HL-LHC. The adoption of ML techniques in this field can help automate and improve the overall system throughput, and reduce operational costs. ML can be applied in many areas of computing infrastructure, workflow and data management.

For example, optimization of dataset placement and reduction of transfer latency can lead to better usage of site resources and an increased throughput of analysis jobs. One of the current examples is predicting the "popularity" of datasets (using information like the number of accesses to a given dataset, the number of users per day recorded for a dataset, and the total number of CPU hours spent on a given dataset) which helps reduce disk resource utilization and the time for physics analysis (see [162] for an application in CMS using ML techniques).

Data volume in data transfers is a current challenge for computing systems as thousands of users need to access thousands of datasets across the Grid. There is a huge amount of metadata collected by application components, e.g. information about failures and file accesses. The optimization of resource utilization based on this data, including Grid components and software stack layers, can improve the overall operations. Understanding the data transfer latencies and network congestion may improve the operational costs of hardware resources.

Networks are going to play a crucial role in data exchange and data delivery to scientific applications during the HL-LHC phase, and ML can help in several ways, e.g. identifying anomalies in network traffic or predicting network congestion.

As we reported so far, there are many areas where ML can help and it is currently helping computing operations. The Operational Intelligence (OpInt) project started as a joint effort of several HEP experiments to increase the level of automation in computing operations and reduce human interventions [163]. Technical and cross-experiment forums are organized where people share ideas and experiences with the aim of developing tools to automate computing operations, exploiting state-of-the-art technology where ML techniques represent a key component.

One of the challenges this project has faced is the predictive maintenance in data centers. In data centers, operations are many and of different types, managed by a set of services. Such services produce log files reporting, with different granularity, the status of the single activities that the services are managing. Log data is often unstructured, where the format and semantics may vary significantly from service to service, making an approach towards a general-purpose log-based anomaly detection system very challenging. Supervised ap-

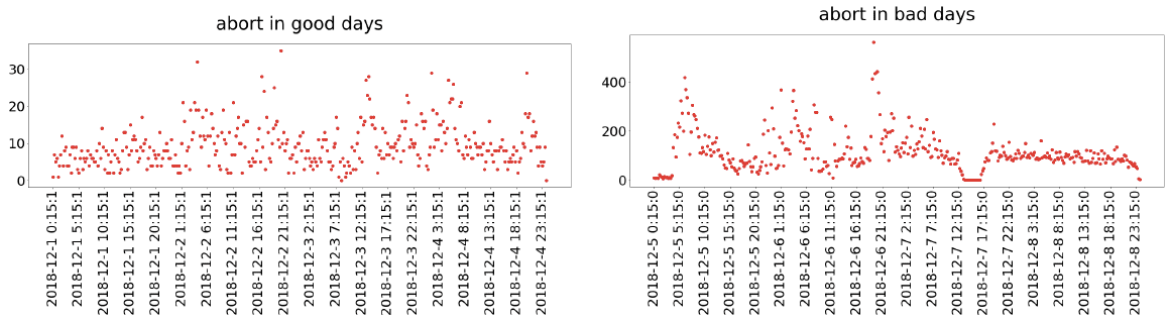


Fig. 3.25: Comparison of the number of abort requests between normal (good) days (plot on the left) and anomaly (bad) days (plot on the right) [164].

proaches provide more accurate results at the cost of requiring a dataset with labeled entries, which is a hard and restrictive requirement in the majority of real-world scenarios.

Supervised learning algorithms are used in [164] to provide a predictive maintenance solution at the INFN-CNAF computing center. In this work, log data for different components related to the SRM service is collected and analyzed for "normal" and "anomaly" periods of time. Text data is parsed through filters and then converted into numeric format through a One Hot Encoding procedure. Subsequently, such data is used as input for training several supervised ML models, which then are able to distinguish data coming from a normal or anomaly period of time. For example, Fig. 3.25 shows in normal and anomaly periods of time, the trend of the feature representing the number of abort requests. This feature was previously identified using different approaches (PCA, feature importance, Recursive Feature Elimination and chi-squared statistical test) as the one with the highest discriminating power between normal and anomaly data.

In the case of unlabeled data, other techniques for anomaly detection (e.g. isolation forest and SVMs) have been adopted in other solutions. Moreover, automated log parsing offers promising results through NLP algorithms.

3.5.1.8 *Sustainable Matrix Element method*

Sustainable Matrix Element Method (ME) is a powerful technique which can be used for making measurements of physical model parameters and direct searches for new phenomena. It has been used extensively by collider experiments at the Tevatron for Standard Model measurements and Higgs boson searches, and at the LHC for measurements in the Higgs and top quark sectors.

The biggest difficulty in the ME method that has limited its applicability to searches for BSM physics and precision measurements is that it is very computationally intensive. If this limitation were overcome, it would allow for a more widespread use of ME methods for LHC data analysis, particularly during the HL-LHC phase. Although the use of NNs for numerical integration is not new, it is a technical challenge to design a sufficiently rich network to encode the complexity of the ME calculation for a given process over the phase space relevant to the signal process. DNNs are strong candidates for networks with sufficient complexity to achieve good approximations [165]. Promising demonstration of

the power of Boosted Decision Trees and GANs for improved Monte Carlo integration can be found in [166].

3.5.2 *Collaborating with other communities*

Establishing a vibrant collaboration between ML and HEP communities with the aim to further science has benefits for both. The HEP community can explore new research directions and applications of ML, novel algorithms, and find direct support for HEP challenges. At the same time, the ML community can benefit from a diverse set of particle physics problems with unique challenges in terms of scale and complexity, contributing to solving problems relevant to both communities. For example, the treatment of systematic uncertainties is an important topic both for HEP and ML communities, and by working together they can further progress in solving such problem. There are a number of existing examples of such kind of collaboration that have produced fruitful results through mostly local connections. To obtain the maximum profit from these collaborations the HEP community needs to define its problems in a language that the ML community can understand, providing necessary information with clear and concise explanations. At the same time, since the ML community has a significant amount of domain knowledge, the ideas and solutions it provides should be presented in an understandable way for scientists without in-depth knowledge.

There are different ways to promote and search for collaboration, and such collaborations should be searched not only within the ML community, but also within the scientific community itself.

- Conferences and workshops are a key aspect of the academic ML community, and organizing or contributing to key conferences is a means of gaining interest. Organizing mini-workshops or sessions within major ML conferences, e.g. NeurIPS [167], would increase the familiarity of HEP within the ML community and help build future collaborations.

- To engage the wider ML community, challenges such as the Higgs Boson ML challenge, the Flavor Physics challenge, and the TrackML Particle Tracking challenge have been organized on Kaggle. These types of challenges attract considerable attention from the ML community and additional similar challenges should be organized in the future.

- In HEP, there is a strong incentive to make public benchmark datasets, beyond just challenges. Within the HEP community, common datasets enable comparisons of algorithms and techniques, and this is very useful for R&D. Such datasets can also be used for teaching, tutorials, and training. The CERN Open Data portal provides more than two PB of open data from particle physics [168].

- Industry developed and adopted ML techniques, as well as it adopted dedicated specialized hardware and high-performance co-processors. Wide use of GPUs, FPGAs,

and high core count co-processors would dramatically increase the performance of ML applications relevant to the HEP community. Interactions with the industry bring specific technology opportunities and access to specialized skills that can be difficult to hire and support internally. CERN OpenLab [169] is a public-private partnership that accelerates the development of cutting-edge ICT solutions for the worldwide LHC community and broader scientific research. Through CERN Openlab, CERN collaborates with the leading ICT companies and research institutes.

- Many communications mediums can be used to broadcast HEP challenges and attract interested collaborators, e.g. popular forums like reddit, social media, personal or official blogs, or direct contact with influential personalities.

- As each HEP experiment has different specific use cases, probably many of these are sufficiently similar to each other that R&D can be done in common. Even when this is not possible, experience with one type of problem can provide insights into how to approach other problems. This is why the Inter-experiment ML forum (IML) was created at CERN in 2016 [170].

- HEP should reach out to other scientific communities that have similar challenges, e.g. astrophysics/cosmology, computational biology, and medium energy nuclear physics. This can lead to more active partnerships in order to collaborate on ideas, techniques, and algorithms.

3.5.3 *Computing and hardware resources*

At present, training of ML algorithms is mostly done using dedicated or private resources. In order to try complex ML algorithms, more computing power is needed in both the training and evaluation phases. This implies the expansion of the current computing model to include architectures that are well suited to ML tasks, e.g. Many Integrated Core (MIC), GPUs, and Tensor Processing Units (TPUs). These architectures provide a significant computational speed improvement for both training and evaluation of ML algorithms, but at the same time require dedicated software configuration, hardware, and drivers. Moreover, the bandwidth and locality of large data stores will need to be optimized to avoid bottlenecks. Data placement and the need for dedicated hardware indicate that a transition to HPC (or HPC-like) architectures may be required to achieve the desired performance.

The potential of deep learning methods is largely due to the ability to train these models in a reasonable amount of time with large-scale parallelism. The training phase needs repeated simultaneous access to many data elements and specialized hardware has been developed to train deep learning models. The speed-up of the training process can be achieved with faster and more capable hardware, parallelization of single training, and splitting training over multiple nodes. Instead, inference can be an operation applied to a single data element at a time and is performed only once. Inference requires less I/O and is limited only by computing power and model complexity. As inference has real-time applications in HEP, throughput and latency constraints are the main challenges. For

inference, FPGAs are used in HEP experiments, e.g. in the trigger system of CMS, since they provide an efficient and low-latency application of ML algorithms directly at the level of hardware.

Many researchers in HEP are currently relying on private or university GPU clusters to perform ML training. To support the growth of Deep Learning research for LHC physics, CERN should invest in a cluster of at least a few hundred GPUs [171]. CERN is currently providing HEP researchers with several solutions for using GPUs. The easiest way is to access via SSH to `lxplus-gpu.cern.ch`, but it has limitations: there is no guarantee of exclusive access to a node, no access to multiple GPUs at once (it is a problem for advanced works with big models), there are some issues with Singularity containers, the memory-usage is limited (jobs are killed when threshold exceeded), and it needs some ssh-keep-alive setup (not a friendly entry point and quite annoying). Another solution is to submit a job to the CERN batch service (using the HTCondor platform), but the cluster is very small with many users for too few GPUs, and the GPUs do not communicate with each other. Recently, SWAN made available a GPU server integrated into the Spark server, but currently, it is not clear how many GPUs will have available. Another promising solution is represented by the Kubeflow project at CERN [143], but currently, the access is limited to one or two GPUs per (few) users. Since the GPUs solutions currently available at CERN are not always sufficient to tackle ML projects, opportunistic resources can be also explored.

ML algorithms significantly benefit from the use of hardware accelerators but the risk is that ML users would be hindered in developing new applications by writing platform-dependent code. Indeed, various interfaces to different hardware architectures are required to efficiently use the available computing resources. The Open Computing Language (OpenCL) allows for the programming of high-level interfaces that can run on various hardware platforms. Moreover, ML tools often provide different sets of APIs to develop and train the models in one language, and various bindings to use trained models in other programming languages.

Regarding other types of resources, current efforts in the HEP community are studying the use of Cloud TPUs for a possible acceleration of the training stage. Moreover, there are efforts to bring and expand the availability of HPC resources in HEP.

4

THE MLAAS4HEP SERVICE

As discussed in Sec. 3.5, ML is nowadays successfully used in many areas of HEP and will play a significant role also in Run 3 and HL-LHC.

Building and developing a ML project and implementing it for production use requires specific skills and is a highly time-consuming task [95]. Generally, HEP physicists do not have the skills in data science to tackle such challenges on their own, while they are mainly focused on HEP, data analysis (including statistics), and whose ultimate goal is to work for a physics publication. Addressing the need to improve a physics data analysis and understanding that ML could be an interesting exploration is therefore only a first step towards actually adopting ML in an analysis. The existence of such a gap between "two different worlds" (HEP data analysts and ML community) is not so easy to bridge, although currently there are many activities, collaborations, and organizations that are working towards this direction (see Sec. 3.4.2 and Sec. 3.5.2). It would be helpful to provide HEP physicists who are not experts in ML with a service (and in particular a MLaaS) that allows them to exploit the potentiality of ML easily: such a solution would help to bridge the aforementioned gap and to have wider use of ML techniques in HEP analyses.

MLaaS solutions have been discussed in detail in Sec. 2.2.2.1. MLaaS is used as an umbrella of various ML tasks such as data pre-processing, model training and evaluation, and inference available through REST APIs [140]. Major IT companies offer their customers MLaaS solutions, which most of the time cover standard use cases, e.g. image classifications, natural language processing, and computer vision. Although a custom ML code can be provided to these platforms, its use in HEP is quite limited for several reasons. For example, the ROOT data format cannot be used directly in any service provider's APIs. Therefore, the operational cost (e.g. data transformation from ROOT files to the data format used by the MLaaS provider APIs), the data management, and data pre-processing can be significant for large datasets. The flattening of data from the dynamic size event-based tree format to the fixed-size data representation does not exist in MLaaS solution. The ROOT data format is used to store HEP events in TTrees, which have tree-based data structures where the size of individual events cannot be determined a-priory (e.g. the number of electrons can vary in each event). Instead, most of the existing ML algorithms rely on a fixed-size data representation of individual events, so the event-based data structures cannot be directly fed to ML frameworks, and special attention must be paid either at the framework or at the data input level. In conclusion, off-the-shelf commercial solutions most of the time are not applicable or are ineffective for HEP use cases (in terms of cost and

functionality). This might change in the future, as various initiatives (e.g. CERN OpenLab) continue to work closely with the most common service providers.

At the same time, there are various R&D activities underway within HEP aimed at providing HEP analysts with tools or services to accomplish ML tasks. We reported some of them in Sec. 3.4.2.1. Nevertheless none of these represent a MLaaS solution for HEP, although the solution recently offered by CERN based on Kubeflow represents a valid service towards MLaaS for HEP.

As there was no final product in 2017 that could be used as MLaaS for HEP and that could cover the entire ML pipeline (in terms of reading data, processing data, training ML models, and serving predictions), an R&D project within CMS started with the aim of providing such an service. A first prototype of such a solution can be found in [95, 172]. These works describe an application on the signal-background discrimination in the fully hadronic $t\bar{t}$ decay at the CMS experiment (see 3.1.1). At that time the project was at the very beginning and the prototype covered only the inference phase. Nevertheless, this application on a specific use case showed the usefulness of a solution allowing HEP analysts to use ML models external to TMVA.

The MLaaS solution proposed in [140] consists of two individual parts. The first part, the Machine Learning as a Service for HEP (MLaaS4HEP) framework [173], covers the data reading, data processing, and ML model training phases, in a completely model-agnostic fashion, directly using ROOT files of arbitrary size from local or distributed data sources. And, the second part, the TensorFlow as a Service (TFaaS) framework [174], can be used to host pre-trained Tensor-based ML models and obtain predictions via HTTP calls.

This chapter, which contains the original contribution of the thesis, is totally focused on the development of the MLaaS4HEP framework, and on the creation of a cloud service that can be used to submit MLaaS4HEP workflows via HTTP calls. The description of the TFaaS service is only reported for the completeness of the project.

In particular, in Sec. 4.1, the overall architecture is described. In Sec. 4.2 the application of MLaaS4HEP on a real physics use case for its validation and test of performance is discussed. In Sec. 4.3, additional studies and developments of the framework are described. In Sec. 4.4.1, the work done with DODAS to automatize the deployment of a platform where directly run MLaaS4HEP without any effort by the user is shown. In Sec. 4.5 the work done for the realization of a working prototype of a cloud service for MLaaS4HEP is presented. Lastly, in Sec. 4.6 final considerations on the project and future plans are reported.

4.1 OVERALL ARCHITECTURE

MLaaS4HEP provides transparent access to HEP datasets stored in the event tree-based ROOT data format into existing Python-based ML frameworks of the user's choice, which usually are designed to operate with row-based data structures, e.g. NumPy arrays, CSV files, and alike. It is based on the Uproot library and XRootD protocol to read small or large tree-based ROOT files from local filesystems or remote sites. MLaaS4HEP has a

modular design that opens up the possibility to train ML models on PB-size datasets remotely accessible from WLCG sites without requiring data transformation to the user, i.e. from ROOT data format to flat data format and subsequent storage of data to be used by underlying ML framework. The caveat on the accepted input ROOT files is that their content is in the form of a flat ROOT TTree, where no C++ object is stored inside, and no further array nesting within branch elements is present. These conditions are fulfilled by the typical user ntuples used for analysis and by the CMS NanoAOD data tier (see Sec. 3.3.2), which represents a common data tier in CMS and is already used by many analyses (currently more than 30%). MLaaS4HEP transforms the Jagged Array representation of ROOT data and feeds it into the ML framework via vector transformations applied to the I/O stream. This opens up a possibility to use favorite non-HEP ML frameworks, train ML models using distributed datasets, and therefore, attract non-HEP ML practitioners to be engaged in HEP ML activities.

The TFaaS framework is independent of MLaaS4HEP. It provides access to any kind of Tensor-based ML models to make inference via HTTP protocol. Although similar functionalities exist in various industry solutions, most of them are integrated as part of their service stack which may not be affordable or accessible to research communities, where an efficient and scalable open-source alternative is desired.

The proposed modular architecture can be easily adapted to any HEP experiment either as an entire pipeline or used partially, without requiring changes to existing frameworks or infrastructure.

4.1.1 *MLaaS4HEP architecture*

The MLaaS4HEP framework has been implemented using the Python programming language and the code is available in the GitHub repository [173].

As mentioned above, MLaaS4HEP allows the streaming of data from local or remote data storage. The development of the reading part has been done using the Uproot library (version 3 [175]). The Uproot library uses NumPy calls to quickly cast blocks of data in ROOT files as NumPy arrays and among its features it allows a partial reading of ROOT TBranches, non-flat TTrees, and histograms. It relies on data caching and parallel processing to achieve high throughput, and data can be read from local ROOT files or remotely via the XRootD protocol. A Python Generator has been implemented in MLaaS4HEP, capable of reading chunks of data from either local or remote file(s), giving as output a NumPy array with flat and Jagged Array attributes. Such an implementation provides efficient access to large datasets, since it does not require loading the entire dataset into the RAM of the training node. Additionally, it can be used to parallelize the data flow into the ML workflow pipeline. The choice of chunk size should be determined by the complexity of the processed events, the available network bandwidth, and the hardware resources.

Subsequently, MLaaS4HEP takes care to transform HEP ROOT data presented as Jagged Array into a flat data format used by ML frameworks. The Jagged Array is a compact rep-

resentation of variable size event data produced in HEP experiments. The HEP tree-based data representation is optimized for data storage but is not directly suited for ML frameworks. Therefore, some data transformation is required to feed tree-based data structures into the ML framework as a flat data structure, and a vector representation with padded values has been implemented in MLaaS4HEP (see Fig. 4.1).

A priori we may not know how many particles are created in a physics event and at the same time how much space is required to be allocated for particle attributes. Therefore, care should be taken to flatten and pad ROOT events in the Jagged Array representation. For that, a two-passes procedure has been implemented. In the first pass across all the events, the maximum dimensionality of each Jagged Array attribute and the min/max values of each attribute are determined. Even if this procedure may not be feasible for very large datasets (at TB or PB scale), it can easily be replaced by alternative approaches with approximate min/max and clipping procedures. In the second pass, the Jagged Array attributes are mapped into a single vector representation with appropriate size (the maximum dimensionality computed for each attribute) and padding (e.g. using NaN values or zeros), see Fig. 4.1. Furthermore, a proper normalization of each attribute is provided during this phase. This layer can be easily abstracted as a Python decorator to allow for multiple implementations of the normalization procedure that can be also provided directly by the user. A separate masking vector is also saved to distinguish the assigned padded values from the real values of the attributes. This masking vector may be important in some types of NNs, e.g. Autoencoders, where the position of padded values in the input vector can be used in the decoding phase.

In this chapter, the term data pre-processing refers to data transformation from Jagged Arrays to flat NumPy arrays with fixing of the dimensions, and data normalization.

Finally, the MLaaS4HEP framework uses data chunks with the proper proportion of events presented in the input ROOT files to train the ML algorithms defined by the user code (provided externally). MLaaS4HEP has been tested using MLPs written in Keras, but in any case, it is abstracted to support any kind of Python-based ML framework and algorithm (with a little effort in adjusting the code). Moreover, with the current implementation, it can be used to tackle only classification problems. In the following, the ML training workflow implemented in MLaaS4HEP is described using the terminology of NNs.

Three parameters, fixed a priori by the user, are used to control the data flow within MLaaS4HEP. The N_{chunk} parameter controls the chunk size of data read from local or remote storage, the N_{batch} parameter defines the batch size, and N_{epochs} parameter defines the number of epochs. The N_{batch} and N_{epochs} parameters are used by the underlying ML framework to control the training phase of the model using N_{chunk} events. The schematic of the data flow used in the MLaaS4HEP workflow is shown in Fig. 4.2.

The first pass (indicated by ① in Fig. 4.2) represents the part where the specs file is created. This part is performed by reading all the ROOT files in chunks (with size N_{chunk}) and the information stored in the specs file is updated chunk by chunk. The specs file contains useful information about the ROOT files, e.g. the maximum dimension of each

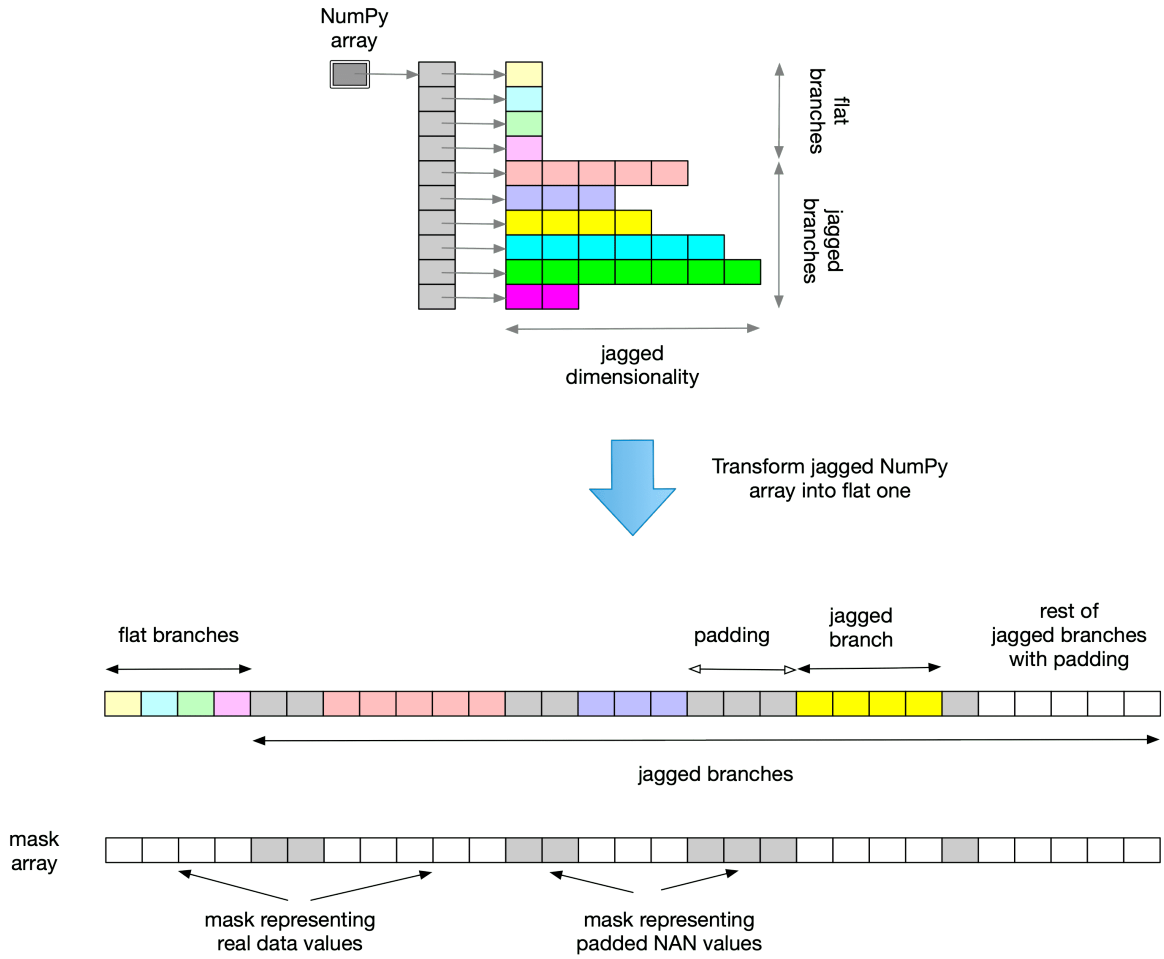


Fig. 4.1: Vector representation of an HEP event, consisting of flat attributes and Jagged Array attributes with padded values. The masking vector that stores the position of the padded values is also saved [140].

Jagged branch, the maximum and minimum value for each branch, and the number of events in each ROOT file[1].

The second part of the flowchart (indicated by ②) represents the ML training phase. At the beginning of the cycle, when the events are not yet read, N_{chunk} events from the i-th file f_i are read and stored into the i-th chunk c_i. Then, $N_{chunk} \cdot n_i / N_{tot}$ events are taken from it, where n_i is the number of events from the file f_i and N_{tot} is the total amount of events from all files. Subsequently, these events are converted into NumPy arrays, with the necessary transformation of the Jagged Arrays dimensions and normalization of the values (based on the information computed during step ① and stored in the specs file). The reading of the events and their pre-processing is carried out for all the files f_i. After creating a chunk of N_{chunk} events properly mixed from the different files, the events are used to train the ML model. The training phase is performed using batches of data of size N_{batch} taken from the created chunk and run for N_{epochs} epochs. In case N_{chunk} is not multiple of N_{batch}, the last batch used to train the ML model contains less than N_{batch} events.

1 Once the specs file is produced, either via the aforementioned procedure or by studying Monte-Carlo distributions to determine attribute dimensions and their min/max values, it can be reused for all files from the given dataset during the ML training phase.

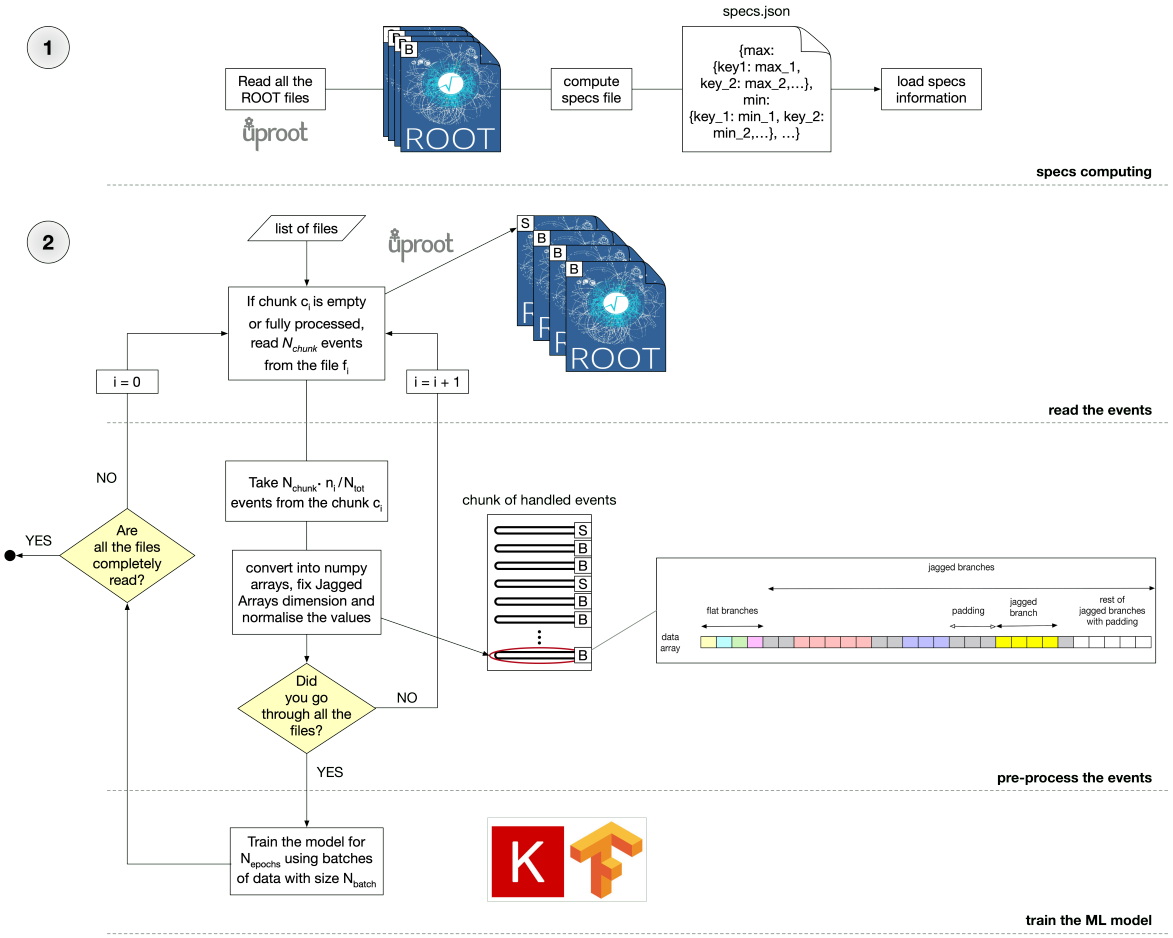


Fig. 4.2: Schematic representation of the steps performed in the MLaaS4HEP workflow (see text for details) [140].

Then the cycle starts again from the beginning of point ②, and if all the events stored in the chunk c_i have already been read, N_{chunk} events are read from the file f_i, otherwise, the proper amount of events ($N_{chunk} \cdot n_i / N_{tot}$) is read from the chunk c_i. The training process continues until all events are read, creating at each cycle a new chunk of events that is used to train the ML model for N_{epochs} epochs. At the end of the cycle, i.e. when all the events from all files are read and the ML training for all the individual epochs is completed, the final trained ML model is ready to be used in physics analysis.

In Appendix A, details on how to run a MLaaS4HEP workflow are reported.

4.1.2 *TFaaS architecture*

An inference layer can be implemented in various ways. It can be either tightly integrated with application frameworks (e.g. both CMS and ATLAS experiments followed this approach in their CMSSW-DNN and LTNN solutions respectively) or it can be developed as a Service (aaS) solution. The former has the advantage of reducing the latency of the inference phase per processing event, but the latter can be easily generalized and become independent from the internal infrastructure. For example, it can be easily integrated into cloud platforms, can be used as a repository of pre-trained models, and even serve mod-

els beyond the boundaries of experiments. However, the speed of the data inference layer (i.e. the throughput of serving predictions) can vary based on the chosen technology. The choice of the HTTP protocol ensures easy adaptation, whereas gRPC protocol can provide the best performance but requires dedicated clients. The inference layer, which complements with MLaaS4HEP, has been implemented as a TensorFlow as a Service (TFaaS) [174] based on the HTTP protocol.

Several ML frameworks have been evaluated and finally, the TensorFlow graphs have been chosen for the inference phase. The TensorFlow model represents a computational graph in a static form, i.e. the mathematical computations, the graph edges, and the data flow are well-defined at run time. Reading TensorFlow models can be done in different programming languages thanks to the support of the APIs provided by the TensorFlow library. Moreover, the TensorFlow graphs are very well optimized for GPUs and TPUs. TFaaS has been implemented using the Go programming language for the following reasons: the Go language natively supports concurrency via *goroutines* and *channels*; it is the language developed and used by Google, and it is very well integrated with the TensorFlow library; it provides a final static executable that greatly simplifies its deployment on-premises and to various (cloud) service providers. Clients can upload their TensorFlow models to the server and use them for their inference needs via the same interface. Both Python and C++ clients have been developed on top of REST APIs (end-points), and other clients can be easily developed thanks to the HTTP protocol. The TFaaS framework can be used outside of HEP to serve any type of TensorFlow-based models uploaded to the TFaaS service via the HTTP protocol (even, for instance, image recognition ML models).

One TFaaS server hosted by CERN and one hosted by INFN are online ([176] and [177] respectively). Moreover, TFaaS has been evaluated by CMS to be integrated in CMSSW as an official ML inference tool.

4.2 VALIDATION AND PERFORMANCE TESTING

To validate the results obtained with the MLaaS4HEP framework from the physics point of view and to test its performance, a real physics use case must be chosen. Due to affinity with the CMS analysis group, the $t\bar{t}$ Higgs analysis ($t\bar{t}H(b\bar{b})$) in the boosted, all-hadronic final state has been chosen (see Sec. 3.1.2.2 for the details of the analysis).

4.2.1 *MLaaS4HEP validation*

The goal of the validation is to demonstrate that the MLaaS4HEP framework can provide a valuable alternative and deliver comparable results with respect to the traditional analysis (that followed a standard BDT-based procedure with TMVA) using a pre-defined set of metrics.

Therefore, a dataset of 9 ROOT files from the resolved-Higgs analysis has been used, where 8 ROOT files contained background events and 1 file contained signal events. All the events contained in these ROOT files passed some selection criteria, briefly described in

Sec. 3.1.2.2. Each file had 27 flat branches, with ≈ 350k events in total, and the total size of the dataset was ≈ 28 MB. The ratio between the number of signal events and background events was ≈ 10.8%.

Firstly, a generic ML algorithm has been used to compare the results obtained inside and outside MLaaS4HEP. In particular, the following approaches have been explored:

- use MLaaS4HEP to read and normalize events, and to train the ML model;
- use MLaaS4HEP to read and normalize events, and a Jupyter notebook to perform the training of the ML model outside MLaaS4HEP;
- use a Jupyter notebook to perform the entire pipeline without using MLaaS4HEP.

The dataset has been split into three parts: 64% for training, 16% for validation, and 20% for test purposes, respectively. A Keras MLP has been used, with two hidden layers made by 128 and 64 neurons, and with a 0.5 dropout regularization between layers (see Listing 5 for the definition of the algorithm). Finally, the model is trained for 5 epochs with a batch size of 100 events, and the chunk size was set equal to the total number of events.

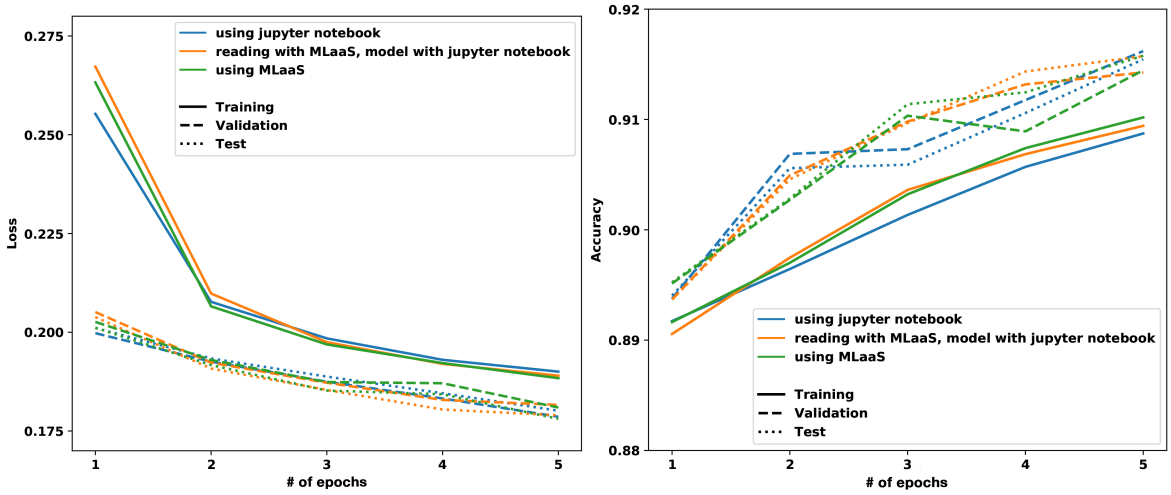
The results of this first validation are shown in Fig. 4.3, and show little or no difference among different approaches. Therefore, this means that a user can decide to read and normalize the events and train the ML model with or without MLaaS4HEP, with the same result. Nevertheless, MLaaS4HEP offers an easy approach that perform the whole pipeline without the user having to implement the single steps.

Subsequently, the MLaaS4HEP framework was used to train the same ML algorithm with the same architecture but now with the chunk size fixed to 10k events. The Fig.s 4.4a, 4.4b, and 4.4c, show the computed loss, accuracy and AUC metrics, respectively.

It can be observed that as the accuracy and AUC increase, the loss decreases with the number of chunks used for fitting the model, indicating that the ML model is actually learning. However, it can be observed that these trends are not smooth, in particular they show sawtooth shape patterns. These behaviors have been investigated by dropping one by one the ROOT files from the dataset, and it was found that a particular ROOT file called *ttH_noDRmatch* and containing background events is responsible alone for this effect. In Fig.s 4.4d, 4.4e, 4.4f, it can be seen that when all files except the one mentioned above are used the loss metric goes rapidly to 0, while the accuracy and the AUC go up to 1, respectively. When only the *ttH_noDRmatch* file is used as a background file the performance is lower: for example, the AUC score (see Fig. 4.4f) is between 0.7 and 0.8 during the training compared to 0.9 and 1 in the former case.

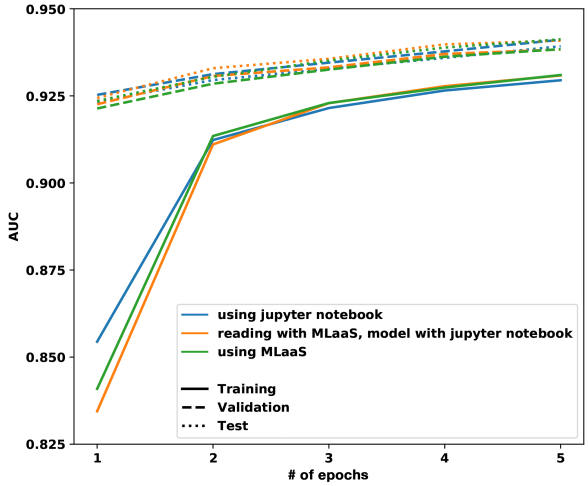
Furthermore, in order to see eventual effects caused by the unbalancing of classes in the data, an additional test has been performed by training the model with chunks made of 50% signal events and 50% of background events. This test confirmed the results obtained before, i.e. the *ttH_noDRmatch* file causes the spikes in the trend of the metrics in Fig.s 4.4a, 4.4b, 4.4c. When this background file is not used, the ML model almost perfectly distinguish signal from the background.

The effect of the *ttH_noDRmatch* file on overall performance is due to the fact that it has a similar signature to signal events, with the only difference that signal events correspond



(a) Loss metric.

(b) Accuracy metric.



(c) AUC metric.

Fig. 4.3: Comparison of the metrics (loss, accuracy, and AUC) scores for the training, validation, and test sets for three different cases: (i) using MLaaS4HEP to read and normalize events, and to train the ML model; (ii) using MLaaS4HEP to read and normalize events, and a Jupyter notebook to perform the training of the ML model outside MLaaS4HEP; (iii) using a Jupyter notebook to perform the entire pipeline without using MLaaS4HEP [140].

with the Higgs boson. Such attribute-level similarities influence the training process and are responsible for the spikes observed in ML evaluation metrics.

The final goal of this validation was not to reproduce and/or match the exact AUC number obtained in the official CMS physics analysis. It was found that the result (in terms of AUC score) obtained using the MLaaS4HEP approach is comparable with the result of the BDT implemented in TMVA and used in the official CMS physics analysis.

4.2.2 *MLaaS4HEP performance*

To test the MLaaS4HEP performance, all available ROOT files without any physics cuts have been used: the corresponding dataset had $\approx$ 28.5M events with 74 branches (22 flat and 52 Jagged), and a total size of $\approx$ 10.1 GB.

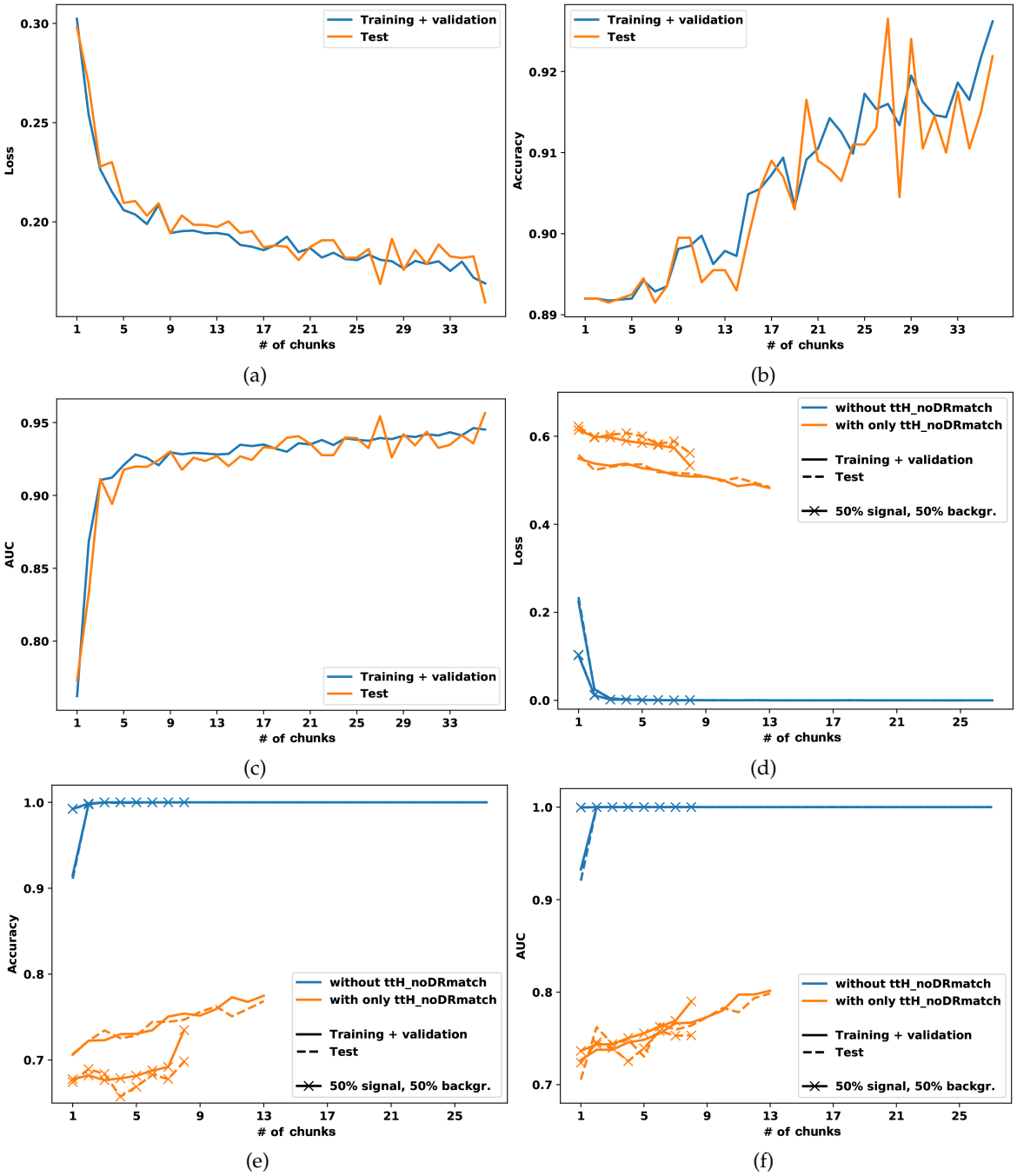


Fig. 4.4: Comparison of the metrics (loss, accuracy, and AUC) scores for the training plus validation, and test set using all the events of the dataset, read in chunk of size 10k (see plots (a), (b) and (c)). The plots (d), (e) and (f) show the comparison of the same metrics for three tests: one without the *ttH_noDRmatch* ROOT file in the background files list, one with only the *ttH_noDRmatch* ROOT file as background file, and finally, the third test (which repeats the first two) with a symmetric composition (50% and 50%) of signal and background in each data chunk (line with the $\times$ marker) [140].

All tests were performed using a macOS (laptop), 2.2 GHz Intel Core i7 dual-core, 8 GB of RAM, and a CentOS 7 Linux, 4 VCPU Intel Core Processor Haswell 2.4 GHz, 7.3 GB of RAM CERN VM. The ROOT files have been read from three data centers: Bologna (BO), Pisa (PI), and Bari (BA). The average available bandwidth was $\approx$ 129 with the Standard Deviation of Mean (SDOM) parameter equal to 4 Mbit/s and 639 (SDOM = 39) Mbit/s

using macOS and CERN VM, respectively (in both cases the values have been obtained after 10 trials).

Table 4.1 summarizes the I/O numbers obtained in the first step of the MLaaS4HEP pipeline[2] (① in Fig. 4.2) using various setups and a chunk size of 100k events. It provides values for the time taken to read the files, the time taken to compute specs values, the total time taken to complete the step ①, and the event throughput for the reading and specs computing steps.

	Reading time (s)	Specs comp. time (s)	Time to complete step ① (s)
macOS with local files	1633 (9)	958 (2)	2599 (11)
macOS with remote files (BO)	2365 (49)	974 (10)	3353 (57)
VM with local files	1131 (3)	963 (2)	2102 (5)
VM with remote files (BO)	2455 (68)	959 (2)	3427 (67)
VM with remote files (BA)	2304 (88)	961 (2)	3279 (89)
VM with remote files (PI)	2129 (41)	1044 (78)	3186 (83)

	Mean event throughput for reading (evts/s)	Mean event throughput for specs comp. (evts/s)	Mean event throughput for reading + specs comp. (evts/s)
macOS with local files	17608 (105)	29704 (63)	11055 (49)
macOS with remote files (BO)	12155 (261)	29245 (293)	8585 (149)
VM with local files	25493 (75)	29568 (73)	13690 (34)
VM with remote files (BO)	11716 (308)	29691 (70)	8396 (158)
VM with remote files (BA)	12538 (487)	29647 (65)	8801 (241)
VM with remote files (PI)	13503 (252)	27779 (1693)	9047 (228)

Table 4.1: Performance of the reading and specs computing phase with chunk size set to 100k events, using the macOS system and the CERN VM. Each value reported in the cells of the table represents the arithmetic mean of 5 trials with the corresponding SDOM reported inside the round brackets. The mean event throughput is the mean of the values obtained chunk by chunk in the phases of step ① of in Fig 4.2. In cases of local storage, the files were stored in a SSD 500 GB in the macOS case and in a Virtual Disk 52 GB in the CERN VM case, respectively. Furthermore, BO, BA, and PI stand for various Italian storage facilities with different WAN configurations (see the text for more details) [140].

Fig. 4.5 shows the mean event throughput for reading data as a function of chunk size for different trials. In all cases, there are no significant peaks. Larger chunk sizes can lead to certain problems, as in the case of the CERN VMs, where a limitation of the underlying hardware can be reached, e.g. big memory footprint. Using local files gave lower reading times and higher event throughput, while in the case of remote files the results were mostly influenced by the available bandwidth (the link connectivity between the processing node and the sites that host the data).

Fig. 4.6 shows the mean event throughput for reading data and computing the specs as a function of chunk size for different trials. The trend is the same of Fig. 4.5 but the values of throughput are lower since the time to computing the specs is added. In particular, the mean event throughput for the specs computing was found in general in the range 29k-31k, independent of the chosen platform, data location, and chunk size.

2 MLaaS4HEP pipeline and MLaaS4HEP workflow are used interchangeably in this thesis.

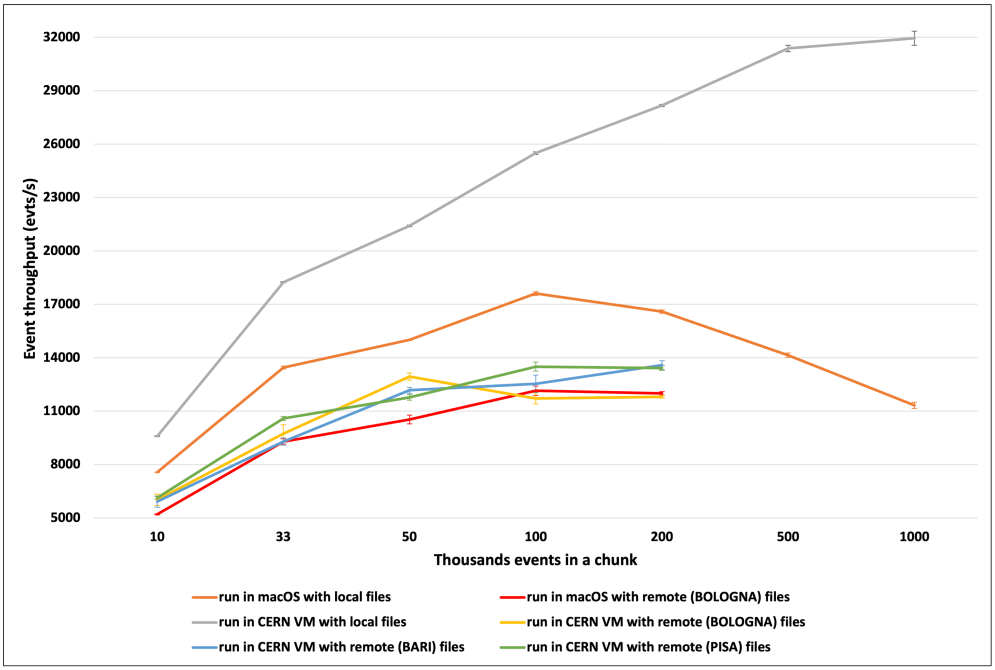


Fig. 4.5: Mean event throughput for reading data as a function of the chunk size for different trials of step ① in Fig. 4.2. The data points represent the arithmetic mean of 5 trials whereas the error bars represent the corresponding SDOM [140].

The performance studies of the second step of the MLaaS4HEP pipeline (② in Fig. 4.2) includes the data reading part, the data pre-processing step, and the time spent in the MLaaS4HEP training step.

As already mentioned in Section 4.1.1, there is a loop on the ROOT files that allows building the chunk used to train the ML model with the appropriate fraction of events. When the chunk containing the events of the i-th ROOT file is empty or fully processed, a new chunk of events is read from the i-th file, and the reading time is added to the total time spent on creating the chunk (see Fig. 4.2). In other words, the time spent on creating a chunk is the sum of n reading actions, and the time to pre-process the events. The event throughput for the data reading part has been already given in Table 4.1 and shown in Fig. 4.5. The event throughput for creating a single data chunk and the event throughput for pre-processing a single data chunk with a chunk size of 100k events are given in Table 4.2. In Fig. 4.7, the event throughput for creating a chunk as a function of the chunk size for different trials is shown.

The time spent for creating a chunk was almost the same for macOS and CERN VM, and similar for local or remote files. Obviously, for remote files, the reading time increased accordingly, and the time for creating the chunk increased, but this difference was pretty much negligible. For example, around 90 seconds were spent to create a chunk of 100k events, which corresponds to an event throughput of about 1.1k etvs/s, as given in Table 4.2. The choice of the chunk size is left to the user and there is no predefined best value for it. A good practice would be to start with a lower value of chunk size (e.g. 1k) and increase it gradually based on the resource availability.

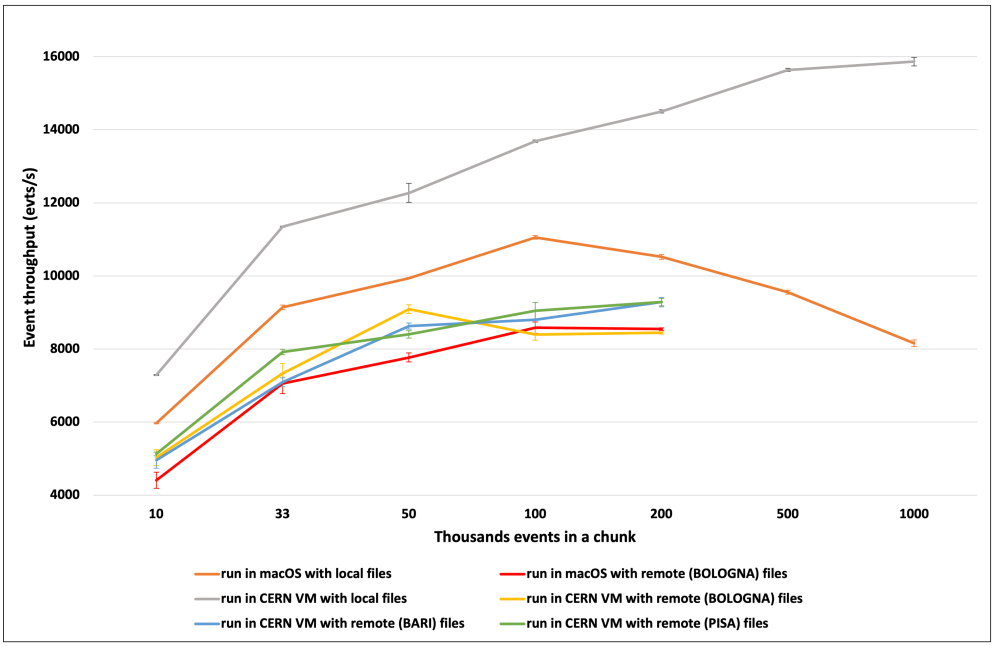


Fig. 4.6: Mean event throughput for reading data and computing the specs as a function of the chunk size for different trials of step ① in Fig. 4.2. The data points represent the arithmetic mean of 5 trials whereas the error bars represent the corresponding SDOM.

	event throughput for creating a chunk (evts/s)	event throughput for pre-processing a chunk (evts/s)
macOS with local files	1102 (11)	1157 (7)
macOS with remote files (BO)	1057 (17)	1138 (4)
VM with local files	1209 (11)	1247 (2)
VM with remote files (BO)	1110 (32)	1243 (5)
VM with remote files (BA)	1071 (19)	1153 (4)
VM with remote files (PI)	1152 (18)	1234 (5)

Table 4.2: Event throughput for the chunk creation and the pre-processing step with a chunk size of 100k events. The difference between the two steps is based on the reading part, i.e. the time for creating a chunk is the sum of the time of n reading actions from the ROOT files, and the time of the pre-processing step. The values in the table represent the arithmetic mean of the values obtained for 10 chunks and the corresponding SDOM is reported inside the round brackets [140].

The actual ML training time is independent of the MLaaS4HEP framework as it is determined by use of the underlying ML framework, e.g. Keras, the complexity of the ML algorithm used and the available hardware resources. In particular, using the simple ML algorithm introduced above and a chunk size of 100k events, for each chunk the time spent to split properly the data for training, validation and test purpose was about 1s (and almost equal for MacOS and CERN VM), and the training time for 5 epochs was about 11s and 13s for MacOS and CERN VM, respectively.

The heavy rewriting of the MLaaS4HEP code compared to the solution presented in [178] resolved few bottlenecks. For example, the reading time has been improved by a factor of 10. This resulted from better handling of Jagged Arrays by flattening the event arrays and computing of the min/max values of each branch. Additionally, the data pre-

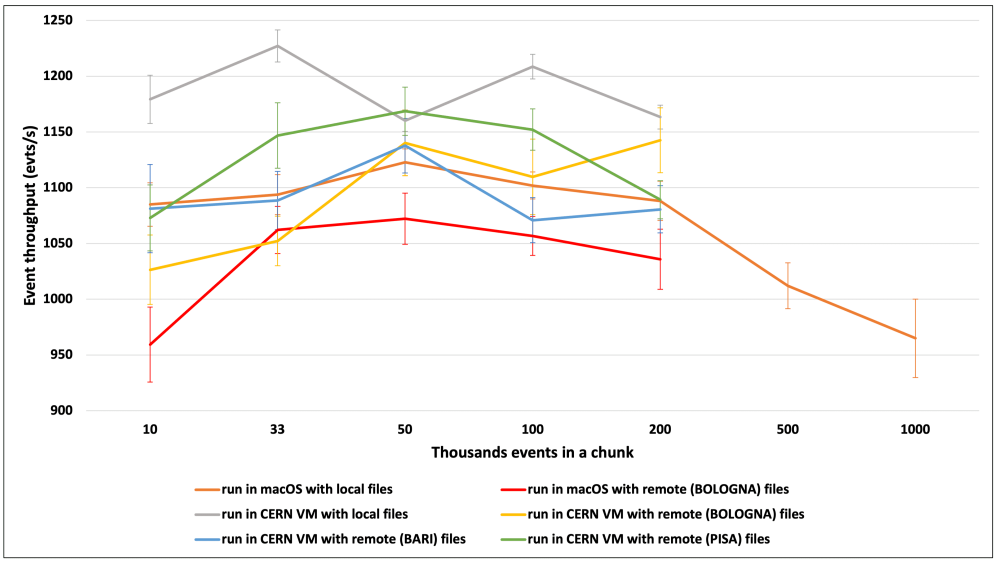


Fig. 4.7: Event throughput for creating a chunk as a function of the chunk size for different trials. The values represent the arithmetic mean of the values obtained for 10 chunks with the SDOM error bars [140].

processing step has been improved by a factor of 2.8 by using lists comprehensions instead of loops within the event.

On MacOS the MLaaS4HEP framework took about 86s to pre-process 100k events with 36%, 26%, 27%, and 6% breakdown used to extract and convert each event into a list of NumPy arrays, the normalization step, setting the dimensions, and creating the masking vectors, respectively.

In conclusion, using ≈ 10.1 GB of data (≈ 28.5M events) of the discussed physics analysis, the following results have been obtained:

- MLaaS4HEP framework is able to work with local and remote files;
- its throughput reaches ≈ 13.7k evts/s for reading the local ROOT files (with specs computing) and ≈ 9k evts/s for remote files;
- the throughput of the pre-processing step is peaked at ≈ 1.2k evts/s.

4.2.2.1 *MLaaS4HEP performance projection*

Based on the studies presented in Sec. 4.2.2 and considering the case of the CERN VM with local files, to process ≈ 10.1 GB of data (≈ 28.5M events) MLaaS4HEP took about 35 minutes for the first step of the pipeline (① in Fig. 4.2), i.e. to obtain min/max boundaries of all attributes across the processed events. The second step of the MLaaS4HEP pipeline (② in Fig. 4.2) took about 7 hours. This time includes reading all data chunks from the ROOT files, pre-processing the events, and feeding the data to the ML framework. Actual ML training time depends on the algorithm provided by the user and does not represent the MLaaS4HEP performance. In the studies reported in Sec. 4.2.2, it added an additional hour to the total time. Therefore, making an estimate using the same hardware resources,

the step ① will take $O(100)$ hours and $O(100k)$ hours for datasets at TB and PB scale, and the time for step ② will be $O(1k)$ hours and $O(1M)$ hours, respectively (to which must be added the time required to train the ML model).

These estimates suggest that further optimization of the MLaaS4HEP pipeline is required to process TB or PB scale datasets, and could involve I/O parallelization, distributed ML training, and other optimization techniques (more details in Sec. 4.6). At this stage, the goal was primarily to demonstrate the feasibility of the MLaaS4HEP pipeline, and to validate its use in the context of a real physics use case rather than performing a real ML training at the TB/PB scale.

4.3 FURTHER DEVELOPMENTS OF MLAAS4HEP

The development of the MLaaS4HEP framework started in 2018 and in these years the code has had many improvements. The most relevant package (dependency) MLaaS4HEP relies on is Uproot, which in turn relies on the Awkward Array library. There have been several releases of these two packages over the years (see Fig. B.1 for the timeline) and the breaking point for the MLaaS4HEP compatibility with Uproot has been the transition from Uproot version 3 (Uproot3) to version 4 (Uproot4) in the second half of 2020. This is mainly due to the transition from Awkward Array version 0 (awkward0) to version 1 (awkward1), which significantly changed the library interface, allowing it to handle a wider variety of nested and variable-sized data, and perform more complex manipulations on them.

In order to keep MLaaS4HEP updated with all its dependencies and since Uproot4 offered interesting features for the MLaaS4HEP development, it was decided to update the code to support Uproot4. Currently, the GitHub repository of the MLaaS4HEP framework [173] contains the Uproot3 and Uproot4 branches to distinguish two versions of the code corresponding to the two supported versions of Uproot. With the upgrade to Uproot4, new methods and functionalities have been introduced and the code has been modified to adopt the new syntax used by Uproot4 (see [108]). Let's take as an example the method used to distinguish if a branch is flat or Jagged. Uproot3 uses the `JaggedArray` type from awkward0 to understand the nature of the branch, whereas Uproot4 uses `AsJagged` from awkward1. In MLaaS4HEP, this results in the following transition:

```
isinstance(branch, awkward0.JaggedArray) #uproot3
isinstance(branch, uproot4.AsJagged) #uproot4
```

This is just an example and several functions have been subject to change. Thus, performance tests for the updated MLaaS4HEP code have been performed using a CentOS 7 Linux, 4 VCPU Intel Xeon Processor Skylake 2.1 GHz, 7.1 GB of RAM CERN VM, and considering the dataset used in the tests of Sec. 4.2.2 and stored locally.

Table 4.3 summarizes the I/O numbers obtained in the first step of the MLaaS4HEP pipeline (① in Fig. 4.2) using a chunk size of 50k events. Each value reported in the cells of the table represents the arithmetic mean of 5 trials with the corresponding SDOM reported inside the round brackets. The mean event throughput is the mean of the values obtained chunk by chunk in the phases of step ① in Fig. 4.2. Moreover, the values of

event throughput for creating a single data chunk in the second step of the MLaaS4HEP pipeline (② in Fig. 4.2) using a chunk size of 50k events are also reported in Table 4.3. In these cases, the values represent the arithmetic mean of the values obtained for 10 chunks and the corresponding SDOM is reported inside the round brackets. The values in the Table 4.3 show that in the reading phase there is a worse performance using Uproot4 than using Uproot3 but in the chunk creation phase, there is better performance with Uproot4.

	Uproot3	Uproot4
reading time (s)	1136 (2)	1301 (4)
specs comp. time (s)	653 (1)	607 (1)
time to complete step 1 (s)	1796 (3)	1914 (5)
mean event throughput for reading (evts/s)	25304 (44)	21995 (73)
mean event throughput for specs comp. (evts/s)	43604 (50)	46968 (50)
mean event throughput for reading + specs comp. (evts/s)	16012 (24)	14980 (38)
event throughput for creating a chunk (evts/s)	1197 (5)	1406 (14)

Table 4.3: Comparison of performance between Uproot3 and Uproot4 cases for the various phases of the MLaaS4HEP pipeline (shown in Fig. 4.2), with chunk size set to 50k events, using the CERN VM, and with the dataset stored locally. More details are in the text.

4.3.1 *New pre-processing operations*

The transition to Uproot4 allowed to introduce new pre-processing operations on events in MLaaS4HEP [179], i.e.:

- new branches definition;
- application of cuts on branches, both new and existing ones.

As reported in Appendix A, the MLaaS4HEP workflow is run using the `workflow.py` script which takes several arguments. An additional one can be used to introduce the new pre-processing operations, i.e.:

```
./workflow.py --files=files.txt --labels=labels.txt --model=model.py --params=params.json
--fout output_model --preproc=preproc.json
```

An example of the `preproc.json` file is shown in Listing 1.

```
{
  "new_branch": {
    "JetLep": {
      "def": "nJets + nLeptons",
      "type": "flat",
      "cut_1": "1<=JetLep<=5",
      "remove": "False"
    }
  },
  "flat_cut": {
```

```
    "nLeptons": {
      "cut": "nLeptons!=2",
      "remove": "False"
    }
  },
  "jagged_cut": {
    "jetPt": {
      "cut": ["jetPt>250","any"],
      "remove": "False"
      }
  }
}
```

Listing 1: Example of a `preproc.json` file provided by the user to apply pre-processing operations on data. See the text for more details.

The file is structured in three main categories: *new_branch*, *flat_cut*, and *jagged_cut*. The *new_branch* component allows the definition of new branches and gives the possibility to apply cuts on them, whereas the *flat_cut* and *jagged_cut* components allow the application of cuts on existing flat and Jagged branches respectively. If the user does not want to create new branches or apply cuts on flat or Jagged branches, he/she just should remove the corresponding category from the `preproc.json` file.

New branches can be defined through mathematical operations involving existing branches. To create new branches, the user must provide a series of information in the *new_branch* category:

- the name of the new branch;
- the mathematical definition of the new branch involving the existing branches, which supports both basic operations (+, -, *, /, **) and NumPy functions [180];
- the type of the new branch (flat or Jagged);
- the cut to apply;
- remove or not the new branch after the cut;
- the list of branches to be removed after the creation of the new branch.

In case of cuts on Jagged branches, the user must specify the type of cut choosing between *all* and *any*. The *all* type is used when the cut condition must be satisfied by all the values of a given Jagged branch, whereas the *any* type is used when at least one element in a given Jagged branch satisfies the cut condition. In case of multiple cuts, they must be sorted using *cut_i* as key, where the *i* indicates the number of the cut. If the user does not want to apply cuts on the new branch or does not want to remove any branches, then the *cut* and *keys_to_remove* keys must be removed from the `preproc.json` file respectively.

To apply cuts on flat and Jagged branches, the structure to be used is similar to the above. The following information must be provided:

- the name of the branch to cut on;

- the cut to be applied;
- remove or not the branch after cutting.

To apply more than one cut within the same branch, just number the cut key using *cut_i* as seen previously.

The proper functioning of the pre-processing operations introduced has been tested by comparing the results obtained by MLaaS4HEP and ROOT, taking the same dataset used in Sec. 4.2.2. The comparison on how to apply cuts on events between the Uproot and ROOT approaches is shown in the Appendix B.2.

The event throughput for the different phases of the MLaaS4HEP pipeline is reported in Table 4.4 considering different situations: without any cuts, with a cut on a flat branch, with a cut on a Jagged branch, with a cut on a new flat branch, and with cuts on flat, Jagged, and new flat branch (mixed cuts). The cuts are the same as reported in Listing 1. In these tests, 50k events have been read with a chunk size of 5k events.

	no cut	flat cut	Jagged cut	new branch cut	mixed cuts
mean event throughput for reading (evts/s)	15157 (71)	15325 (56)	22505 (64)	19718 (51)	19375 (20)
mean event throughput for specs comp. (evts/s)	44004 (52)	43600 (136)	947 (4)	878 (11)	944 (5)
mean event throughput for reading + specs comp. (evts/s)	11273 (37)	11339 (29)	908 (3)	841 (10)	900 (5)
event throughput for creating a chunk (evts/s)	1363 (3)	1395 (8)	125 (1)	124 (1)	124 (1)

Table 4.4: Event throughput for the different phases of the MLaaS4HEP pipeline (shown in Fig. 4.2) and for different cuts applied. The total number of events read was set to 50k and the chunk size was set to 5k events. The tests have been performed using the CERN VM and considering the dataset stored locally.

The values in Table 4.4 shows a strong performance degradation when the cut on the Jagged branch and on the new flat branch is applied (and consequently also in the case of mixed cuts). The reason for this behavior is essentially related to a conversion issue: by applying cuts on flat branches the same data type (i.e. Numpy ndarray) is kept for the whole MLaaS4HEP workflow. However, this is no longer true when defining new branches or applying cuts on Jagged branches because data is converted by Uproot to a type (i.e. AwkwardArray) which in turn must be converted, event by event, causing the slowdown. Thus, when the user needs to perform complex operations, e.g. creating new branches or cutting on Jagged branches, and needs good time performance, it is preferable to perform such operations outside MLaaS4HEP, especially when dealing with very large datasets. Instead, when simple operations (e.g. cuts on flat branches) are required, the performance is very similar to the case without cuts.

4.3.2 *New training procedure*

Considering the case of a NN, the current (named "original" in this subsection) MLaaS4HEP training procedure is performed chunk by chunk where each chunk is used

one at a time to train the model for N_{epochs} epochs, with N_{epochs} defined by the user. An additional training procedure (named "standard" in this subsection) has been introduced, in which for each epoch all the chunks are used. Both training approaches have pros and cons. The original MLaaS4HEP training approach is useful when the dataset is large and exceeds the amount of RAM of the training node, as only a chunk at a time is stored in the memory and the chunk size can be adjusted to fit in the hardware resources. However, the user should pay close attention to the ML model convergence and validate it after each chunk [181]. The standard MLaaS4HEP training approach by using the entire dataset for each epoch can guarantee the ML model convergence, but the dataset should fit into the RAM of the training node. In terms of training speed, the original approach is faster than the standard one (this effect is prominent when remote ROOT files are used), since after having used a chunk for the training, that chunk is no longer read and used later. Since MLaaS4HEP now comes with two training procedures, the user can try both and select the best one for him/her use case.

To make a proper comparison of the two training approaches, the AUC and loss scores are plotted in Fig. 4.8, using the ML algorithm in Listing 5 (but with an SGD optimizer), and using the dataset introduced in Sec. 4.2.1. The plots of Fig.s 4.8a and 4.8b are obtained using a chunk size of 100 events, whereas the plots of Fig.s 4.8c and 4.8d are obtained using a chunk size of 100k events. The plots show that the two MLaaS4HEP training approaches produce very similar results when the chunk size is set to 100k, whereas in the case of 100 events as chunk size the standard approach shows improvements in the performance with a smoother trend and better learning. This means that in the original approach "large" chunks are enough for the model to learn from data, but when the chunk size is decreased it results in a loosing of performance and less convergence. On the contrary, the model benefits from a training procedure that foresees seeing all chunks for each epoch. The trends obtained depend on the use case chosen, but generally when two classes are difficult to distinguish it is always preferred to use small chunks to ensure correct learning of the model.

Another factor to take care of is the time to perform all the MLaaS4HEP training procedure, and a trade-off between the time spent and the performance should be considered. The time spent by MLaaS4HEP for the training procedure as a function of the number of epochs is shown in Fig.s 4.9a and 4.9b (considering the same use case as before). In Fig. 4.9a, the values are obtained by fixing the chunk size to 100 events, while in Fig. 4.9b, the chunk size was set to 100k. In both cases, the original MLaaS4HEP approach takes much less time than the standard one since each chunk is read only one time and then discarded. Moreover, with 100 events as chunk size the time is an order of magnitude greater than with 100k events as chunk size for both training approaches. Thus the user should take into consideration all these elements before choosing which approach to adopt.

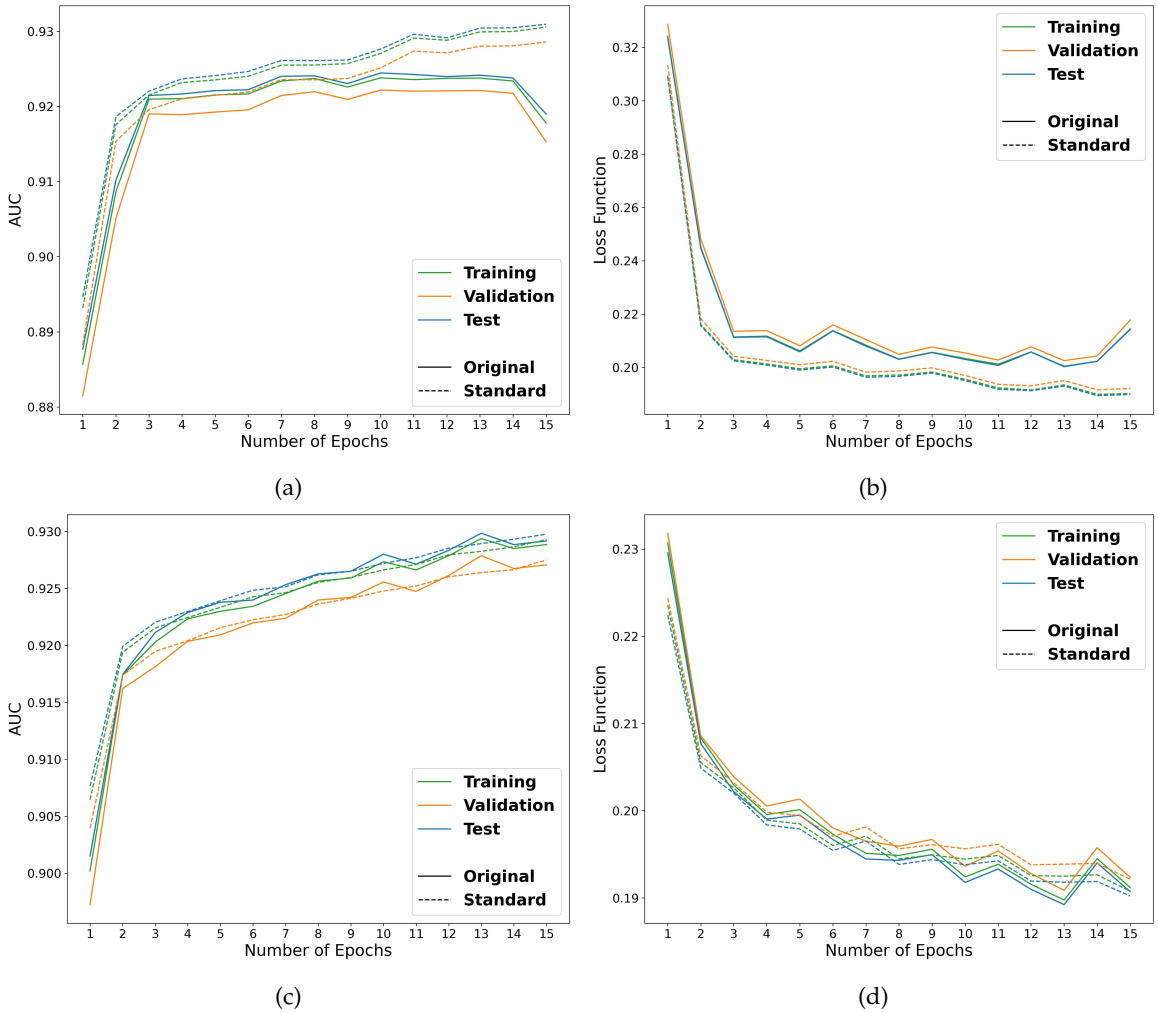


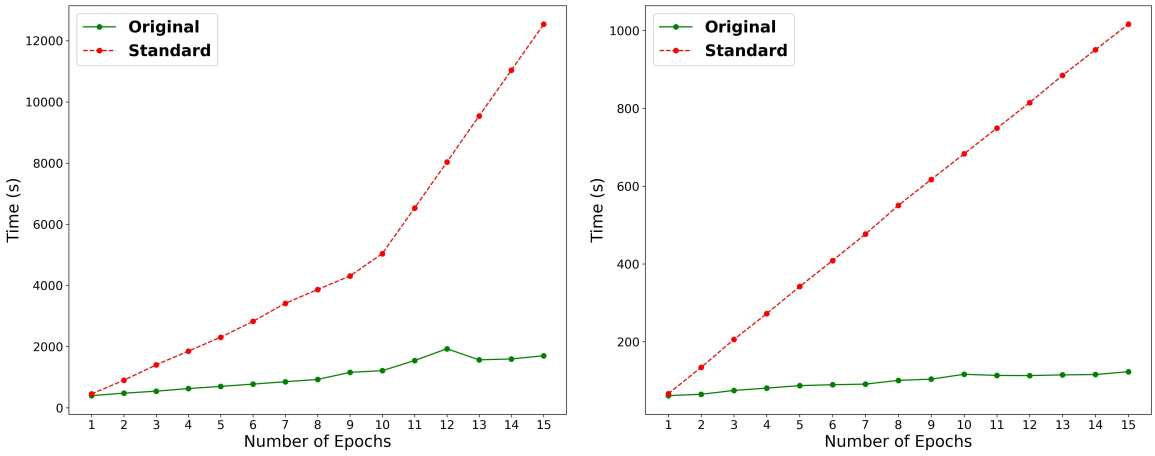
Fig. 4.8: AUC and loss metrics comparison between original and standard MLaaS4HEP training approaches using the model in Listing 5 and an SGD optimizer. Plots (a) and (b) were obtained by setting the chunk size to 100 events, while plots (c) and (d) setting the chunk size to 100k events [108].

4.3.3 *Generalization to other ML frameworks*

As already mentioned, MLaaS4HEP is abstracted to support any kind of Python-based ML framework and algorithm (with a little effort in adjusting the code). So far, in the tests reported, only Keras MLPs have been used, and to enable the use of other frameworks and algorithms the MLaaS4HEP framework needed additional work. Thanks to the new implementations, MLaaS4HEP has been successfully tested using:

- MLP written in PyTorch;
- MLP, Gradient Boosting, AdaBoost, Random Forest, Decision Tree, kNN, SVM, and Logistic Regression written in Scikit-learn;
- Gradient Boosting written in XGBoost.

See Appendix B.4 for more details on the definition of ML algorithms to be used in MLaaS4HEP and for a PyTorch example.



(a) (b)

Fig. 4.9: Comparison between original and standard training approaches in terms of training time. In (a), the chunk size was set to 100 events, and in (b), the chunk size was set to 100k events [108].

4.3.4 *Providing metrics score*

Since metrics are useful to quantify the goodness of trained ML models, it was considered useful to add the possibility for the user to have MLaaS4HEP print some of the most common metrics, i.e. AUC, Confusion Matrix, Precision, Recall, and F1.

4.3.5 *MLaaS4HEP application on the Higgs Boson ML challenge*

MLaaS4HEP has been successfully used to tackle the Higgs Boson ML challenge (see Sec. 3.1.2.1 for all the details of the analysis). In [108], the strategy adopted for the challenge using MLaaS4HEP is described and it has been reproduced in a Jupyter notebook that can be found in [182]. The notebook is designed to be run on Google Colab but can easily be adapted to run on any platform. This application shows that MLaaS4HEP is also HEP experiment agnostic: in fact, even if it is a CMS project, it can also be used with data of other experiments adopting ROOT, like ATLAS. Moreover, the data of the challenge is open data, meaning accessible also from outside the experiment and CERN itself. This adds value to the MLaaS4HEP framework, as it can be used as a tool to show non-HEP experts (e.g. during outreach events) how ML techniques can be applied to HEP problems.

4.4 TOWARDS A MLAAS SOLUTION FOR MLAAS4HEP

The MLaaS4HEP performance reported in Sec. 4.2.2 is highly dependent on the available hardware resources, and while it might be acceptable for datasets with a size of $\approx$ 10 GB, it should be improved for larger datasets (e.g. at TB or PB scale) looking to the upcoming HL-LHC. This improvement can be achieved by exploring different ways, such as adopting new solutions in the code, investing in better and more expensive on-premise resources,

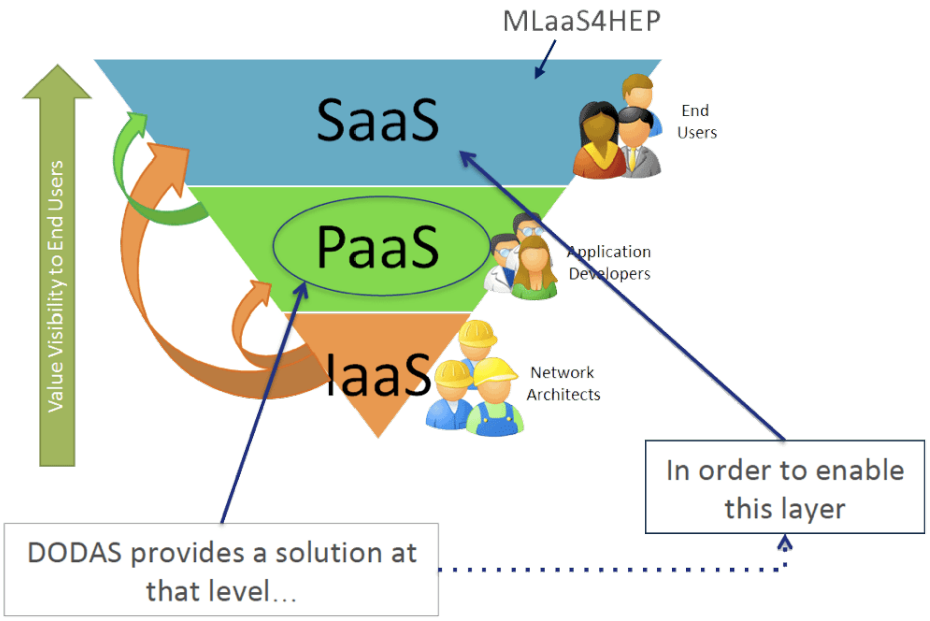


Fig. 4.10: Schematic representation of the levels of a service. DODAS provides a solution at the PaaS level, and integrating the MLaaS4HEP framework the results is towards a SaaS solution [183].

or moving to the cloud. The "cloudification" operation has two benefits: it allows to use of heterogeneous and potentially more performing resources but it also opens the doors to the creation of an "as a Service" solution. At this stage, MLaaS4HEP is only a framework that can be installed and used where the user prefers, but it is not a service (with APIs) and it is not an "as a Service" solution. This section shows the work done [183] towards a SaaS[3] solution with MLaaS4HEP using the DODAS service [184]. In fact, by integrating the MLaaS4HEP framework with the platform (i.e. a VM) provided by DODAS, the final product is towards a SaaS (in particular, a MLaaS in this case) that allows the user to run the MLaaS4HEP workflow without the need to install it and its dependencies (see Fig. 4.10). In the end, what will be missing for a real SaaS is to create a MLaaS4HEP service and integrate it with DODAS which will manage the deployment of the final solution using cloud resources.

The choice to use DODAS as PaaS was mostly driven by the proximity to the DODAS working group and the ease of using its resources. At the same this work provides compliance with the INFN-Cloud [185] portfolio of services, which opens up the possibility for INFN-Cloud users to use the MLaaS4HEP framework "as a Service".

4.4.1 *DODAS*

Dynamic On Demand Analysis Service (DODAS) is a PaaS tool for generating container-based solutions on cloud and on-demand resources. It is designed for scientists looking to easily leverage distributed and heterogeneous cloud resources, by automatizing the process of provisioning, creating, managing, and accessing resources. Its development began

3 See Sec. 2.2.2 for the datails on the cloud service models.

in the context of the INDIGO-DataCloud [186] project and its architecture has been realized mostly using INDIGO building blocks. It implements a services composition model based on templates, where services can be chosen depending on the use case, providing in this way a customizable platform. The DODAS architectural pillars are the following.

- The **PaaS Orchestrator** is the DODAS entry point where user's requests for the application or service deployments, typically expressed in a TOSCA [4] template, are taken in charge. Its role is to provide the best infrastructure for the deployment considering the requests of the users, the availability, and the state of health of the IaaS services. The interaction with the infrastructure is delegated to the Infrastructure Manager.

- The **Infrastructure Manager** (IM) is responsible to deploy customized virtual infrastructures on different IaaS Cloud deployments, allowing support for various cloud providers (e.g OpenStack, Amazon AWS, and Microsoft Azure). It facilitates the access and usability of IaaS clouds by automating the Virtual Machine Image (VMI) selection, deployment, configuration, software installation, monitoring, and update of the virtual infrastructure.

- The **Identity and Access Management** service (IAM) manages identities, enrollment, group membership, attributes, and policies for accessing distributed services and resources. It supports federated authentication mechanisms behind the INDIGO Authentication and Authorization Infrastructure (AAI). Users can authenticate with any of the supported mechanisms (SAML, OIDC, X.509 certificates, and local username and password), also using the credentials provided by existing Identity Federations (e.g. IDEM and eduGAIN).

Resource abstraction and full automation are implemented by combining PaaS Orchestrator and IM together. In this way, end users are able to leverage computational resources without knowing the IaaS details. Cluster and services configuration is automated using Ansible recipes. The combination of TOSCA and Ansible allows an easy procedure to describe complex computing infrastructures.

4.4.2 *Using DODAS for MLaaS4HEP claudification*

For the MLaaS4HEP cloudification with DODAS, the following steps have been performed (for more details see Appendix C and [187]):

1. create a Docker image able to run the MLaaS4HEP pipeline;

2. create an Ansible role to automatize the configuration and deployment of the container with dependencies (e.g. download Docker, pull the Docker image, and run the container);

4 Topology and Orchestration Specification for Cloud Applications (TOSCA) is an OASIS open standard used to define the topology of services provisioned in IT infrastructures.

3. create a Tosca template to define the resource requirements (e.g. the number of CPUs and memory size of the VM to be delivered to the user) and the input parameters for the creation of the docker container (e.g. define a folder of the host file system to be connected with the container);

4. create the deployment from the command line.

Therefore, DODAS has been used to automatize the overall deployment via the command line interface and to equip the user with a platform (leveraging cloud resources available to the DODAS project) where run the MLaaS4HEP pipeline. In addition to SSH access to the VM, a JupyterHub interface is provided which can be very useful in research groups as it provides a user-friendly solution with a shareable Jupyter notebook, i.e. the user can work in a shared environment with colleagues using the same cloud interface. It integrates cloud storage for the management of the required files (e.g. ROOT files, and ML algorithms definition) that can be shared with other users. It implements a token-based access using the IAM service and it has the support for a customizable environment, i.e. the user can choose the Docker image he/she wants (see Fig. C.2). Using this interface the user can simply open a terminal and run the MLaaS4HEP pipeline.

Functional tests of this solution have been performed by deploying an Ubuntu 18 Linux, 8 AMD Opteron 62xx class CPU 2.6 GHz, 16 GB RAM VM with DODAS and running the MLaaS4HEP pipeline on it. The average available bandwidth was $\approx$ 576 (16) Mbit/s obtained in 6 trials. Table 4.5 summarizes the I/O numbers obtained in the first step of the MLaaS4HEP pipeline (① in Fig. 4.2) using a chunk size of 100k events. Each value reported in the cells of the table represents the arithmetic mean of 5 trials with the corresponding SDOM reported inside the round brackets. The mean event throughput is the mean of the values obtained chunk by chunk in the phases of step ① in Fig. 4.2. Moreover, the values of event throughput for creating a single data chunk and the event throughput for pre-processing a single data chunk in the second step of the MLaaS4HEP pipeline (② in Fig. 4.2) using a chunk size of 100k events are also reported in Table 4.5. In these cases, the values represent the arithmetic mean of the values obtained for 10 chunks and the corresponding SDOM is reported inside the round brackets. The dataset used is the same as the tests of Sec. 4.2.2 and two cases have been considered: the dataset stored locally and in the Pisa data center.

Regarding the ML training time, considering the simple ML algorithm used in Sec. 4.2.1 (see Listing 5) and a chunk size of 100k events, for each chunk the time spent to split properly the data for training, validation, and test purpose was about 1.5 s, and the training time for 5 epochs was about 27 s.

A comparison with the results reported in Tables 4.1 and 4.2 shows that the values obtained with the VM deployed with DODAS are in general worse than the macOS and CERN VM cases. The reason is that in the former case, the VM was shared with other users and during the tests an exclusive usage of the resources was not possible. Anyway, this was just a functional test to show the feasibility of the whole procedure, and not to get the highest possible performance values.

For more details on the work of cloudification with DODAS see Appendix C.

	local files	remote files (PI)
reading time (s)	1687 (1)	2538 (46)
specs comp. time (s)	2245 (38)	2376 (50)
time to complete step 1 (s)	3953 (35)	4937 (50)
mean event throughput for reading (evts/s)	17111 (6)	11355 (209)
mean event throughput for specs comp. (evts/s)	12697 (211)	12000 (251)
mean event throughput for reading + specs comp. (evts/s)	7286 (69)	5828 (57)
event throughput for creating a chunk (evts/s)	494 (2)	498 (4)
event throughput for pre-processing a chunk (evts/s)	505 (1)	512 (1)

Table 4.5: Performance of the various phases of the MLaaS4HEP pipeline (shown in Fig. 4.2) with chunk size set to 100k events, using the VM deployed with DODAS, and considering data stored locally and in the Pisa data center. More details are in the text.

4.5 CLOUD NATIVE APPROACH FOR MLAAS4HEP

The work described in Sec. 4.4.1 shows a general procedure to create an automated deployment of a service applied in the specific case of the MLaaS4HEP framework. However, MLaaS4HEP is not yet a service, meaning that it lacks the APIs through which a user can interact with it. To be integrated into a cloud and to be designed as a cloud native application, the final product should be developed in terms of interconnected microservices each of them in charge of different tasks. The following microservices have been identified as the pillars of such a service that should be implemented [179]:

- a MLaaS4HEP server, which allows to submit MLaaS4HEP workflow requests and manage all the actions related to it;
- an authentication/authorization layer, which allows to authenticate the users and authorize their requests to the MLaaS4HEP server;
- an XRootD Proxy server, which allows using X.509 proxies for the remote access of data.

In Sec. 2.7.1, detailed information about cloud native applications and microservices has already been provided. A cloud native application is based on containerization, i.e. every part of such an application is housed within its own container. This allows them to work in different cloud environments, and to be moved more easily between different cloud environments and between on-premises and the cloud, even because they carry with them all of their dependencies. Therefore, the services mentioned above must be developed as containers.

The MLaaS4HEP service has been developed using the Flask framework (which allows developing easily web applications in Python), building APIs that allow the user to manage a MLaaS4HEP workflow by making HTTP calls via curl, i.e.:

- upload a folder with all the files needed to run a MLaaS4HEP workflow (e.g. the input ROOT files and the definition of the ML algorithms);

- delete the uploaded folder;
- submit a MLaaS4HEP workflow, which in turn is containerized and runs in background;
- monitor the status of the "job" (container);
- download the logs of the container;
- download the trained ML model.

The code of the MLaaS4HEP server is available in [188]. To run a MLaaS4HEP server locally and run a MLaaS4HEP workflow, the following lines should be executed:

```
FLASK_ENV=development FLASK_APP=server.py flask run -p 8080
curl -H "Content-Type: application/json" -d @submit.json http://localhost:8080/submit
```

In this case, the MLaaS4HEP server (defined in the `server.py` file) listens on port 8080. The submit API takes as argument a json file containing information about the resources that the container (which runs the MLaaS4HEP workflow) can use (i.e. the memory size and the number of CPUs), and the files to use for the workflow. A container for the MLaaS4HEP server has not yet been developed, but it is expected to be done as it is a requirement of a microservice.

The easiest and most efficient approach for the implementation of the authentication/authorization layer is to "do not reinvent the wheel" and use existing solutions that rely on industry standard, i.e. OAuth 2.0 and OIDC protocols. Therefore two reverse proxies have been tested and successfully integrated: auth-proxy-server [189] (which is an R&D product of CMS mainly focused on the needs of CMS) and OAuth2-Proxy [190]. To be as much generic as possible, the second option has been chosen for the continuation of the work. The steps performed to properly integrate and use the OAuth2-Proxy with the MLaaS4HEP server are the following.

- Register a client using oidc-agent [191], choosing *https://cms-auth.web.cern.ch/* as the authorization server. At this stage, the user is authenticated and gives authorization to the client to talk with the authorization server.
- Obtain an access token for the registered client, setting the audience to the *CLIENT_ID*.
- For TLS connections, get a certificate for the MLaaS4HEP server from a trusted, third-party Certificate Authority (for simplicity, a self-signed certificate has been used, obtained with the openssl package).
- Prepare a configuration file for the OAuth2-Proxy server setting the *CLIENT_ID* and *CLIENT_SECRET* of the registered client (a similar one was prepared for the case of TLS connections).
- Run the OAuth2-Proxy server using the pre-built Docker image.

- Make a curl call to a MLaaS4HEP server API (e.g. `/submit`) passing the token previously obtained and using the port where the proxy is running, such as the following:

```
curl -L -k -H "Authorization: Bearer ${TOKEN}" -H "Content-Type: application/json"
-d @submit.json http://localhost:4180/submit
```

In this case, the OAuth2-Proxy server is listening on port 4180, so that when the user submits the request is redirected to the authentication service that authorizes or not the request. A proper authorization procedure has not been implemented at this stage, meaning that the token claims are not inspected and all the requests for the resource server (i.e. the MLaaS4HEP server) with a valid access token are accepted by the authorization server. For more details on the authentication/authorization via tokens, see Secs. 2.8.4 and 2.8.5.

The last missing piece is an XRootD Proxy server that allows access to remote ROOT files located on WLCG sites. For this task the existing compose-xrootd solution [192] has been chosen, a product of the Italian CMS Computing R&D group. It consists of a Docker compose of two services, xrootd proxy-cache server and X.509 proxy renewer. The former behaves like a cache where X.509 proxies are stored and accessed by external services. The latter checks at regular intervals (e.g. every 30 s) whether the X.509 proxy has expired or not. If it is, the proxy is renewed.

After developing and configuring all the pieces of the overall service, they have been connected to each other and deployed on a VM of the INFN-Cloud. To complete this picture, a TFaaS server has also been deployed in the VM so that once a ML model has been trained by MLaaS4HEP it is available for TFaaS to make inferences on the events given by the user. A schematic representation of the current configuration is shown in Fig. 4.11.

A working prototype of such a solution is currently running on the INFN-Cloud: the MLaaS4HEP server APIs can be reached at the following address `https://90.147.174.27:4433`, while the TFaaS ones at `https://90.147.174.27:8081` [193]. Once the user obtains an access token from the authorization server, he/she can contact the MLaaS4HEP server or TFaaS using curl, e.g. in the following ways:

```
curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" -H "Content-Type:
application/json" -d @submit.json https://90.147.174.27:4433/submit

curl -L -k -H "Authorization: Bearer ${TOKEN_TFAAS}" -X POST -H "Content-type:
application/json" -d @predict_bkg.json https://90.147.174.27:8081/json
```

The former command allows training a ML model, whereas the latter allows using this model to get the prediction on a given event (stored in the `predict_bkg.json` file).

For more details on the work related to the cloud native solution of MLaaS4HEP see Appendix D.

4.6 FINAL CONSIDERATIONS AND FUTURE PLANS

The MLaaS4HEP project is constantly evolving, both in improving the functionality of the framework and in creating a real SaaS that can be used by many HEP users.

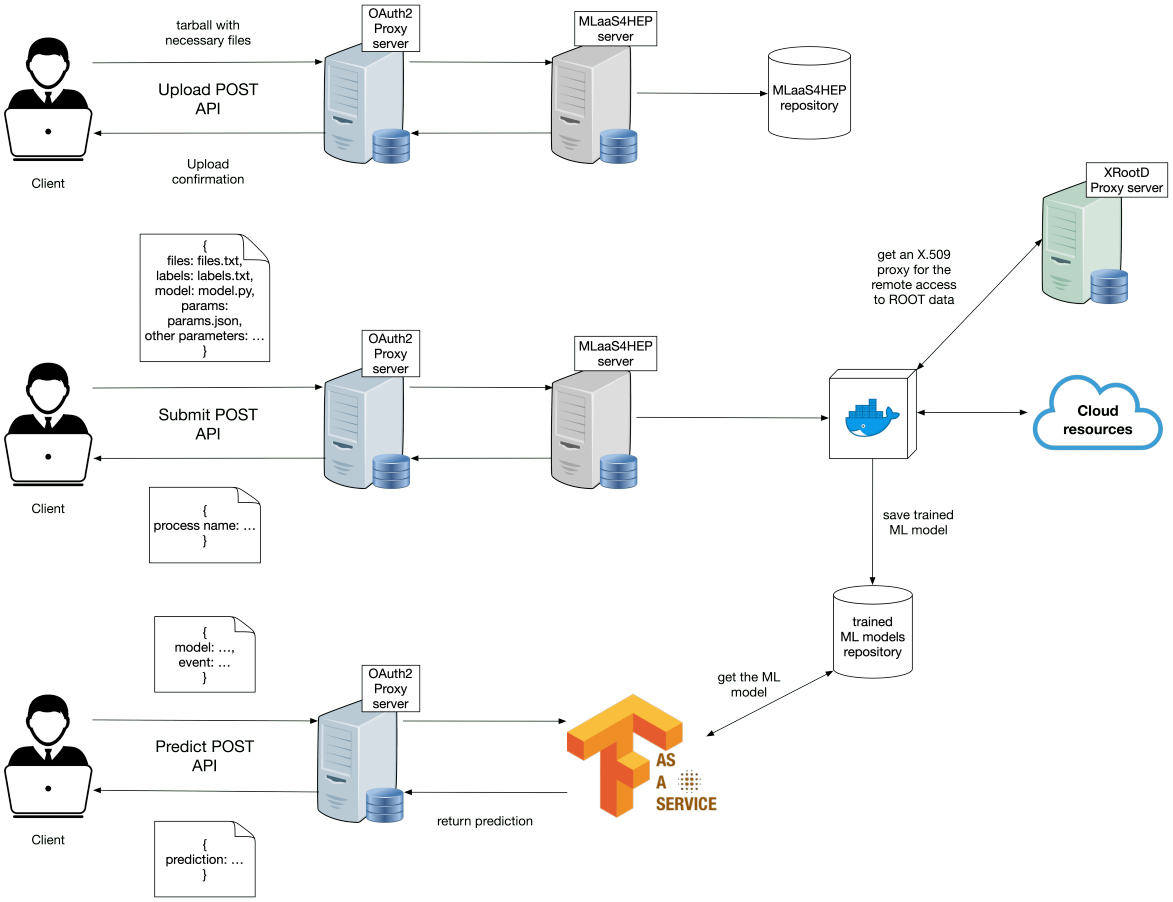


Fig. 4.11: Solution connecting OAuth2-Proxy server, MLaaS4HEP server, XRootD Proxy server, and TFaaS. Firstly, a client uploads a tarball with the necessary files to the MLaaS4HEP server, then submits a MLaaS4HEP workflow which consists in running a Docker container in the server, which can read remote ROOT files using valid X.509 proxies, and that trains and saves the ML model. This model is accessed by the TFaaS service that uses it to make inference. In front of the MLaaS4HEP server and TFaaS server, two OAuth2-Proxy servers are used to handle user authentication and request authorization [179].

Regarding the development of the MLaaS4HEP code, a relevant aspect has been the close contact with the people performing HEP analyses with whom can talk and discuss problems, and understand what they need from a ML service to be used in the analyses. Ideas such as the creation of new branches and the application of cuts on branches emerged from such discussions. Furthermore, the use of chat platforms such as [194] allowed discussions with the Uproot developers to better understand the issues raised and how to deal with them.

These forms of interaction have been a fundamental aspect for the evolution of MLaaS4HEP and will continue to offer insights to add new functionality to the code and improve it. The communicative aspect with people of the HEP world is crucial for the success of the project and for this reason the MLaaS4HEP framework has been presented in several international conferences and workshops, e.g. Large Hadron Collider Physics (LHCP) [172], International Conference on High Energy Physics (ICHEP) [193], International Symposium on Grids & Clouds (ISGC) [179, 183], IML workshop [195–197], CMS ML Town Hall workshop [198, 199] and workshop of data analysis at CMS Italia [200]. On

these occasions, talking with people and presenting the developments of the project allowed MLaaS4HEP to be known and to arouse interest. Currently, MLaaS4HEP is present in the official CMS ML documentation [181] and in the Review of Particle Physics by the Particle Data Group (PDG) [94].

Moreover, there are some working directions under investigation and development for the MLaaS4HEP project.

- It is planned to provide a general inference service to the entire MLaaS4HEP service described in Sec. 4.5, since TFaaS allows to use only Tensorflow/Keras algorithms. For example, Torchserve has been successfully used to make inference with PyTorch models [201]. A reverse proxy server can be deployed to work with TFaaS and Torchserve as backends. Other inference services supporting other frameworks can be integrated or developed to broaden the spectrum of the supported solutions.

- It is planned to deploy the entire service with DODAS, to provide a SaaS in a more appropriate way. In addition, a web application for the service with a proper graphic interface should be developed to allow a more user-friendly solution.

- Since the entire MLaaS4HEP service should support the submission of multiple requests by multiple users with a multi-tenant architecture, a resource scheduler should be integrated. Moreover, the MLaaS4HEP framework code should be in part redesigned to separate the part of reading with pre-processing of data and the part of training the ML model into two different processes. This would allow to better optimize the resource (even heterogeneous) for the single pieces of the workflow, for example parallelizing the phase of reading with pre-processing of data and exploiting distributed ML training. These two processes should be containerized and properly interconnected (see Fig. 4.12). Using Kubernetes would allow to schedule the resource requests of the submitted workflows and control the processes offering a scalable solution. In this direction, Kubeflow may provide a valid choice to follow [202].

- The MLaaS4HEP framework is developed primarily to accept flat ROOT TTrees as input for HEP classification problems. However, it is planned to enable MLaaS4HEP to address also regressions problems and image classifications. Furthermore, MLaaS4HEP is equipped to read not only ROOT files but also Json, CSV, Avro, and Parquet files both locally and from HDFS. Their use must be carefully tested and requires further development. Nevertheless, a positive test has been done to predict the activity of Rucio data placement reading Json files from HDFS with the MLaaS4HEP framework [203].

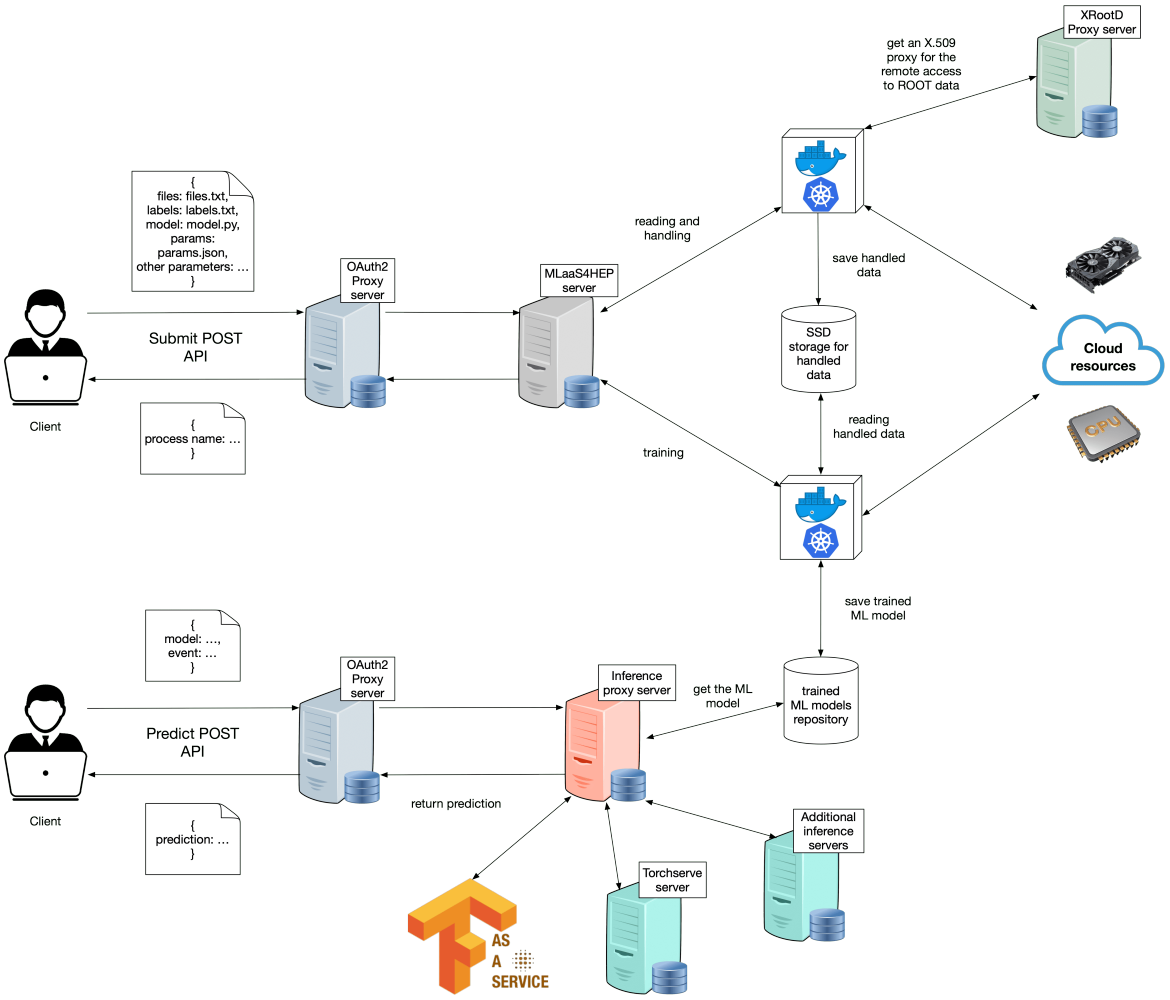


Fig. 4.12: Desirable solution connecting OAuth2-Proxy server, MLaaS4HEP server, XRootD Proxy server, and an inference server working with multiple backends (i.e. TFaaS, Torchserve, and others). The MLaaS4HEP workflow is split into two different containerized processes, one for reading with pre-processing of the events and another for the training of the ML model. The former writes handled data in an SSD storage for fast access of the latter. A solution involving Kubernetes would allow to schedule the workflows and control the processes offering a scalable solution.

CONCLUSIONS

This PhD thesis described the development of a MLaaS solution for HEP aiming to provide a cloud service that allows HEP users to run ML pipelines via HTTP calls, consisting of reading data, processing data, training ML models, and serving predictions directly using ROOT files of arbitrary size from local or distributed data sources.

The development of the MLaaS4HEP framework is at the basis of such a project and for an appropriate evaluation of its usability in the HEP world the $t\bar{t}$ Higgs analysis in the boosted, all-hadronic final state has been used. This use case allowed to validate MLaaS4HEP from the physics point of view and to test its performance. To read ROOT data MLaaS4HEP relies on Uproot and over the years several versions of this library have been released. The migration from Uproot3 to Uproot4 caused a breaking point for the compatibility of MLaaS4HEP with Uproot, and to solve this issue and to exploit the new features introduced by this library, additional work has been done to maintain the code updated. This allowed adding the possibility for the user to create new branches and apply cuts on new and existing branches, responding to the needs of HEP analysts. Moreover, additional improvements to the code have been integrated, e.g. a new ML training procedure and new ML algorithms and frameworks are now supported.

The creation of a SaaS solution for MLaaS4HEP (and therefore a MLaaS) takes place through a process of automating the platform deployment to be delivered to the user, and this work has been done using DODAS as PaaS. The user can access the VM via browser and exploit the JupyterHub interface to upload all the necessary data and run MLaaS4HEP. Subsequently, since such a MLaaS solution should be deployed on a cloud and should be equipped with APIs, a MLaaS4HEP service has been developed and joined by other (micro)services: OAuth2-Proxy, XRootD Proxy, and TFaaS. In this way, a user after being authenticated and authorized can submit MLaaS4HEP workflows using ROOT data previously uploaded or read from distributed data sources, producing trained ML models. These models are then accessible to TFaaS so that the user can use them to make inferences.

A working prototype of this service is currently running on a VM of INFN-Cloud and is compliant to be added to the INFN Cloud portfolio of services.

A

RUN A MLAAS4HEP WORKFLOW

MLaaS4HEP can be installed in any platform, for example using Anaconda and following the recipe reported in Listing 2.

```
conda create -n env
conda activate env
conda config --add channels conda-forge
conda install pyarrow uproot=3.12 numba scikit-learn numpy pandas wget
pip install tensorflow

mkdir work_dir
cd work_dir
git clone https://github.com/vkuznet/MLaaS4HEP.git

export PYTHONPATH=$PYTHONPATH:$PWD/MLaaS4HEP/src/python/
export PATH=$PWD/MLaaS4HEP/bin:$PATH
```

Listing 2: How to install MLaaS4HEP with Anaconda.

The MLaaS4HEP workflow is performed by running the `workflow.py` script in the following way:

```
./workflow.py --files=files.txt --labels=labels.txt --model=model.py --params=params.json
--fout output_model
```

Several files must be passed to the `workflow.py` script as arguments (the name of the files is arbitrary):

- `files.txt` stores the path of the input ROOT files (see an example in Listing 3);
- `labels.txt` stores the labels related to ROOT files content needed for classification problems (see an example in Listing 4);
- `model.py` stores the definition of the custom ML algorithm to use in the training phase (see an example in Listing 5);
- `params.json` stores the parameters on which MLaaS4HEP is based, e.g. number of events to use, chunk size, batch size, the branches of the ROOT files to be selected and/or excluded, redirector path for files located in remote storage (see an example in Listing 6);
- `output_model` indicates where the ML model will be stored after the training procedure.

```
/store/user/lgiommi/ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
/store/user/lgiommi/TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root
```

Listing 3: Example of input ROOT files.

```
1
0
```

Listing 4: Example of labels related to the ROOT files content.

```
from tensorflow import keras

def model(idim):
    ml_model = keras.Sequential([keras.layers.Dense(128,activation='relu',input_shape=(idim,)),
                                 keras.layers.Dropout(0.5),
                                 keras.layers.Dense(64, activation='relu'),
                                 keras.layers.Dropout(0.5),
                                 keras.layers.Dense(1, activation='sigmoid')])
    ml_model.compile(optimizer=keras.optimizers.Adam(lr=1e-3),
                     loss=keras.losses.BinaryCrossentropy(),
                     metrics=[keras.metrics.BinaryAccuracy(name='accuracy'),
                              keras.metrics.AUC(name='auc')])
    return ml_model
```

Listing 5: Example of a MLP definition in Keras.

```
{
   "nevts": 30000,
   "shuffle": true,
   "chunk_size": 10000,
   "epochs": 2,
   "batch_size": 100,
   "branch": "boostedAk8/events",
   "identifier": ["runNo","evtNo","lumi"],
   "selected_branches": "",
   "exclude_branches": "",
   "hist": "pdfs",
   "redirector": "root://xrootd.ba.infn.it",
   "verbose": 1
 }
```

Listing 6: Example of parameters used to run the MLaaS4HEP workflow

As mentioned in Sec. 4.1.1, the MLaaS4HEP workflow is composed of two phases. In the first one, data is read from the input ROOT files to compute and write a specs file. In the second one, chunks of events are created by proportionally reading events from the input ROOT files and appropriately processing them. The output of MLaaS4HEP in verbose mode reports the actions taken during the workflow. An example of such output using the files shown in the listings is shown in Fig. A.1. The output produced for the initialization of the ML model and for the training with the first chunk is shown in Fig.A.2.

An example of MLaaS4HEP usage with open data can be found in [204].

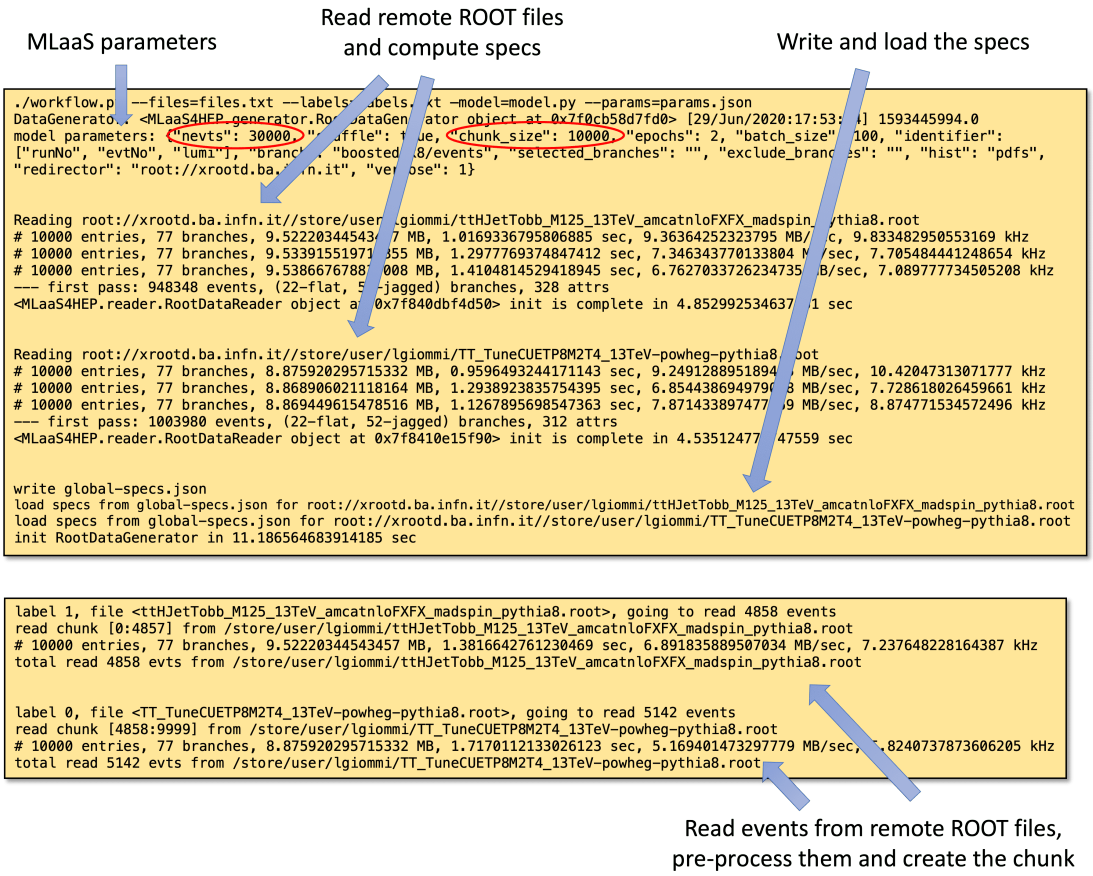


Fig. A.1: Output of the MLaaS4HEP workflow using the files shown in the listings.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               49152
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 57,473
Trainable params: 57,473
Non-trainable params: 0
_________________________________________________________________
```

```
Train on 7000 samples, validate on 3000 samples
Epoch 1/2
7000/7000 [==============================] - 2s 220us/sample - loss: 1.5275 - auc: 0.7845 - accuracy: 0.7307 -
val_loss: 2.5731e-04 - val_auc: 1.0000 - val_accuracy: 1.0000
Epoch 2/2
7000/7000 [==============================] - 0s 20us/sample - loss: 0.1406 - auc: 0.9883 - accuracy: 0.9543 -
val_loss: 8.8477e-06 - val_auc: 1.0000 - val_accuracy: 1.0000
```

Fig. A.2: Output produced for the initialization of the ML model and for the training with the first chunk.

B

DETAILS ON FURTHER DEVELOPMENTS OF MLAAS4HEP

B.1 TRANSITION FROM UPROOT3 TO UPROOT4

MLaaS4HEP needed a rewrite of parts of the code after the transition from Uproot3 to Uproot4 which also allowed to introduce new features in MLaaS4HEP itself, as described in Sec. 4.3. Fig. B.1 shows the timeline of the Uproot and Awkward Array packages.

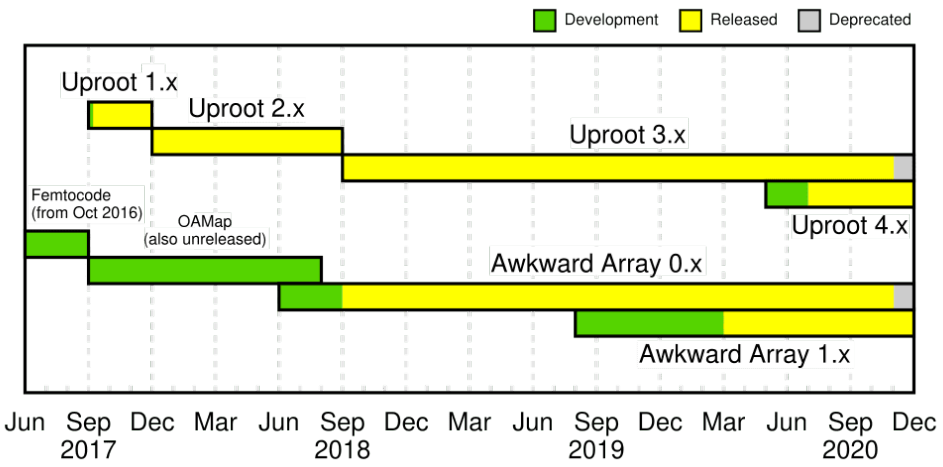


Fig. B.1: Timeline of the Uproot and Awkward Array packages [205].

B.2 ROOT VS UPROOT4 CUTS

In the following, the codes that allowed to verify the validity of the cuts performed by MLaaS4HEP with Uproot4 are reported by comparing them with ROOT. Here the cutting conditions have been reproduced in a generic manner.

```
void script () {

    // setting the output file
    TFile* outputfile = new TFile("output_file.root ", "RECREATE");
    TTree* tree_name = new TTree("tree_name", "tree_name");
    Float_t branch_flat, branch_jagged
    tree_name->Branch("branch_flat", &branch_flat , "branch_flat /F");
    std::vector<int> branch_jagged;
```

```
9
10  // setting the input file
11  TFile* inputfile = new TFile("input_file.root", "READ");
12  TTree* evt = (TTree*) inputfile->Get("input_tree_name");
13
14  // variables setting
15  float mybranch_flat = 0;
16  evt->SetBranchAddress("branch_flat", &mybranch_flat);
17  std::vector<int>* mybranch_jagged = 0;
18  evt->SetBranchAddress("branch_jagged ", &mybranch_jagged);
19  int events_after_cut = 0;
20
21  // loop over the events
22  for (Long64_t ievent = 0; ievent < 10000; ++ ievent){
23      bool keep_mybranch_flat = false ;
24      bool keep_mybranch_jagged = true ;
25
26      // get i-th entry in tree
27      evt->GetEntry (ievent);
28
29      // cut on flat branch
30      if (mybranch_flat >= cut_cond1){
31          keep_mybranch_flat = true ;
32          }
33
34      // a second for loop to access each element of the i-th event for a jagged branch
35      for(int idx =0; idx < mybranch_jagged -> size (); idx ++) {
36          if (mybranch_jagged->at(idx) >= cut_cond2){
37              keep_mybranch_jagged = false ; // all the elements must satisfy the cut
38              }
39          }
40
41      // check the condition of both cuts of the i-th event
42      if ((keep_mybranch_flat & keep_mybranch_jagged) == true){
43      // if the condition is satisfied, fill the output file with the i-th event
44          branch_flat = mybranch_flat;
45          for(int idx =0; idx < mybranch_jagged -> size(); idx ++) {
46              branch_jagged.push_back(mybranch_jagged->at(idx);
47              }
48              events_after_cut += 1;
49              tree_name->Fill();
50              branch_jagged.clear();
51          }
52      } // end loop over events
53
54  cout <<"Events after cut: " << events_after_cut <<endl;
55  inputfile->Close ();
56  outputfile->Write ();
57  outputfile->Close ();
58  delete outputfile ;
59  }
```

Listing 7: Performing cuts on a flat and a Jagged branch using ROOT.

```
import uproot
import awkward as ak

tree = uproot.open ("/path/input_file.root")["input_tree_name"]
chunk_size = 10000

# here we set the cut on flat branch
gen = tree.iterate(step_size = chunk_size, cut="branch_flat >= cut_cond1", library ="ak")
for original_chunk in gen:
    # here we set the cut on the jagged branch
    chunk = original_chunk[ak.all(original_chunk["branch_jagged"] < cut_cond2, axis = 1)]
    x =len(chunk)
    print ("events after cut in a chunk: {}". format(x))
```

Listing 8: Performing cuts on a flat and a Jagged branch using Uproot4.

B.3 NEW TRAINING PROCEDURE

With the recent improvements to the MLaaS4HEP code, a new training approach has been introduced, as described in Sec. 4.3.2. Now the training method the user wants to apply can be selected by adding a new key (called *training*) within the `params.json` file. In case the user decides to apply the newly introduced method, he/she needs to associate the *training* key with the value *standard*. If the value associated with the key is different or the key is not present, the original training mode will be carried out.

B.4 GENERALIZATION TO OTHER ML FRAMEWORKS

The ML algorithm definition must be provided to MLaaS4HEP in an external python file. Such a python file contains the import of all the necessary dependencies and the definition of the algorithm through a function called *model* which takes the needed parameters as arguments. For example, for a MLP in Keras the input dimension is passed as argument (e.g. called *idim* in Listing 5). Finally, once the algorithm is defined, the function needs to return the classifier.

As reported in Sec. 4.3.3, thanks to the new implementations on the code, MLaaS4HEP has been successfully tested using other algorithms from other frameworks and libraries, in addition to MLPs in Keras.

The most common way to use a PyTorch algorithm is to provide its own class definition, containing the structure of the algorithm and how it should be trained, and this can be done easily by using the PyTorch *nn.Module* class. In the file with the definition of the algorithm, the user can also define a custom method to train the model: in this case, the training method should be called *torch_train* (see an example in Listing 9). If no training method is provided, the default one is selected.

```
import torch
import torch.nn as nn
```

```
import torch.nn.functional as fun
import os.path
from datetime import datetime

class ClassifierNN(nn.Module):

    def __init__(self, idim,
                 activation=fun.relu):
        super().__init__()

        self.last_save = None
        self.layout = (idim, 256, 128, 1)
        self.inference_mode = True
        self.activation = activation
        self.layers = nn.ModuleList()
        for num_nodes, num_nodes_next in zip(self.layout[:-1], self.layout[1:]):
            self.layers.append(nn.Linear(num_nodes, num_nodes_next))

    def forward(self, x):
        for layer in self.layers[:-1]:
            if not isinstance(x, torch.Tensor):
                x = torch.Tensor(x)
            x = self.activation(layer(x))

        x = torch.sigmoid(self.layers[-1](x))
        return x

    def train(self, mode=True):
        super(ClassifierNN, self).train()
        self.inference_mode = False

    def eval(self):
        super(ClassifierNN, self).eval()
        self.inference_mode = True

    def torch_train(self, model, train_loader, val_loader):
        epochs = 5
        loss_func = nn.BCELoss()
        optim = torch.optim.Adam(model.parameters(), lr=1e-3)
        self.model.train()
        for epoch in range(1, epochs+1):
            mean_train_loss = 0.0
            for i, (xs, ys) in enumerate(train_loader):
                optim.zero_grad() # reset gradients
                outputs = model(xs)
                train_loss = loss_func(outputs, ys)
                train_loss.backward() # gradient back propagation
                optim.step()
                mean_train_loss = (mean_train_loss * i + float(train_loss)) / (i + 1)

            mean_val_loss = 0.0
            for i, (xs, ys) in enumerate(val_loader):
```

```
55                 outputs = model(xs)
56                 val_loss = loss_func(outputs, ys)
57                 mean_val_loss = (mean_val_loss * i + float(val_loss)) / (i + 1)
58             print('Epoch {}\nMean train/validation loss: {:.4f}/{:.4f}'.format(epoch,
    mean_train_loss, mean_val_loss))
59
60 def model(idim):
61     torch_model = ClassifierNN(idim)
62     return torch_model
```

Listing 9: Example of a MLP written in PyTorch and successfully tested with MLaaS4HEP.

Moreover, the parameters passed to MLaaS4HEP and stored in the `params.json` file need to be adjusted depending on the algorithm used. For example, the presence of *epochs* and *chunk_size* parameter assumes that the defined algorithm knows the concept of epoch and is capable of incremental learning. If not, the user should remove the *epochs* key and change the value associated with the *chunk_size* key by setting it to -1: in this way, the training of the model will be carried out using a single chunk containing all events.

B.5 PROVIDING METRICS SCORE

To print useful metrics (i.e. AUC, Confusion Matrix, Precision, Recall, and F1) for the training and validation sets the user should add *"metrics": true* in the `params.json` file.

C

DETAILS ON THE MLAAS4HEP CLOUDIFICATION WITH DODAS

All the material used for the MLaaS4HEP "cloudification" with DODAS can be found in [187] and other useful information is in [184].

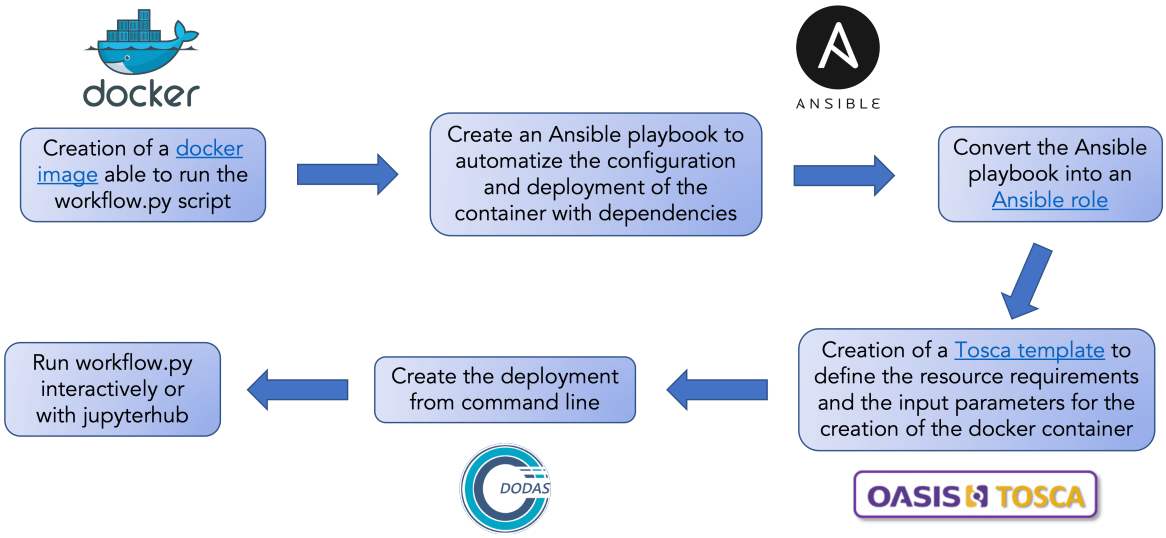


Fig. C.1: Schematic representation of the steps performed to automatize the deployment of the VM equipped with MLaaS4HEP using DODAS.

Fig. C.1 shows the schematic representation of the steps performed to automatize the deployment of the VM equipped with MLaaS4HEP using DODAS. The Dockerfile for the creation of the Docker image to use for the deployment with DODAS is reported in Listing 10 and can be found in [206].

```
FROM continuumio/anaconda3
LABEL maintainer="Luca Giommi luca.giommi3@unibo.it"

# add environment
ENV WDIR=/workarea

# install several useful packages
RUN /opt/conda/bin/conda config --add channels conda-forge
RUN /opt/conda/bin/conda update --all
RUN /opt/conda/bin/conda install xrootd -y
RUN apt-get update && apt-get install -y libgtk2.0-dev && \
    rm -rf /var/lib/apt/lists/* && \
    /opt/conda/bin/conda install jupyter jupyterhub notebook -y && \
```

```
    /opt/conda/bin/conda install vim numpy pandas scikit-learn matplotlib pyyaml h5py keras -y
    && \
    /opt/conda/bin/conda upgrade dask && \
    /opt/conda/bin/conda install backports.lzma -y && \
    pip install gdown && \
    pip install tensorflow imutils

# install voms-clients
RUN apt-get update && apt-get install -y voms-clients

# install uproot
RUN /opt/conda/bin/conda install uproot==3.12

# install pytorch
RUN /opt/conda/bin/conda install pytorch -y

# build mlaas
WORKDIR ${WDIR}
RUN git clone https://github.com/vkuznet/MLaaS4HEP.git
ENV PYTHONPATH="${WDIR}/MLaaS4HEP/src/python:${PYTHONPATH}"
ENV PATH="${WDIR}/MLaaS4HEP/bin:${PATH}"

# run mlaas
WORKDIR ${WDIR}/folder_test
ENTRYPOINT ["python", "../MLaaS4HEP/src/python/MLaaS4HEP/workflow.py"]
```

Listing 10: Dockerfile for the creation of the Docker image to use for the deployment with DODAS.

The Ansible playbook created to automatize the configuration and deployment of the container with dependencies has been converted into an Ansible role which consists of four directories: defaults, meta, tasks, and tests. The relevant content is in two yaml files stored in the defaults and tasks directories. In the former, the default values for several parameters are provided (see Listing 11). In the latter, the actions to be performed in the VM are listed, e.g. download Docker, pull the Docker image, and run the container (see Listing 12)

```
run: "false"
work_dir: /tmp/work_dir
default_container_name: docker
default_container_image: felixfelicislp/mlaas_cloud:mlaas_jupyterhub
default_container_command: --files=files_test.txt --labels=labels_test.txt --model=keras_model.
    py --params=params_test.json
```

Listing 11: Yaml file where default values for several parameters are defined.

```
- name: Install gdown with pip
  pip: name=gdown

- name: install docker [Ubuntu]
  shell:
    cmd: curl https://get.docker.com | sh
    creates: /usr/bin/docker
```

```
    warn: False

- name: install docker-py
  pip: name=docker-py

- name: Creates working directory
  file:
    path: "{{ work_dir }}"
    state: directory

- name: Download folder with data
  command: "{{ item }}"
  with_items:
    - gdown https://drive.google.com/<folder_with_root_files>
    - tar -xzvf folder_test.tar.gz
    - rm folder_test.tar.gz
  args:
    chdir: "{{ work_dir }}"

- name: Pull default Docker image
  docker_image:
    name: "{{ default_container_image }}"
  become: true

- name: Create default containers
  docker_container:
    detach: false
    name: "{{ default_container_name }}"
    image: "{{ default_container_image }}"
    command: "{{ default_container_command }}"
    volumes:
      - "{{ work_dir }}/folder_test:/data/folder_test"
    state: started
  when: run=="true"
```

Listing 12: Yaml file where the tasks to be performed in the VM are defined.

The next step was to create a Tosca template, which consists of two yaml files containing the information about the resource requirements and the needed information to run the Docker container (see Linstings 13 and 14).

```
tosca_definitions_version: tosca_simple_yaml_1_2

imports:
  - https://raw.githubusercontent.com/dodas-ts/dodas-apps/master/tosca-types/dodas_types.yml

node_types:
  tosca.nodes.DODAS.MLaaS:
    derived_from: tosca.nodes.SoftwareComponent
    properties:
      run:
        type: string
        default: "true"
```

```
      work_dir:
        type: string
        default: "/tmp/work_dir"
      default_container_name:
        type: string
        default: "docker"
      default_container_image:
        type: string
        default: "felixfelicislp/mlaas_cloud:mlaas_voms"
      default_container_command:
        type: string
        default: "--files=files_test.txt --labels=labels_test.txt --model=keras_model.py --
    params=params_test.json"
      role_name:
        type: string
        required: false
        default: mlaas_cloud
    artifacts:
      ml_role:
        file: git+https://github.com/lgiommi/mlaas_cloud
        type: tosca.artifacts.AnsibleGalaxy.role
    interfaces:
      Standard:
        start:
          implementation: https://raw.githubusercontent.com/dodas-ts/dodas-apps/master/tosca-
    types/dodas_artifacts/ansible.yaml
          inputs:
            role_name: { get_property: [ SELF, role_name ] }
            run: { get_property: [ SELF, run ] }
            work_dir: { get_property : [SELF, work_dir]}
            default_container_name: { get_property: [ SELF, default_container_name ] }
            default_container_image: { get_property: [ SELF, default_container_image ] }
            default_container_command: { get_property: [ SELF, default_container_command ] }
```

Listing 13: Tosca type.

```
tosca_definitions_version: tosca_simple_yaml_1_2

imports:
  - https://raw.githubusercontent.com/lgiommi/mlaas_cloud/master/lgiommi-custom-type.yml

description: TOSCA template for MLaaS - LUCA

topology_template:
  inputs:
    num_cpus:
      type: integer
      default: 4
    mem_size:
      type: string
      default: "8 GB"
    server_image:
```

```
      type: string
      # default: ost://cloud-api-pub.cr.cnaf.infn.it/94a76d1e-cacb-48ce-ad69-c0b26fd9bb53 # ubuntu 16.04
      default: ost://cloud-api-pub.cr.cnaf.infn.it/9ed8e7ef-e932-4850-9576-805668f7ce25 # ubuntu 18.04

    # MLaaS
    run:
      type: string
      default: "true"
    work_dir:
      type: string
      default: "/tmp/work_dir"
    default_container_name:
      type: string
      default: "docker"
    default_container_image:
      type: string
      default: "felixfelicislp/mlaas_cloud:mlaas_voms"
    default_container_command:
      type: string
      default: "--files=files_test.txt --labels=labels_test.txt --model=keras_model.py --params=params_test.json"

  node_templates:
    mlaas_install:
      type: tosca.nodes.DODAS.MLaaS
      properties:
        run: { get_input: run }
        work_dir: { get_input: work_dir }
        default_container_name: { get_input: default_container_name }
        default_container_image: { get_input: default_container_image }
        default_container_command: { get_input: default_container_command }
      requirements:
        - host: vm_server

    vm_server:
      type: tosca.nodes.indigo.Compute
      capabilities:
        endpoint:
          properties:
            network_name: PUBLIC
            ports:
              jupyter:
                protocol: tcp
                source: 8888
              grafana:
                protocol: tcp
                source: 3000
        scalable:
          properties:
            count: 1
```

```
      host:
        properties:
          instance_type: "dodas.xlarge"
          #num_cpus: { get_input: num_cpus }
          #mem_size: { get_input: mem_size }
      os:
        properties:
          image: { get_input: server_image }

outputs:
  node_ip:
    value: { get_attribute: [ vm_server, public_address, 0 ] }
  node_creds:
    value: { get_attribute: [ vm_server, endpoint, credential, 0 ] }
```

Listing 14: Tosca template.

Then, a configuration file to access DODAS has been written (among the information an access token must be provided) and the deployment has been created from the command line

```
dodas create lgiommi-template.yml
```

When the deployment is completed, the VM can be accessed by command line or using a JupyterHub interface.

In particular, in the latter case, when a user connects to the VM via browser, he/she is redirected to the authentication page with several available possibilities. Once the user is authenticated, he/she can customize the environment by choosing the Docker image and the memory size of the VM. Then, the system prepares and runs the deployment in real time, and once the process is finished, a Jupyter notebook interface is shown to the user with three directories: MLaaS4HEP which contains the code of the MLaaS4HEP framework, whereas the public and private folders contain data to be shared or not to be shared with other users, respectively. In this specific case, some ROOT files were stored in the public directory so that all the users connecting to the same VM can see and share these files. Lastly, the user can open the terminal and run the MLaaS4HEP workflow directly, without having to install anything. Fig. C.2 shows a summary of the steps described so far using the JupyterHub interface. A demo showing these steps can be found in [187].

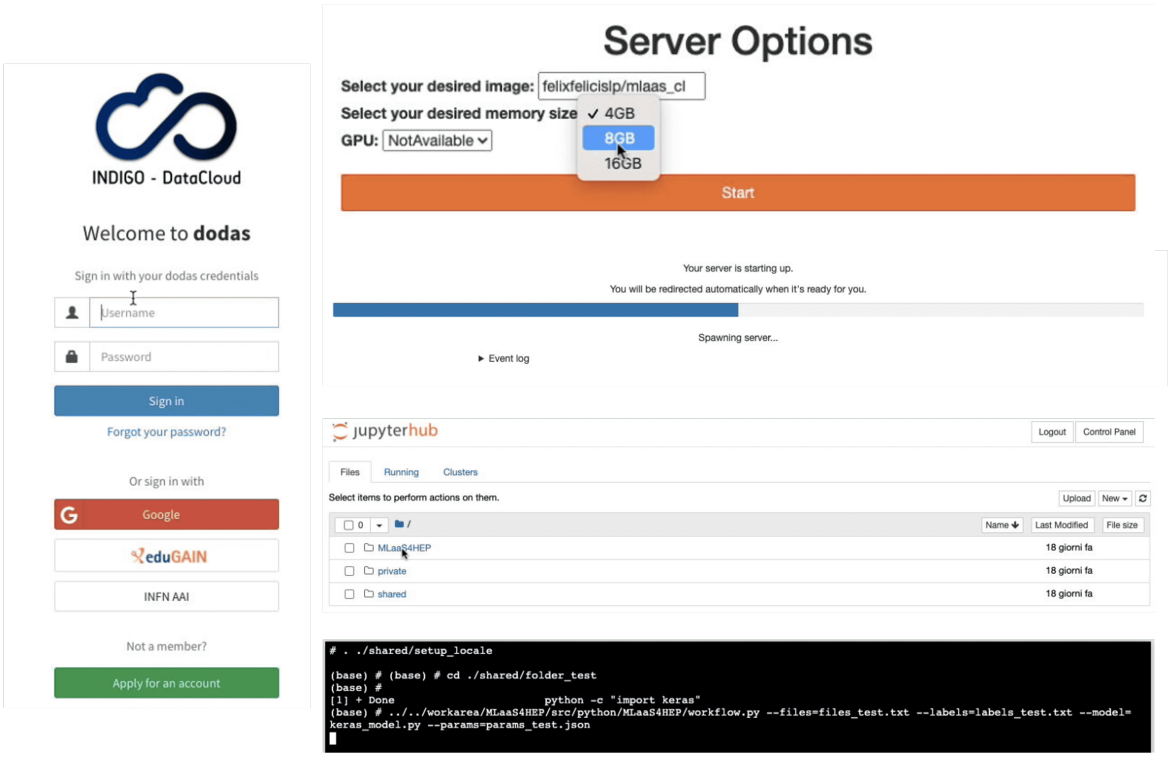


Fig. C.2: Summary of the steps in using the solution provided by DODAS with the JupyterHub interface.

D

DETAILS ON THE CLOUD NATIVE SOLUTION OF MLAAS4HEP

All the material used for the creation of the MLaaS4HEP service can be found in [188].

The Dockerfile used to create the image to run the MLaaS4HEP workflow is reported in Listing 15.

```
FROM redhat/ubi8:latest
LABEL maintainer="Luca Giommi luca.giommi3@unibo.it"
ARG DEBIAN_FRONTEND=noninteractive

RUN yum update && yum install -y python3 python3-pip python3-devel git cmake pkg-config gcc-c++
RUN pip3 install --upgrade pip
RUN pip3 install sklearn tensorflow keras uproot
RUN yum install -y krb5-devel
RUN yum install -y libuuid-devel
RUN yum install -y libxml2-devel
RUN yum install -y openssl-devel
RUN yum install -y systemd-devel
RUN yum install -y zlib-devel
RUN pip3 install xrootd

ENV WDIR=/workarea
WORKDIR ${WDIR}

RUN git clone https://github.com/lgiommi/MLaaS4HEP.git
ENV PYTHONPATH="${WDIR}/MLaaS4HEP/src/python:${PYTHONPATH}"
ENV PATH="${WDIR}/MLaaS4HEP/bin:${PATH}"
WORKDIR ${WDIR}/MLaaS4HEP
RUN git checkout uproot4

ENV X509_CERT_DIR=${WDIR}/certificates
ENV X509_USER_PROXY=${WDIR}/folder_test/x509_proxy
WORKDIR ${WDIR}/folder_test
ENV PYTHONPATH="${WDIR}/folder_test:${PYTHONPATH}"
ENTRYPOINT ["python3", "../MLaaS4HEP/src/python/MLaaS4HEP/workflow.py"]
```

Listing 15: Dockerfile used to run MLaaS4HEP workflows.

The configuration file used to run the OAuth2-Proxy server for TLS connections is reported in Listing 16, whereas the command used to run the proxy is:

```
docker run -v /data/oauth_proxy:/etc --net=host quay.io/oauth2-proxy/oauth2-proxy:latest
```

```
--config=/etc/config_file.cfg  --skip-jwt-bearer-tokens=true --ssl-insecure-skip-verify=true
--tls-min-version=TLS1.3
```

```
provider="oidc"
https_address = ":4433"
redirect_url = "https://90.147.174.27:4433/oauth2/callback"
oidc_issuer_url = "https://cms-auth.web.cern.ch/"
 upstreams = [
     "http://127.0.0.1:8080/"
 ]
 email_domains = [
     "*"
 ]
client_id = "<my_client_ID>"
client_secret = "<my_client_secret>"
cookie_name = "<my_cookie_name>"
cookie_secret = "<my_cookie_secret>"

tls_cert_file = "<my_server_certificate>"
tls_key_file = "<my_server_key>"

standard_logging = true
request_logging = true
auth_logging = true
pass_basic_auth = true
pass_user_headers = true
```

Listing 16: Configuration file of the OAuth2-Proxy server for TLS connections.

The Docker-compose file used to build the XRootD Proxy server that connects xrootd proxy-cache server and X.509 proxy renewer is shown in Listing 17. These services are started simply running:

```
docker-compose up -d
```

Among the actions performed, the action of checking the validity of the proxy and the action of requesting a new proxy should be noted, summarized in the following two commands:

```
voms-proxy-info --exists --valid 6:00 --file /etc/x509proxy/proxy/x509_proxy
```

```
voms-proxy-init --pwstdin --voms cms --cert usercert_test.pem --key usercert_test.key --debug
--out /etc/x509proxy/proxy/x509_proxy
```

```
version: '3.7'
services:
  cache:
    depends_on:
    - x509-refresh
    image: dodasts/cache:v2
    restart: always
```

```
    entrypoint:
      - bash
      - -c
      - "mkdir -p /metadata; chown -R 999:998 /metadata; mkdir -p /data; chown -R 999:998 /data
    ; mkdir -p /etc/my_folder; chown -R 999:998 /etc/my_folder; cp /etc/grid-security/
    certificates/* /etc/my_folder; cd /data; env; chown -R 999:998 /var/log/xrootd; chown -R
    999:998 /etc/xrootd/conf; sudo -E -u xrootd /usr/bin/xrootd -l /var/log/xrootd/xrootd.log -
    c /etc/xrootd/conf/xrootd.cfg -n xrd ; tail -f /var/log/xrootd/xrd/xrootd.log"
    privileged: true
    network_mode: "host"
    volumes:
    - type: bind
      source: ./server_certificates
      target: /etc/my_folder
    - type: bind
      source: ./metadata
      target: /metadata
    - type: bind
      source: ./data
      target: /data
    - type: bind
      source: ./logs
      target: /var/log/xrootd
    - type: bind
      source: ./config
      target: /etc/xrootd/conf
    - type: bind
      source: ./proxy
      target: /etc/x509proxy
    healthcheck:
      test: ["CMD", "/bin/bash", "-c", "pgrep xrootd"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 30s
    env_file:
    - .env
  x509-refresh:
    image: dodasts/x509-renewer:v1
    secrets:
      - my_secret
    restart: always
    entrypoint:
      - bash
      - -c
      - "cd /etc/certs && while true; do chown -R 999:998 /etc/x509proxy/proxy; voms-proxy-info
     --file /etc/x509proxy/proxy --exists --valid 8:00 || cat /run/secrets/my_secret | voms-
    proxy-init --pwstdin --voms cms --cert usercert_test.pem --key usercert_test.key --debug --
    out /etc/x509proxy/proxy/x509_proxy; chown -R 999:998 /etc/x509proxy/proxy; sleep 600; done
    "
    environment:
    - REPO_LIST=cms.cern.ch  grid.cern.ch cms.dodas.infn.it
```

```
    privileged: true
    healthcheck:
      test: ["CMD", "/bin/bash", "-c", "voms-proxy-info --exists --valid 6:00 --file /etc/x509proxy/proxy/x509_proxy"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 30s
    volumes:
    - type: bind
      source: ./certs
      target: /etc/certs
    - type: bind
      source: ./proxy
      target: /etc/x509proxy/proxy

  autoheal:
    restart: always
    environment:
    - AUTOHEAL_CONTAINER_LABEL=all
    volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    image: willfarrell/autoheal

secrets:
  my_secret:
    file: ./certs/test.txt
```

Listing 17: Docker-compose file used to build the XRootD Proxy server

The TFaaS server is run using the following command:

```
docker run -v /data/models_repo:/data/models --rm -p 127.0.0.1:8083:8083
-i -t cmssw/tfaas:v01.01.06
```

Once the servers are up, the overall system can be used to send requests to MLaaS4HEP endpoints [188] and TFaaS endpoints [174]. An example of a possible flux of requests and the corresponding outputs is shown in Listing 18. For a demo version of the commands in Listing 18, see [188, 207].

```
### setup oidc-agent

oidc-agent
eval 'oidc-agent'

OIDC_SOCK=/tmp/oidc-VfwO5P/oidc-agent.55537; export OIDC_SOCK;
OIDCD_PID=55542; export OIDCD_PID;
echo Agent pid $OIDCD_PID
Agent pid 55546


### Obtain a token for MLaaS4HEP_server and TFaaS

```

```
TOKEN_MLAAS=$(oidc-token luca_api2 --aud=a15f41c9-a974-48ec-967a-2a36d255d524)
TOKEN_TFAAS=$(oidc-token luca_api2 --aud=f343be72-8479-4c95-892b-9dfcda0faac1)


### Check if there are models loaded in the TFaaS server

curl -L -k -H "Authorization: Bearer ${TOKEN_TFAAS}" https://90.147.174.27:8081/models

null


### Prepare the folder with the necessary files

ls folder_to_upload

QCD_HT1000to1500.root QCD_HT2000toInf.root  QCD_HT500to700.root TTBar.root    files_local.txt
      labels_local.txt  params_local.json ttH_noDRmatch.root
QCD_HT1500to2000.root QCD_HT300to500.root QCD_HT700to1000.root  ex_keras_model.py files_remote.
    txt  labels_remote.txt params_remote.json  ttH_signal.root


### File with the definition of the ML model

cat ex_keras_model.py

"""
Basic example of ML model implemented via Keras framework
"""
from tensorflow import keras

def model(idim):
    "Simple Keras model for testing purposes"
    ml_model = keras.Sequential([keras.layers.Dense(128, activation='relu',input_shape=(idim,),
    name='inputs'),
                                 keras.layers.Dropout(0.5),
                                 keras.layers.Dense(64, activation='relu'),
                                 keras.layers.Dropout(0.5),
                                 keras.layers.Dense(1, activation='sigmoid')])
    ml_model.compile(optimizer=keras.optimizers.Adam(lr=1e-3), loss=keras.losses.
    BinaryCrossentropy(),
                  metrics=[keras.metrics.BinaryAccuracy(name='accuracy'), keras.metrics.AUC(
    name='auc')])
    return ml_model


### File with the name of local ROOT files

cat files_local.txt

ttH_signal.root
ttH_noDRmatch.root
TTBar.root
```

```
QCD_HT700to1000.root
QCD_HT1500to2000.root
QCD_HT1000to1500.root
QCD_HT2000toInf.root
QCD_HT300to500.root
QCD_HT500to700.root


### File with the path and name of remote ROOT files

cat files_remote.txt

/store/user/lgiommi/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
/store/user/lgiommi/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root
/store/user/lgiommi/flatTree_QCD_HT300to500_TuneCUETP8M1_13TeV-madgraphMLM-pythia8.root
/store/user/lgiommi/flatTree_QCD_HT500to700_TuneCUETP8M1_13TeV-madgraphMLM-pythia8.root
/store/user/lgiommi/flatTree_QCD_HT700to1000_TuneCUETP8M1_13TeV-madgraphMLM-pythia8.root
/store/user/lgiommi/flatTree_QCD_HT1000to1500_TuneCUETP8M1_13TeV-madgraphMLM-pythia8.root
/store/user/lgiommi/flatTree_QCD_HT1500to2000_TuneCUETP8M1_13TeV-madgraphMLM-pythia8.root
/store/user/lgiommi/flatTree_QCD_HT2000toInf_TuneCUETP8M1_13TeV-madgraphMLM-pythia8.root


### File with labels of the ROOT files

cat labels_local.json

1
0
0
0
0
0
0
0


### File with MLaaS4HEP parameters

cat params_local.json

{
    "nevts": 30000,
    "shuffle": true,
    "chunk_size": 10000,
    "epochs": 5,
    "batch_size": 100,
    "identifier":["runNo", "evtNo", "lumi"],
    "branch": "events",
    "selected_branches":"",
    "exclude_branches": "",
    "hist": "pdfs",
```

```
    "redirector": "root://stormgf1.pi.infn.it",
    "verbose": 1
  }


### Load the folder to the MLaaS4HEP server

curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" -F "file=@folder_to_upload.tar.gz" https
    ://90.147.174.27:4433/upload?name=luca

Successfully uploaded!


### Prepare a submission file

cat submit.json

{
    "name": "luca",
    "device": "cpu",
    "memory": "3gb",
    "cpus": "2",
    "files": "files_local.txt",
    "labels": "labels_local.txt",
    "model": "ex_keras_model.py",
    "params": "params_local.json"
  }


### Submit a MLaaS4HEP workflow using the loaded ROOT files

curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" -H "Content-Type: application/json" -d
    @submit.json https://90.147.174.27:4433/submit

{
 "process_name": "luca_1",
 "job_id": 5557
}


### Verify the status of the process

curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" https://90.147.174.27:4433/status_docker?
    process_name=luca_1

{
 "process_name": "luca_1",
 "status": "Up 8 seconds"
}


### Get back and save the logs of the process
```

```
curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" -o logs.txt https://90.147.174.27:4433/logs?process_name=luca_1
cat logs.txt | head -20

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 27881  100 27881    0     0   124k      0 --:--:-- --:--:-- --:--:--  126k

2022-06-14 15:28:44.016611: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory
2022-06-14 15:28:44.016674: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
load ex_keras_model.py <function model at 0x7fd34d81df28> Simple Keras model for testing purposes
DataGenerator: <MLaaS4HEP.generator.RootDataGenerator object at 0x7fd34d7d76a0> [14/Jun/2022:15:28:46] 1655220526.0
model parameters: {"nevts": 30000, "shuffle": true, "chunk_size": 10000, "epochs": 5, "batch_size": 100, "identifier": ["runNo", "evtNo", "lumi"], "branch": "events", "selected_branches": "", "exclude_branches": "", "hist": "pdfs", "redirector": "root://stormgf1.pi.infn.it", "verbose": 1}
Reading ttH_signal.root
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.05584907531738281 sec, 19.808066664389877 MB/sec, 179.05399405758 kHz
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.019169092178344727 sec, 57.7107249909827 MB/sec, 521.6731135184885 kHz
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.016225099563598633 sec, 68.18215214612141 MB/sec, 616.3290376618224 kHz
###total time elapsed for reading + specs computing: 0.09430861473083496; number of chunks 3
###total time elapsed for reading: 0.09124016761779785; number of chunks 3

--- first pass: 38036 events, (29-flat, 0-jagged) branches, 29 attrs
<MLaaS4HEP.reader.RootDataReader object at 0x7fd34d7d79e8> init is complete in 0.1035768985748291 sec
writing specs specs-ttH_signal.json
write specs-ttH_signal.json
Reading ttH_noDRmatch.root
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.046973228454589844 sec, 23.550908537204343 MB/sec, 212.8872195716171 kHz
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.017837047576904297 sec, 62.020477450744515 MB/sec, 560.6308979602749 kHz
# 10000 entries, 29 branches, 1.10626220703125 MB, 0.01699066162109375 sec, 65.11001347103726 MB/sec, 588.5585990121239 kHz


### Download the trained ML model

curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" -o luca_1.tar.gz https://90.147.174.27:4433/model?process_name=luca_1
mkdir -p luca_1 && tar -xvf luca_1.tar.gz -C luca_1
ls luca_1
```

```
197
198   % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
199                                  Dload  Upload   Total   Spent    Left  Speed
200 100  147k  100  147k    0     0   680k      0 --:--:-- --:--:-- --:--:--  717k
201
202 x ./
203 x ./assets/
204 x ./saved_model.pb
205 x ./keras_metadata.pb
206 x ./params.json
207 x ./variables/
208 x ./variables/variables.index
209 x ./variables/variables.data-00000-of-00001
210 assets      keras_metadata.pb params.json   saved_model.pb    variables
211
212
213 ### Check the models loaded in the TFaaS server
214
215 curl -L -k -H "Authorization: Bearer ${TOKEN_TFAAS}" https://90.147.174.27:8081/models
216
217 [{"name":"luca_1","model":"saved_model.pb","labels":"","options":null,"inputNode":"","
    outputNode":"","description":"","timestamp":"2022-06-14 15:29:17.456513917 +0000 UTC m
    =+261933.688268477"}]
218
219
220 ### Choose an event to test inference using the trained ML model
221
222 cat predict_bkg.json
223
224 {
225     "values": [0.19563319290790765, 0.8628343629750731, 0.20469675544301077,
    0.5979233885840486, 0.5624403641089002, 0.4966360687831127, 0.9971232134875923,
    0.9641571184466814, 0.016140890033353065, 0.012642983007060364, 0.044779417127065256,
    0.04623121102305415, 0.027365998536597175, 0.004034759345313149, 0.07267125173331217,
    0.4668842294559054, 0.10894376909915392, 0.044679238817932156, 1.0, 0.9496461903794253,
    0.9982200258458422, 0.5, 0.0, 0.0, 0.35235498377667923, 0.6612158851740676,
    0.6065199265679636, 0.3931907707503391, 0.37482121042755157],
226     "model": "luca_1"
227 }
228
229
230 ### Obtain prediction for the selected event using TFaaS
231
232 curl -L -k -H "Authorization: Bearer ${TOKEN_TFAAS}" -X POST -H "Content-type: application/json
    " -d @predict_bkg.json https://90.147.174.27:8081/json
233
234 [0.08601278]
235
236
237 ### Delete specs files to be prepared for reading remote ROOT files
238
```

```
curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" https://90.147.174.27:4433/delete_specs?name=luca

Specs deletion completed!


### Submit a MLaaS4HEP workflow using remote ROOT files

curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" -H "Content-Type: application/json" -d @submit_remote.json https://90.147.174.27:4433/submit

{
 "process_name": "luca_2",
 "job_id": 6138
}


### Get back and save the logs of the process

curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" -o logs_remote.txt https://90.147.174.27:4433/logs?process_name=luca_2
cat logs_remote.txt | head -20

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2021  100  2021    0     0  12608      0 --:--:-- --:--:-- --:--:-- 13123

2022-06-14 15:29:40.371538: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory
2022-06-14 15:29:40.371628: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
load ex_keras_model.py <function model at 0x7fc254249f28> Simple Keras model for testing purposes
DataGenerator: <MLaaS4HEP.generator.RootDataGenerator object at 0x7fc2541fe630> [14/Jun/2022:15:29:42] 1655220582.0
model parameters: {"nevts": 30000, "shuffle": true, "chunk_size": 10000, "epochs": 5, "batch_size": 100, "identifier": ["runNo", "evtNo", "lumi"], "branch": "boostedAk8/events", "selected_branches": "", "exclude_branches": "", "hist": "pdfs", "redirector": "root://stormgf1.pi.infn.it", "verbose": 1}
Reading root://stormgf1.pi.infn.it//store/user/lgiommi/flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.root
# 10000 entries, 77 branches, 4.863739013671875 MB, 3.1020548343658447 sec, 1.5679087809117251 MB/sec, 3.223669642849594 kHz
# 10000 entries, 77 branches, 4.863739013671875 MB, 2.2151570320129395 sec, 2.1956633066560243 MB/sec, 4.514352642039505 kHz
# 10000 entries, 77 branches, 4.863739013671875 MB, 2.0409839153289795 sec, 2.3830364252958383 MB/sec, 4.899597652335311 kHz
###total time elapsed for reading + specs computing: 8.13843321800232; number of chunks 3
###total time elapsed for reading: 7.35819149017334; number of chunks 3

--- first pass: 948348 events, (22-flat, 52-jagged) branches, 328 attrs
```

```
<MLaaS4HEP.reader.RootDataReader object at 0x7fc2541fe978> init is complete in
    8.150626182556152 sec
writing specs specs-flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.json
write specs-flatTree_ttHJetTobb_M125_13TeV_amcatnloFXFX_madspin_pythia8.json
Reading root://stormgf1.pi.infn.it//store/user/lgiommi/flatTree_TT_TuneCUETP8M2T4_13TeV-powheg-
    pythia8.root


### Delete the loaded folder with all the material and the trained ML models

curl -L -k -H "Authorization: Bearer ${TOKEN_MLAAS}" https://90.147.174.27:4433/delete_folder?
    name=luca
curl -L -k -H "Authorization: Bearer ${TOKEN_TFAAS}" -X DELETE https://90.147.174.27:8081/
    delete -F "model=luca_1"
curl -L -k -H "Authorization: Bearer ${TOKEN_TFAAS}" -X DELETE https://90.147.174.27:8081/
    delete -F "model=luca_2"
curl -L -k -H "Authorization: Bearer ${TOKEN_TFAAS}" https://90.147.174.27:8081/models

Folder deletion completed!
null
```

Listing 18: Example of requests to MLaaS4HEP and TFaaS and the corresponding outputs

Considering the commands run in Listing 18, before deleting the ML models from the TFaaS repository, it is possible to go to the web page of TFaaS `https://90.147.174.27:8081`, check the ML models created by the MLaaS4HEP server (see Fig. D.1), and inspect the ML models representation (see Fig. D.2).

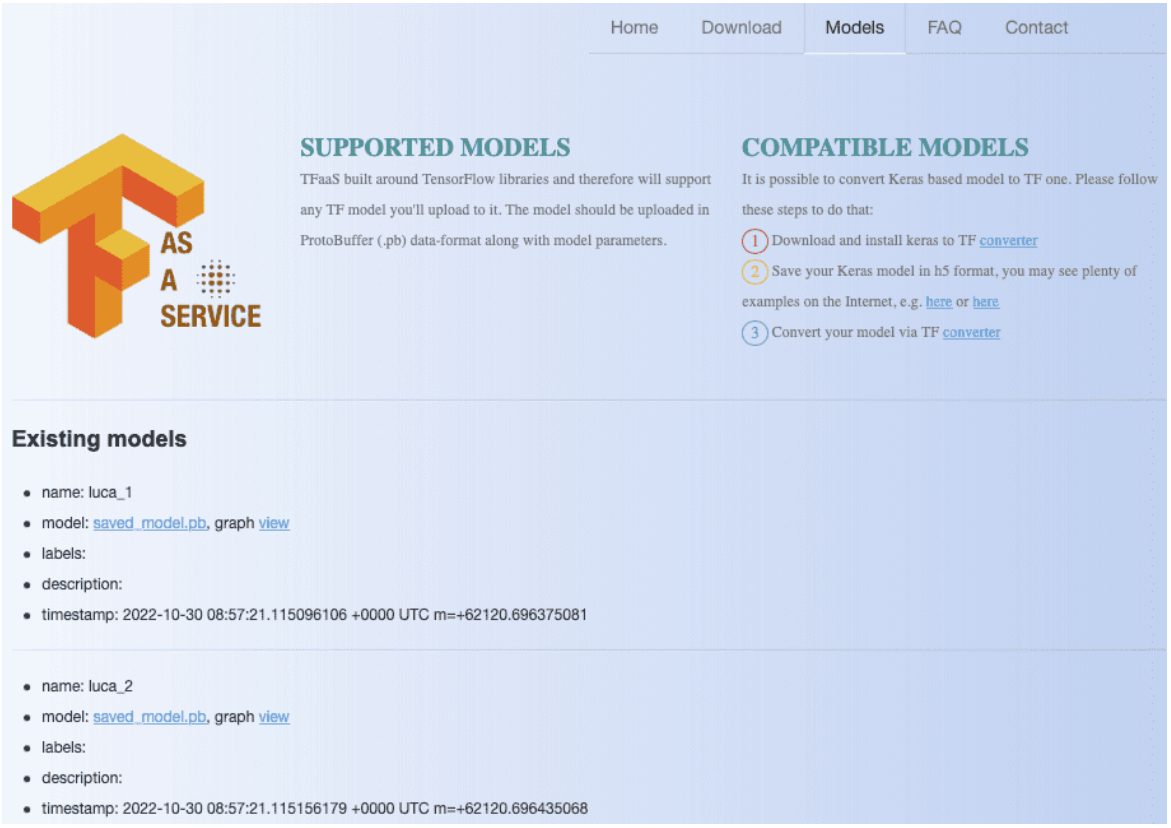


Fig. D.1: ML models created by the MLaaS4HEP service with the commands of Listing 18 and loaded into the TFaaS repository.

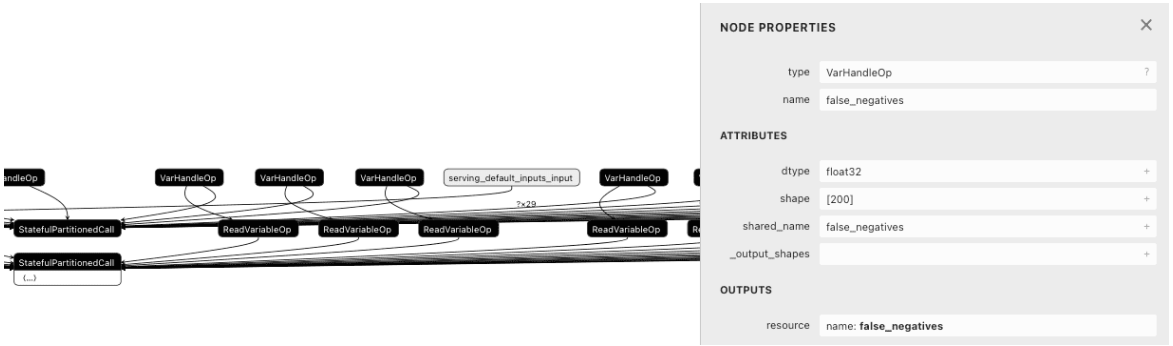


Fig. D.2: Graph of a ML model loaded into TFaaS.

BIBLIOGRAPHY

[1] B. Railey, *Python Machine Learning: A Practical Beginner's Guide to Understanding Machine Learning, Deep Learning and Neural Networks with Python*. Brandon Railey, 2019, ISBN: 1093241446; 9781093241440 (cit. on p. 1).

[2] *Data Science vs. Artificial Intelligence & Machine Learning: What's the Difference?* [Online]. Available: https://csweb.rice.edu/academics/graduate-programs/online-mds/blog/data-science-vs-ai-and-ml (cit. on p. 1).

[3] T. Provost Foster;Fawcett, *Data Science for Business: What you need to know about data mining and data-analytic thinking*, 1. ed. O'Reilly Media, 2013, ISBN: 1449361323; 9781449361327 (cit. on pp. 1, 5).

[4] M. A. Davy Cielen Arno Meysman, *Introducing Data Science: Big Data, Machine Learning, and more, using Python tools*, 1st ed. Manning Publications, 2016, ISBN: 1633430030; 9781633430037 (cit. on p. 2).

[5] E. E. Services, *Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data*, 1st ed. Wiley, 2015, ISBN: 9781118876053; 1118876059; 9781118876138; 111887613X; 9781118876220; 1118876229 (cit. on p. 2).

[6] *Understanding the Data Science Lifecycle*. [Online]. Available: https://www.sudeep.co/data-science/2018/02/09/Understanding-the-Data-Science-Lifecycle.html (cit. on p. 2).

[7] *Data Scientist: The Sexiest Job of the 21st Century*. [Online]. Available: https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century (cit. on p. 3).

[8] *Is Data Scientist Still the Sexiest Job of the 21st Century?* [Online]. Available: https://hbr.org/2022/07/is-data-scientist-still-the-sexiest-job-of-the-21st-century (cit. on p. 3).

[9] *State of Machine Learning and Data Science 2021*. [Online]. Available: https://storage.googleapis.com/kaggle-media/surveys/Kaggle's%20State%20of%20Machine%20Learning%20and%20Data%20Science%202021.pdf (cit. on pp. 4, 10, 11, 21).

[10] *Data Creation and Replication Will Grow at a Faster Rate than Installed Storage Capacity, According to the IDC Global DataSphere and StorageSphere Forecasts*. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS47560321 (cit. on p. 5).

[11] *Worldwide Global DataSphere Forecast, 2021-2025: The World Keeps Creating More Data - Now, What Do We Do with It All?* [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US46410421 (cit. on p. 5).

[12] *Data Never Sleeps 9.0*. [Online]. Available: https://www.domo.com/learn/infographic/data-never-sleeps-9 (cit. on p. 6).

[13] A. Geron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd. O'Reilly Media, 2019, ISBN: 1492032646; 9781492032649. [Online]. Available: libgen.li/file.php?md5=e585d7ea43b3477e96a53a0bc1220f37 (cit. on pp. 6, 14, 16, 17, 26, 30, 33, 35, 36).

[14] *Difference Between Algorithm and Model in Machine Learning*. [Online]. Available: https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/ (cit. on p. 7).

[15] *A Timeline of Machine Learning History*. [Online]. Available: https://www.techtarget.com/whatis/A-Timeline-of-Machine-Learning-History (cit. on p. 7).

[16] *A Short History of Machine Learning – Every Manager Should Read*. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2016/02/19/a-short-history-of-machine-learning-every-manager-should-read/?sh=be388cb15e78 (cit. on p. 7).

[17] *Machine Learning (ML) Market Size, Share & COVID-19 Impact Analysis [...], 2022-2029*. [Online]. Available: https://www.fortunebusinessinsights.com/machine-learning-market-102226 (cit. on p. 9).

[18] *Data from Google Trends on the interest for Machine Learning, Artificial Intelligence and Data Science in Google searches*. [Online]. Available: https://trends.google.com/trends/explore?date=all&q=machine%20learning,artificial%20intelligence,data%20science (cit. on p. 10).

[19] *Big data*. [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/big-data (cit. on p. 9).

[20] *State of the developer nation. 22nd edition. The latest trends from our Q1 2022 survey of 20,000+ developers*. [Online]. Available: https://slashdata-website-cms.s3.amazonaws.com/sample_reports/VZtJWxZw5Q9NDSAQ.pdf (cit. on p. 10).

[21] *Supervised Machine Learning: Regression and Classification*. [Online]. Available: https://www.coursera.org/learn/machine-learning (cit. on p. 12).

[22] J. Hull, *Machine Learning in Business: An Introduction to the World of Data Science*, 2nd ed. 2020, ISBN: 9798644074372 (cit. on p. 16).

[23] H.-T. L. Yaser S. Abu-Mostafa Malik Magdon-Ismail, *Learning From Data: A short course*, 1st ed. AML-Book.com, 2012, ISBN: 1600490069; 9781600490064 (cit. on p. 17).

[24] *8 Feature Engineering Techniques for Machine Learning*. [Online]. Available: https://www.projectpro.io/article/8-feature-engineering-techniques-for-machine-learning/423 (cit. on p. 19).

[25] K. Baker, 'Singular value decomposition tutorial', *The Ohio State University*, vol. 24, 2005 (cit. on p. 22).

[26] *Batch , Mini Batch and Stochastic gradient descent*. [Online]. Available: https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461 (cit. on p. 24).

[27] *Plot different SVM classifiers in the iris dataset*. [Online]. Available: https://scikit-learn.org/stable/auto_examples/svm/plot_iris_svc.html (cit. on p. 28).

[28] S. Shalev-Shwartz Shai;Ben-David, *Understanding Machine Learning*. Cambridge University Press, 2014, ISBN: 9781107057135; 1107057132 (cit. on p. 28).

[29] *The Iris Dataset*. [Online]. Available: https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html (cit. on p. 29).

[30] *The Chain Rule of Calculus for Univariate and Multivariate Functions*. [Online]. Available: https://machinelearningmastery.com/the-chain-rule-of-calculus-for-univariate-and-multivariate-functions/ (cit. on p. 36).

[31] *Softmax Function*. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer (cit. on p. 37).

[32] *Evaluation Metrics For Multi-class Classification*. [Online]. Available: https://www.kaggle.com/code/nkitgupta/evaluation-metrics-for-multi-class-classification (cit. on p. 43).

[33] *Classification: ROC Curve and AUC*. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=en (cit. on p. 42).

[34] *Bias/variance trade-off*. [Online]. Available: https://i.stack.imgur.com/UpdVr.jpg (cit. on p. 44).

[35] A. Lisdorf, *Cloud Computing Basics: A Non-Technical Introduction*. Apress Berkeley, CA, Jan. 2021, ISBN: 978-1-4842-6920-6. DOI: 10.1007/978-1-4842-6921-3 (cit. on pp. 47, 57, 82).

[36] *What is Cloud Computing?* [Online]. Available: https://aws.amazon.com/what-is-cloud-computing/?nc1=h_ls (cit. on p. 47).

[37] T. Erl, Z. Mahmood and R. Puttini, *Cloud Computing: Concepts, Technology & Architecture*, 1st Edition, ser. The Prentice Hall Service Technology Series from Thomas Erl. Prentice Hall, 2013, ISBN: 0133387526; 9780133387520 (cit. on pp. 49, 51, 59, 61).

[38] *Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $500 Billion in 2022*. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-500-billion-in-2022 (cit. on p. 49).

[39] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, Sep. 2011. DOI: 10.6028/NIST.SP.800-145 (cit. on p. 50).

[40] D. Chou, *Cloud Service Models (IaaS, PaaS, SaaS) Diagram*. [Online]. Available: https://dachou.github.io/2018/09/28/cloud-service-models.html (cit. on p. 52).

[41] D. Kirsch and J. Hurwitz, *Cloud Computing For Dummies*. John Wiley & Sons, Inc., 2020, ISBN: 978-1-119-54665-8, 978-1-119-54675-7, 978-1-119-54677-1 (cit. on pp. 51, 71, 79).

[42] M. Ribeiro, K. Grolinger and M. A. Capretz, 'Mlaas: Machine learning as a service', in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015, pp. 896–902. DOI: 10.1109/ICMLA.2015.152 (cit. on p. 52).

[43] *Machine Learning as a Service (MLaaS) Market - Growth, Trends, COVID-19 Impact, and Forecasts (2022 - 2027)*. [Online]. Available: https://www.researchandmarkets.com/reports/4774985/machine-learning-as-a-service-mlaas-market (cit. on p. 53).

[44] Y. Yao, Z. Xiao, B. Wang, B. Viswanath, H. Zheng and B. Y. Zhao, 'Complexity vs. performance: Empirical analysis of machine learning as a service', in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 384–397. DOI: 10.1145/3131365.3131372 (cit. on p. 53).

[45] *Comparing Machine Learning as a Service: Amazon, Microsoft Azure, Google Cloud AI, IBM Watson*. [Online]. Available: https://www.altexsoft.com/blog/datascience/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google-cloud-ai-ibm-watson/ (cit. on p. 54).

[46] *Machine Learning as a Service: What It Is, When to Use It and What Are the Best Tools Out There*. [Online]. Available: https://neptune.ai/blog/machine-learning-as-a-service-what-it-is-when-to-use-it-and-what-are-the-best-tools-out-there (cit. on pp. 53, 55).

[47] *What Is Amazon SageMaker?* [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html (cit. on p. 55).

[48] *What is a public cloud?* [Online]. Available: https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-a-public-cloud/ (cit. on p. 56).

[49] *What is a private cloud?* [Online]. Available: https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-a-private-cloud/ (cit. on p. 56).

[50] *What is HTTP?* [Online]. Available: https://www.cloudflare.com/it-it/learning/ddos/glossary/hypertext-transfer-protocol-http/ (cit. on p. 62).

[51] *An overview of HTTP*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (cit. on p. 62).

[52] *Application Programming Interface (API)*. [Online]. Available: https://www.ibm.com/cloud/learn/api (cit. on p. 63).

[53] *What is API: Definition, Types, Specifications, Documentation*. [Online]. Available: https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/ (cit. on p. 63).

[54] J. Higginbotham, *Principles of Web API Design: Delivering Value with APIs and Microservices*, 1st ed., ser. Addison-Wesley Signature Series (Vernon). Addison-Wesley Professional, 2021, ISBN: 0137355637; 9780137355631 (cit. on p. 64).

[55] *REST APIs*. [Online]. Available: https://www.ibm.com/cloud/learn/rest-apis (cit. on p. 64).

[56] *What is data center automation?* [Online]. Available: https://www.netapp.com/it/data-storage/what-is-data-center-automation/ (cit. on p. 66).

[57] *What is Docker?* [Online]. Available: https://www.ibm.com/uk-en/cloud/learn/docker (cit. on p. 68).

[58] *Docker Engine overview*. [Online]. Available: https://docs.docker.com/engine/ (cit. on p. 68).

[59] *Docker overview*. [Online]. Available: https://docs.docker.com/get-started/overview/ (cit. on pp. 68, 69).

[60] *What is Docker? How Does it Work?* [Online]. Available: https://devopscube.com/what-is-docker/ (cit. on p. 69).

[61] *Podman vs Docker: What are the differences?* [Online]. Available: https://www.imaginarycloud.com/blog/podman-vs-docker/ (cit. on p. 70).

[62] L. Gerber, *Containerization for HPC in the Cloud: Docker vs Singularity - A Comparative Performance Benchmark*, 2018. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1277794/FULLTEXT01.pdf (cit. on pp. 70, 74).

[63] G. M. Kurtzer, V. Sochat and M. W. Bauer, 'Singularity: Scientific containers for mobility of compute', *PLOS ONE*, vol. 12, no. 5, pp. 1–20, May 2017. DOI: 10.1371/journal.pone.0177459 (cit. on p. 70).

[64] *Containers interoperability: How compatible is portable?* [Online]. Available: https://developer.ibm.com/articles/containers-interoperability/ (cit. on p. 71).

[65] *Kubernetes components*. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/ (cit. on p. 71).

[66] *What is a Kubernetes pod?* [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-kubernetes-pod (cit. on p. 72).

[67] *Grid Computing*. [Online]. Available: https://hazelcast.com/glossary/grid-computing/ (cit. on p. 72).

[68] F. Magoules, J. Pan, K. Tan and A. Kumar, *Introduction to Grid Computing*, ser. Chapman & Hall/CRC Numerical Analysis and Scientific Computing Series. CRC Press, 2009, ISBN: 9781420074079. [Online]. Available: https://books.google.it/books?id=a3abbpPHUyoC (cit. on p. 72).

[69] *What Is High Performance Computing (HPC)?* [Online]. Available: https://www.intel.com/content/www/us/en/high-performance-computing/what-is-hpc.html (cit. on pp. 73, 74).

[70] J. D. Hidary, *Quantum Computing: An Applied Approach*. Springer Cham, 2019, ISBN: 978-3-030-23922-0. DOI: 10.1007/978-3-030-23922-0 (cit. on p. 73).

[71] *Cloud-based Grid Computing*. [Online]. Available: https://www.gartner.com/en/information-technology/glossary/cloud-based-grid-computing (cit. on p. 74).

[72] M. Krief, *Learning DevOps: A comprehensive guide to accelerating DevOps culture adoption with Terraform, Azure DevOps, Kubernetes, and Jenkins*, 2nd ed. 2022, ISBN: 9781801818964; 1801818967 (cit. on pp. 74, 77).

[73] J. A. Justin Domingus, *Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud*, 2nd ed. O'Reilly Media, 2022, ISBN: 9781098116828; 1098116828 (cit. on p. 74).

[74] *What Is DevOps?* [Online]. Available: https://orangematter.solarwinds.com/2022/03/21/what-is-devops/ (cit. on p. 75).

[75] *10 Best Source Code Management Tools For Version Control*. [Online]. Available: https://www.softwaretestinghelp.com/best-source-code-management-tools/#1_GitHub (cit. on p. 77).

[76] *Source code management*. [Online]. Available: https://www.atlassian.com/git/tutorials/source-code-management (cit. on p. 77).

[77] J. Geewax, *API Design Patterns*. Manning Publications, 2021, ISBN: 161729585X; 9781617295850 (cit. on p. 79).

[78] T. Coombs, *Cloud Security For Dummies (For Dummies (Computer/Tech))*, 1st ed. John Wiley & Sons, Inc., 2022, ISBN: 9781119790464; 1119790468 (cit. on p. 80).

[79] *What is cloud security?* [Online]. Available: https://www.ibm.com/topics/cloud-security (cit. on p. 81).

[80] C. Dotson, *Practical Cloud Security*. O'Reilly Media, Inc., March 2019, ISBN: 9781492037507; 1492037508 (cit. on p. 81).

[81] *SSL/TLS Encryption*. [Online]. Available: https://www.f5.com/services/resources/glossary/ssl-tls-encryption (cit. on p. 84).

[82] *How Does HTTPS Work to Improve Website Security?* [Online]. Available: https://sectigostore.com/page/how-does-https-work/ (cit. on p. 84).

[83] *TLS Basics*. [Online]. Available: https://www.internetsociety.org/deploy360/tls/basics/ (cit. on p. 84).

[84] P. Siriwardena, 'Advanced API Security: OAuth 2.0 And Beyond', 2020. DOI: 10.1007/978-1-4842-2050-4 (cit. on pp. 84, 93, 94).

[85] A. H. Yvonne Wilson, *Solving Identity Management In Modern Applications: Demystifying OAuth 2.0, OpenID Connect, And SAML 2.0*, 1st ed. Apress, 2019, ISBN: 148425094X; 9781484250945; 9781484250952; 1484250958. DOI: 10.1007/978-1-4842-5095-2 (cit. on pp. 86, 90, 93).

[86] *JSON Web Tokens*. [Online]. Available: https://auth0.com/docs/secure/tokens/json-web-tokens (cit. on p. 92).

[87] *JSON Web Token (JWT)*. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519 (cit. on p. 92).

[88] *JWT*. [Online]. Available: https://jwt.io (cit. on p. 93).

[89] *SAML vs. OpenID (OIDC)*. [Online]. Available: https://auth0.com/intro-to-iam/saml-vs-openid-connect-oidc/ (cit. on p. 93).

[90] *OpenID Connect Core 1.0*. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims (cit. on p. 94).

[91] *ID Token and Access Token: What's the Difference?* [Online]. Available: https://auth0.com/blog/id-token-access-token-what-is-the-difference/ (cit. on p. 95).

[92] CERN, 'CERN Annual report 2021', CERN, Geneva, Tech. Rep., 2022. DOI: 10.17181/AnnualReport2021. [Online]. Available: https://cds.cern.ch/record/2807619 (cit. on p. 97).

[93] D. Griffiths, *Introduction to Elementary Particles*, 2nd. Wiley-VCH, 2008, ISBN: 3527406018; 9783527406012 (cit. on p. 98).

[94] R. L. Workman *et al.* [Particle Data Group Collaboration], 'Review of Particle Physics', *PTEP*, vol. 2022, p. 083C01, 2022. DOI: 10.1093/ptep/ptac097 (cit. on pp. 98, 169).

[95] L. Giommi, 'Prototype of machine learning "as a service" for cms physics in signal vs background discrimination', M.S. thesis, University of Bologna, 2018. [Online]. Available: http://amslaurea.unibo.it/15803/ (cit. on pp. 98, 108, 109, 141, 142).

[96] F. Iemmi, 'Search for tth(bb) events at 13 tev in the fully hadronic final state targeting large-radius jets and a resolved decay of the higgs boson with the cms experiment', Ph.D. dissertation, University of Bologna, 2021. [Online]. Available: http://amsdottorato.unibo.it/9628/ (cit. on pp. 99, 103).

[97] C. Campagnari and M. Franklin, 'The Discovery of the top quark', *Rev. Mod. Phys.*, vol. 69, pp. 137–212, 1997. DOI: 10.1103/RevModPhys.69.137. arXiv: hep-ex/9608003 (cit. on p. 99).

[98] F. Iemmi, 'Measurement of the inclusive and differential t/t cross sections at 13 tev in the all-jets boosted regime with cms', M.S. thesis, University of Bologna, 2016. [Online]. Available: http://amslaurea.unibo.it/12032/ (cit. on p. 101).

[99] G. Aad *et al.* [ATLAS Collaboration], 'Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC', *Physics Letters B*, vol. 716, no. 1, pp. 1–29, 2012, ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.020 (cit. on p. 101).

[100] S. Chatrchyan *et al.* [CMS Collaboration], 'Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC', *Physics Letters B*, vol. 716, no. 1, pp. 30–61, 2012, ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.021 (cit. on p. 101).

[101] *Learning to discover: the Higgs boson machine learning challenge*. [Online]. Available: https://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf (cit. on pp. 101, 103).

[102] R. L. Workman *et al.* [Particle Data Group Collaboration], 'Handbook of LHC Higgs cross sections: 4. Deciphering the nature of the Higgs sector', *CERN Yellow Reports: Monographs*, vol. 2, 2017. DOI: 10.1093/ptep/ptac097 (cit. on p. 102).

[103] S. Chatrchyan *et al.* [CMS Collaboration], 'Search for $t\bar{t}H$ production in the $H \to b\bar{b}$ decay channel with $\sqrt{s} = 13$ TeV pp collisions at the CMS experiment', [Online]. Available: https://cds.cern.ch/record/2139578/files/HIG-16-004-pas.pdf (cit. on p. 103).

[104] F. Iemmi, '$t\bar{t}H$ associated production in the all-jets final state with the CMS experiment', *Nuovo Cim. C*, vol. 43, no. 2-3, p. 77, 2020. DOI: 10.1393/ncc/i2020-20077-4 (cit. on p. 103).

[105] *The Large Hadron Collider*. [Online]. Available: https://home.cern/science/accelerators/large-hadron-collider (cit. on p. 104).

[106] *Facts and figures about the LHC*. [Online]. Available: https://home.cern/resources/faqs/facts-and-figures-about-lhc (cit. on p. 105).

[107] *CERN's accelerator complex*. [Online]. Available: https://home.cern/science/accelerators/accelerator-complex (cit. on p. 105).

[108] M. Paladino, 'Machine learning "as a service" for high energy physics (mlaas4hep): Evolution of a framework for ml-based physics analyses', M.S. thesis, University of Bologna, 2022. [Online]. Available: https://amslaurea.unibo.it/26129/ (cit. on pp. 104, 108, 155, 160, 161).

[109] *The HL-LHC project*. [Online]. Available: https://hilumilhc.web.cern.ch/content/hl-lhc-project (cit. on p. 106).

[110] *ALICE*. [Online]. Available: https://home.cern/science/experiments/alice (cit. on p. 106).

[111] *ATLAS*. [Online]. Available: https://home.cern/science/experiments/atlas (cit. on p. 106).

[112] *LHCb*. [Online]. Available: https://home.cern/science/experiments/lhcb (cit. on p. 106).

[113] *CMS*. [Online]. Available: https://home.cern/science/experiments/cms (cit. on p. 107).

[114] *CMS detector*. [Online]. Available: https://cms.cern/detector (cit. on p. 108).

[115] *Triggering and data acquisition*. [Online]. Available: https://cms.cern/detector/triggering-and-data-acquisition (cit. on p. 108).

[116] *WLCG Document Repository / reporting / accounting*. [Online]. Available: https://wlcg-docs.web.cern.ch/?dir=reporting%2Faccounting (cit. on p. 109).

[117] *WLCG Token Transition Timeline*. DOI: 10.5281/zenodo.7014668 (cit. on p. 110).

[118] *Tiers of WLCG*. [Online]. Available: https://wlcg-public.web.cern.ch/tiers (cit. on p. 111).

[119] C. O. Software and Computing, 'Evolution of the CMS Computing Model towards Phase-2', CERN, Geneva, Tech. Rep., 2021. [Online]. Available: https://cds.cern.ch/record/2751565 (cit. on p. 111).

[120] C. O. Software and Computing, 'CMS Phase-2 Computing Model: Update Document', CERN, Geneva, Tech. Rep., 2022. [Online]. Available: https://cds.cern.ch/record/2815292 (cit. on pp. 112, 113, 115, 119).

[121] S. Campana, I. Bird and B. Panzer-Steindel, *Overview of the WLCG strategy towards HL-LHC computing - April 2020 LHCC review*, Sep. 2021. DOI: 10.5281/zenodo.5499655. [Online]. Available: https://doi.org/10.5281/zenodo.5499655 (cit. on pp. 113, 118).

[122] *Data Formats and Data Tiers*. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookDataFormats%5C#AboutTiers (cit. on p. 114).

[123] M. Peruzzi, 'The NanoAOD event data format in CMS', Presentation at the 19th International Workshop on Advanced Computing and Analysis Techniques in Physics Research Saas-Fee, Switzerland, 2019. [Online]. Available: https://indico.cern.ch/event/708041/contributions/3276172/attachments/1809727/2955119/peruzzi_NanoAOD_ACAT_2019.pdf (cit. on p. 115).

[124] M. Peruzzi, G. Petrucciani and A. Rizzi [CMS Collaboration], 'The NanoAOD event data format in CMS', *J. Phys. Conf. Ser.*, vol. 1525, no. 1, p. 012 038, 2020. DOI: 10.1088/1742-6596/1525/1/012038 (cit. on p. 114).

[125] A. Rizzi, G. Petrucciani and M. Peruzzi [CMS Collaboration], 'A further reduction in CMS event data for analysis: the NANOAOD format', *EPJ Web Conf.*, vol. 214, 06021. 6 p, 2019. DOI: 10.1051/epjconf/201921406021. [Online]. Available: https://cds.cern.ch/record/2699585 (cit. on p. 114).

[126] *The CMS NanoAOD data tier*. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookNanoAOD (cit. on p. 115).

[127] S. Chatrchyan *et al.* [CMS Collaboration], 'The CMS Experiment at the CERN LHC', *JINST*, vol. 3, S08004, 2008. DOI: 10.1088/1748-0221/3/08/S08004 (cit. on p. 117).

[128] *CMS Offline and Computing Public Results. Approved HL-LHC resource projections (July 2022)*. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults (cit. on pp. 122, 123).

[129] *CMSSW Application Framework*. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCMSSWFramework (cit. on p. 121).

[130] G. Amadio *et al.* [ROOT Team Collaboration], 'Software Challenges For HL-LHC Data Analysis', May 2020. arXiv: 2004.07675 [physics.data-an] (cit. on p. 121).

[131] *About ROOT*. [Online]. Available: https://root.cern/about/ (cit. on p. 121).

[132] A. Naumann *et al.*, 'ROOT for the HL-LHC: data format', Apr. 2022. arXiv: 2204.04557 [hep-ex] (cit. on p. 121).

[133] D. Piparo, P. Canal, E. Guiraud, X. Valls Pla, G. Ganis, G. Amadio, A. Naumann and E. Tejedor, 'RDataFrame: Easy Parallel ROOT Analysis at 100 Threads', *EPJ Web Conf.*, vol. 214, A. Forti, L. Betev, M. Litmaath, O. Smirnova and P. Hristov, Eds., p. 06 029, 2019. DOI: 10.1051/epjconf/201921406029 (cit. on p. 124).

[134] O. Shadura, B. Krikler and G. A. Stewart, 'HL-LHC Computing Review Stage 2, Common Software Projects: Data Science Tools for Analysis', in *LHCC Review of HL-LHC Computing*, J. Pivarski, E. Rodrigues and K. Pedro, Eds., Feb. 2022. arXiv: 2202.02194 [physics.data-an] (cit. on pp. 124, 125, 126).

[135] J. Albrecht *et al.* [HEP Software Foundation Collaboration], 'A Roadmap for HEP Software and Computing R&D for the 2020s', *Comput. Softw. Big Sci.*, vol. 3, no. 1, p. 7, 2019. DOI: 10.1007/s41781-018-0018-8. arXiv: 1712.06982 [physics.comp-ph] (cit. on pp. 124, 126).

[136] *Awkward*. [Online]. Available: https://awkward-array.readthedocs.io/en/stable/ (cit. on p. 125).

[137] *Uproot*. [Online]. Available: https://uproot.readthedocs.io/en/latest/index.html (cit. on p. 125).

[138] D. H. Guest *et al.*, *Lwtnn/lwtnn: Version 2.7.1*, version v2.7.1, 2018. DOI: 10.5281/zenodo.1403888 (cit. on p. 126).

[139] K. Albertsson *et al.*, 'Machine learning in high energy physics community white paper', 2018. DOI: 10.48550/ARXIV.1807.02876. [Online]. Available: https://arxiv.org/abs/1807.02876 (cit. on pp. 126, 127, 130).

[140] V. Kuznetsov, L. Giommi and D. Bonacorsi, 'MLaaS4HEP: Machine Learning as a Service for HEP', *Comput. Softw. Big Sci.*, vol. 5, no. 1, p. 17, 2021. DOI: 10.1007/s41781-021-00061-3. arXiv: 2007.14781 [hep-ex] (cit. on pp. 127, 141, 142, 145, 146, 149, 150, 151, 152, 153, 154).

[141] M. Migliorini, R. Castellotti, L. Canali and M. Zanetti, 'Machine Learning Pipelines with Modern Big Data Tools for High Energy Physics', *Comput. Softw. Big Sci.*, vol. 4, no. 1, p. 8, 2020. DOI: 10.1007/s41781-020-00040-0. arXiv: 1909.10389 [cs.DC] (cit. on p. 128).

[142] A. M. Sirunyan *et al.* [CMS Collaboration], 'A deep neural network to search for new long-lived particles decaying to jets', *Mach. Learn. Sci. Tech.*, vol. 1, p. 035 012, 2020. DOI: 10.1088/2632-2153/ab9023. arXiv: 1912.12238 [hep-ex] (cit. on p. 128).

[143] Golubovic, Dejan and Rocha, Ricardo, 'Training and serving ml workloads with kubeflow at cern', *EPJ Web Conf.*, vol. 251, p. 02 067, 2021. DOI: 10.1051/epjconf/202125102067. [Online]. Available: https://doi.org/10.1051/epjconf/202125102067 (cit. on pp. 128, 140).

[144] *Guidelines for the ml.cern.ch service*. [Online]. Available: https://ml.docs.cern.ch (cit. on p. 128).

[145] J. Pivarski, P. Elmer, B. Bockelman and Z. Zhang, 'Fast access to columnar, hierarchically nested data via code transformation', 2017. DOI: 10.48550/ARXIV.1708.08319 (cit. on p. 129).

[146] *Building a testbed for an Analysis Facility at INFN*. [Online]. Available: https://infn-cms-analysisfacility.readthedocs.io/en/latest/introduction/ (cit. on p. 129).

[147] D. Guest, K. Cranmer and D. Whiteson, 'Deep Learning and its Application to LHC Physics', *Ann. Rev. Nucl. Part. Sci.*, vol. 68, pp. 161–181, 2018. DOI: 10.1146/annurev-nucl-101917-021019. arXiv: 1806.11484 [hep-ex] (cit. on p. 130).

[148] *A Living Review of Machine Learning for Particle Physics*. [Online]. Available: https://iml-wg.github.io/HEPML-LivingReview/ (cit. on pp. 130, 131, 132).

[149] P. Baldi, P. Sadowski and D. Whiteson, 'Searching for Exotic Particles in High-Energy Physics with Deep Learning', *Nature Commun.*, vol. 5, p. 4308, 2014. DOI: 10.1038/ncomms5308. arXiv: 1402.4735 [hep-ph] (cit. on pp. 131, 132).

[150] R. Abbasi *et al.* [IceCube Collaboration], 'Graph Neural Networks for Low-Energy Event Classification & Reconstruction in IceCube', Sep. 2022. arXiv: 2209.03042 [hep-ex] (cit. on p. 132).

[151] J. Cogan, M. Kagan, E. Strauss and A. Schwartzman, 'Jet-Images: Computer Vision Inspired Techniques for Jet Tagging', *JHEP*, vol. 02, p. 118, 2015. DOI: 10.1007/JHEP02(2015)118. arXiv: 1407.5675 [hep-ph] (cit. on p. 133).

[152] M. Dordevic [CMS Collaboration], 'The CMS Particle Flow Algorithm', *EPJ Web Conf.*, vol. 191, V. E. Volkova, Y. V. Zhezher, D. G. Levkov, V. A. Rubakov and V. A. Matveev, Eds., p. 02 016, 2018. DOI: 10.1051/epjconf/201819102016 (cit. on p. 134).

[153] J. Pata, J. Duarte, J.-R. Vlimant, M. Pierini and M. Spiropulu, 'MLPF: Efficient machine-learned particle-flow reconstruction using graph neural networks', *Eur. Phys. J. C*, vol. 81, no. 5, p. 381, 2021. DOI: 10.1140/epjc/s10052-021-09158-w. arXiv: 2101.08578 [physics.data-an] (cit. on p. 134).

[154] J. Pata, J. Duarte, F. Mokhtar, E. Wulff, J. Yoo, J.-R. Vlimant, M. Pierini and M. Girone [CMS Collaboration], 'Machine Learning for Particle Flow Reconstruction at CMS', in *20th International Workshop on Advanced Computing and Analysis Techniques in Physics Research: AI Decoded - Towards Sustainable, Diverse, Performant and Effective Scientific Computing*, Mar. 2022. arXiv: 2203.00330 [physics.data-an] (cit. on p. 134).

[155] T. Diotalevi, 'Application of deep learning techniques in the search for bsm higgs bosons in the $\mu\mu$ final state in cms', Ph.D. dissertation, University of Bologna, 2022. [Online]. Available: http://amsdottorato.unibo.it/10356/ (cit. on p. 134).

[156] J. Duarte *et al.*, 'Fast inference of deep neural networks in FPGAs for particle physics', *JINST*, vol. 13, no. 07, P07027, 2018. DOI: 10.1088/1748-0221/13/07/P07027. arXiv: 1804.06913 [physics.ins-det] (cit. on p. 135).

[157] T. Diotalevi, M. Lorusso, R. Travaglini, C. Battilana and D. Bonacorsi [CMS Collaboration], 'Deep Learning fast inference on FPGA for CMS Muon Level-1 Trigger studies', *PoS*, vol. ISGC2021, p. 005, 2021. DOI: 10.22323/1.378.0005 (cit. on p. 135).

[158] G. Aad *et al.*, 'Artificial Neural Networks on FPGAs for Real-Time Energy Reconstruction of the ATLAS LAr Calorimeters', *Comput. Softw. Big Sci.*, vol. 5, no. 1, p. 19, 2021. DOI: 10.1007/s41781-021-00066-y (cit. on p. 135).

[159] M. Paganini, L. de Oliveira and B. Nachman, 'Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters', *Phys. Rev. Lett.*, vol. 120, no. 4, p. 042 003, 2018. DOI: 10.1103/PhysRevLett.120.042003. arXiv: 1705.02355 [hep-ex] (cit. on p. 135).

[160] L. de Oliveira, M. Paganini and B. Nachman, 'Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis', *Comput. Softw. Big Sci.*, vol. 1, no. 1, p. 4, 2017. DOI: 10.1007/s41781-017-0004-6. arXiv: 1701.05927 [stat.ML] (cit. on p. 135).

[161] V. Wachirapusitanand [CMS Collaboration], 'Machine Learning applications for Data Quality Monitoring and Data Certification within CMS', CERN, Geneva, Tech. Rep., 2022. [Online]. Available: https://cds.cern.ch/record/2801635 (cit. on p. 136).

[162] V. Kuznetsov, T. Li, L. Giommi, D. Bonacorsi and T. Wildish, 'Predicting dataset popularity for the CMS experiment', *J. Phys. Conf. Ser.*, vol. 762, no. 1, L. Salinas and C. Torres, Eds., p. 012 048, 2016. DOI: 10.1088/1742-6596/762/1/012048. arXiv: 1602.07226 [physics.data-an] (cit. on p. 136).

[163] A. Di Girolamo *et al.*, 'Preparing Distributed Computing Operations for the HL-LHC Era With Operational Intelligence', *Front. Big Data*, vol. 4, p. 753 409, 2022. DOI: 10.3389/fdata.2021.753409 (cit. on p. 136).

[164] L. Giommi, D. Bonacorsi, T. Diotalevi, L. Rinaldi, L. Morganti, A. Falabella, E. Ronchieri, A. Ceccanti, B. Martelli and S. Tisbeni, 'Towards Predictive Maintenance with Machine Learning at the INFN-CNAF computing centre', *PoS*, vol. ISGC2019, p. 003, 2019. DOI: 10.22323/1.351.0003 (cit. on p. 137).

[165] P. Artoisenet, V. Lemaitre, F. Maltoni and O. Mattelaer, 'Automation of the matrix element reweighting method', *JHEP*, vol. 12, p. 068, 2010. DOI: 10.1007/JHEP12(2010)068. arXiv: 1007.3300 [hep-ph] (cit. on p. 137).

[166] J. Bendavid, 'Efficient Monte Carlo Integration Using Boosted Decision Trees and Generative Deep Neural Networks', Jun. 2017. arXiv: 1707.00028 [hep-ph] (cit. on p. 138).

[167] *Neural Information Processing Systems (NeurIPS)*. [Online]. Available: https://nips.cc (cit. on p. 138).

[168] *Open Data CERN*. [Online]. Available: https://opendata.cern.ch (cit. on p. 138).

[169] *CERN openlab*. [Online]. Available: https://home.cern/science/computing/cern-openlab (cit. on p. 139).

[170] *Inter-Experimental LHC Machine Learning Working Group*. [Online]. Available: https://iml.web.cern.ch/homepage (cit. on p. 139).

[171] *Resources at CERN: a user experience*. [Online]. Available: https://indico.cern.ch/event/1173110/contributions/4934099/attachments/2475317/4247725/TrainingAtCern.pdf (cit. on p. 140).

[172] L. Giommi, D. Bonacorsi and V. Kuznetsov, 'Prototype of Machine Learning "as a Service" for CMS Physics in Signal vs Background discrimination', *PoS*, vol. LHCP2018, p. 093, 2018. DOI: 10.22323/1.321.0093 (cit. on pp. 142, 168).

[173] *MLaaS4HEP*. [Online]. Available: https://github.com/lgiommi/MLaaS4HEP (cit. on pp. 142, 143, 155).

[174] *TFaaS*. [Online]. Available: https://github.com/vkuznet/TFaaS (cit. on pp. 142, 147, 194).

[175] *Uproot3*. [Online]. Available: https://github.com/scikit-hep/uproot3 (cit. on p. 143).

[176] *TFaaS at CERN*. [Online]. Available: https://cms-tfaas.cern.ch/ (cit. on p. 147).

[177] *TFaaS at INFN*. [Online]. Available: https://90.147.174.27:8081/ (cit. on p. 147).

[178] V. Kuznetsov, *Machine Learning as a Service for HEP*, 2018. DOI: 10.48550/ARXIV.1811.04492. [Online]. Available: https://arxiv.org/abs/1811.04492 (cit. on p. 153).

[179] L. Giommi, D. Spiga, V. Kuznetsov, D. Bonacorsi and M. Paladino, 'Cloud native approach for Machine Learning as a Service for High Energy Physics', *PoS*, vol. ISGC2022, p. 012, 2022. DOI: 10.22323/1.415.0012 (cit. on pp. 156, 165, 168).

[180] *Numpy functions usable with Uproot*. [Online]. Available: https://github.com/scikit-hep/uproot5/blob/main/src/uproot/language/python.py#L257 (cit. on p. 157).

[181] *CMS Machine Learning Documentation. MLaaS4HEP*. [Online]. Available: https://cms-ml.github.io/documentation/training/MLaaS4HEP.html (cit. on pp. 159, 169).

[182] *The Higgs Boson ML Challenge using MLaaS4HEP*. [Online]. Available: https://github.com/lgiommi/MLaaS4HEP/blob/uproot4/doc/Higgs_challenge.ipynb (cit. on p. 161).

[183] L. Giommi, V. Kuznetsov, D. Bonacorsi and D. Spiga, 'Machine Learning as a Service for High Energy Physics on heterogeneous computing resources', *PoS*, vol. ISGC2021, p. 019, 2021. DOI: 10.22323/1.378.0019 (cit. on pp. 162, 168).

[184] *Dynamic On Demand Analysis Service*. [Online]. Available: https://dodas-ts.github.io/dodas-apps/ (cit. on pp. 162, 183).

[185] *INFN Cloud*. [Online]. Available: https://www.cloud.infn.it (cit. on p. 162).

[186] D. Salomoni, I. Campos, L. Gaido, G. Donvito, M. Antonacci *et al.*, *INDIGO-Datacloud: foundations and architectural description of a Platform as a Service oriented to scientific computing*, 2016. DOI: 10.48550/ARXIV.1603.09536. [Online]. Available: https://arxiv.org/abs/1603.09536 (cit. on p. 163).

[187] *Material for MLaaS4HEP cloudification with DODAS*. [Online]. Available: https://github.com/lgiommi/mlaas_cloud (cit. on pp. 163, 183, 188).

[188] *MLaaS4HEP_server*. [Online]. Available: https://github.com/lgiommi/MLaaS4HEP_server (cit. on pp. 166, 191, 194).

[189] *auth-proxy-server*. [Online]. Available: https://github.com/dmwm/auth-proxy-server (cit. on p. 166).

[190] *OAuth2-Proxy server*. [Online]. Available: https://oauth2-proxy.github.io/oauth2-proxy/ (cit. on p. 166).

[191] *oidc-agent*. [Online]. Available: https://indigo-dc.gitbook.io/oidc-agent/ (cit. on p. 166).

[192] *compose-xrootd*. [Online]. Available: https://github.com/comp-dev-cms-ita/compose-xrootd (cit. on p. 167).

[193] L. Giommi, 'Prototype of a cloud native solution of Machine Learning as Service for HEP', Poster presentation at the XLI International Conference on High Energy Physics (ICHEP), Bologna (Italy). Submitted to the proceedings., 2022. [Online]. Available: https://agenda.infn.it/event/28874/contributions/168847/ (cit. on pp. 167, 168).

[194] *Scikit-HEP/uproot chat on Gitter*. [Online]. Available: https://gitter.im/Scikit-HEP/uproot (cit. on p. 168).

[195] L. Giommi, 'MLaaS4HEP: Machine Learning as a Service for HEP', Presentation at the 4^{th} Inter-experiment Machine Learning (IML) workshop, virtual event, 2020. [Online]. Available: https://indico.cern.ch/event/852553/contributions/4060360/ (cit. on p. 168).

[196] L. Giommi, 'MLaaS4HEP: Machine Learning as a Service for HEP', Presentation at the Inter-experiment Machine Learning (IML) Working Group meeting, virtual event, 2021. [Online]. Available: https://indico.cern.ch/event/1015407/#21-mlaas4hep-machine-learning (cit. on p. 168).

[197] L. Giommi, 'MLaaS4HEP: Machine Learning as a Service for HEP', Presentation at the 5^{th} Inter-experiment Machine Learning (IML) workshop, CERN, 2022. [Online]. Available: https://indico.cern.ch/event/1078970/contributions/4833327/ (cit. on p. 168).

[198] L. Giommi, 'Machine Learning as a Service for HEP', Presentation at the CMS Machine Learning Town Hall workshop, virtual event, 2020. [Online]. Available: https://indico.cern.ch/event/922319/contributions/3928257/ (cit. on p. 168).

[199] L. Giommi, 'MLaaS4HEP: Machine Learning as a Service for HEP', Presentation at the CMS Machine Learning Town Hall workshop, virtual event, 2021. [Online]. Available: https://indico.cern.ch/event/1045606/contributions/4462511/ (cit. on p. 168).

[200] L. Giommi, 'Machine Learning as a Service for HEP', Workshop analisi dati at CMS Italia, Florence (Italy), 2022. [Online]. Available: https://indico.cern.ch/event/1107463/contributions/4728558/ (cit. on p. 168).

[201] *Use Torchserve for inference*. [Online]. Available: https://github.com/lgiommi/Torchserve_test (cit. on p. 169).

[202] *Kubeflow: the Machine Learning Toolkit for Kubernetes*. [Online]. Available: https://www.kubeflow.org (cit. on p. 169).

[203] *Predict activity of Rucio data placement*. [Online]. Available: https://github.com/lgiommi/MLaaS4HEP/blob/uproot4/doc/hdfs-example.md#predict-activity-of-rucio-data-placement (cit. on p. 169).

[204] *MLaaS4HEP worflow recipe*. [Online]. Available: https://github.com/lgiommi/MLaaS4HEP/blob/uproot4/doc/workflow_recipe.md (cit. on p. 174).

[205] *Uproot 3 → 4 cheat-sheet*. [Online]. Available: https://uproot.readthedocs.io/en/latest/uproot3-to-4.html (cit. on p. 177).

[206] *Docker image used with DODAS*. [Online]. Available: https://hub.docker.com/layers/139371886/felixfelicislp/mlaas_cloud/mlaas_jupyterhub/images/sha256-f9da08f00e3810de6494921c6ffa9148fea29eb028fd4432503e4cceadf111af?context=explore (cit. on p. 183).

[207] *MLaaS4HEP service demo*. [Online]. Available: https://www.youtube.com/watch?v=_JHg4oTeVbc (cit. on p. 194).