Edge AI Fundamentals

* Edge AI is Process of running artificial intelligence (AI) algorithms on devices at the edge of internet or other networks.

* The traditional approach to AI and machine learning (ML) is to use powerful, cloud based Servers to Perform model training as well as inference (Prediction serving).

* While edge devices might have limited resources compared to their cloud based cousins, they offer reduced bandwidth usage, lower latency & additional data privacy.

The network edge

* Edge Computing is a strategy where data is processed and stored at the periphery of a computer network.

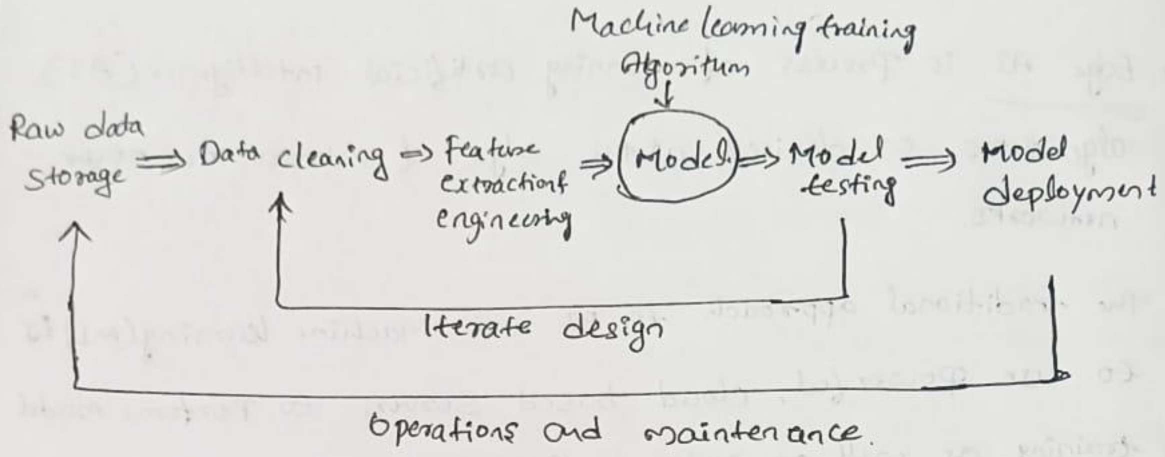
* In most cases, Processing and storing data on remote servers, especially internet servers, is known as "Cloud Computing"

* The edge includes all computing devices not part of cloud

Type of Hardware available to run over AI

- ✓ low power Microcontrollers
- ✓ Single Port Computers
- ✓ Graphic Processors
- ✓ powerful Servers
- ✓ AI excelerators

* Overview of Edge AI Life cycle


Machine learning training Algorithm
Raw data Storage
Data cleaning
Feature extraction engineering
Model
Model testing
Model deployment
Iterate design
Operations and maintenance.

* Edge Computing is a computer Networking Strategy where data is Processed & stored at the ~~property~~ periphery of the Network.

* The periphery includes end user devices & equipment that connect those devices to large networking infrastructure. Such as the internet. ex - Laptops, Smartphones, IOT devices, routers, & local switches count as edge computing devices.

* By processing data closer to where the data is generated, we can reduce Latency, limit bandwidth usage, improve reliability & increase data privacy.

* Network architecture overview

- Most networking architectures can be divided into "cloud" and "edge".

* Cloud computing consist of applications and services running on remote, internet connected devices.

* Edge computing is essentially everything that is not part of the cloud (i.e. in the internet)

* Benefits of Cloud Computing :-

- Large servers offer powerfull computing capabilities that can crunch numbers and run complex algorithms quickly
- Remote access to services from any device.
- Processing and storage can be scaled on demand
- physical servers are managed by large companies (google, microsoft, amazon) so that you do not need to handle the infrastructure & maintenance

* Edge Computing:

- The Network edge can be divided into Near edge & far edge

* Near edge equipment consist of on-premises or regional servers & routing equipment controlled by you or your business.
* Near refers to physical proximity or relatively low number of router hops it takes for traffic to go from the border of the internet to your equipment

* Far edge consist of the devices further away from the internet gateway on your network
ex- laptops. smartphones. IOT devices. routers, & switches.

* Benefits of Edge Computing.

- Reduced bandwidth usage.
- Reduced Network latency
- Improved energy efficiency
- Increased reliability
- Better data privacy

★ Dissadvantages :

* Limitations of edge Computing
  * Resource Constraints.
  * Limited Remote access
  * Security
  * Scaling

★ Examples :-

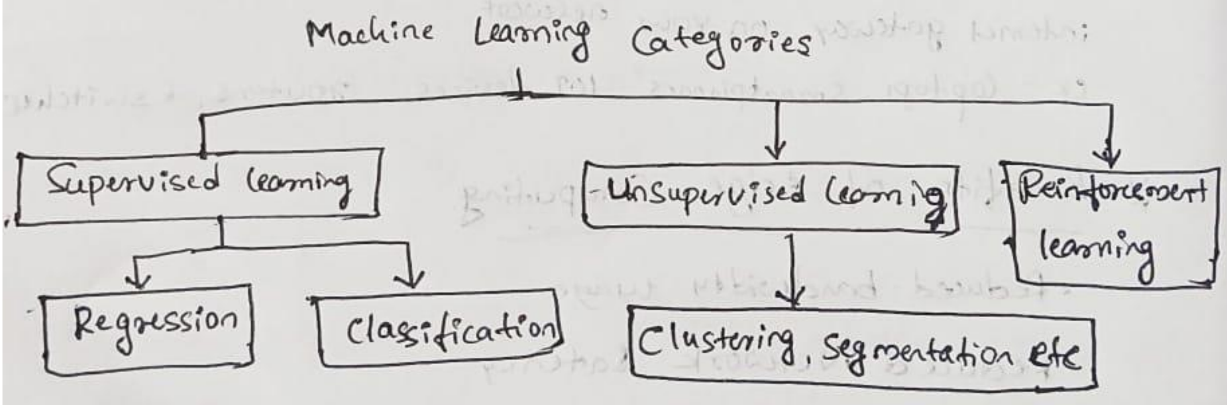
Edge Computing ex - Smart watches . Smart Speakers like Alexa PDE, Anything that runs locally on Your Computer

Cloud Computing ex - Netflix , google docs.

both - video Confrencing applications like zoom

★ AI is the development of algorithms & Systems to Simulate human intelligence. This can include automatically making decisions based on input data as well as Systems that can learn over time.

★ Machine learning is the development of algorithms & Systems that learn over time from data.



★ Other Categories - Semi supervised learning . and they often involve combinations of the main three categories.

* Supervised learning is concerned with finding a function (or mathematical model) that maps input data to some output. Such as predicted value, or classification.
* Supervised learning requires ground - truth tables to be present with data training. Such labels are usually set by humans.

* Regression - The model attempts to predict a continuous value.

* Classification is the process of predicting how well the input data belongs to one (or more) of several discrete classes.

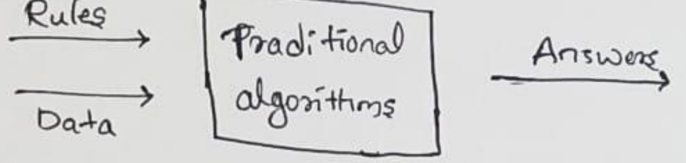
* Unsupervised learning is used to identify in data.

* Reinforcement learning focuses on models that learn a policy that selects actions based on provided input. Such models attempt to achieve goals through trial & error by interacting with environment.

~~* Other~~

* Traditional vs. machine learning algorithms.

* When developing traditional algorithms, the parameter and rules of the system are designed by a human. Such algorithms accept data as input & produce results.

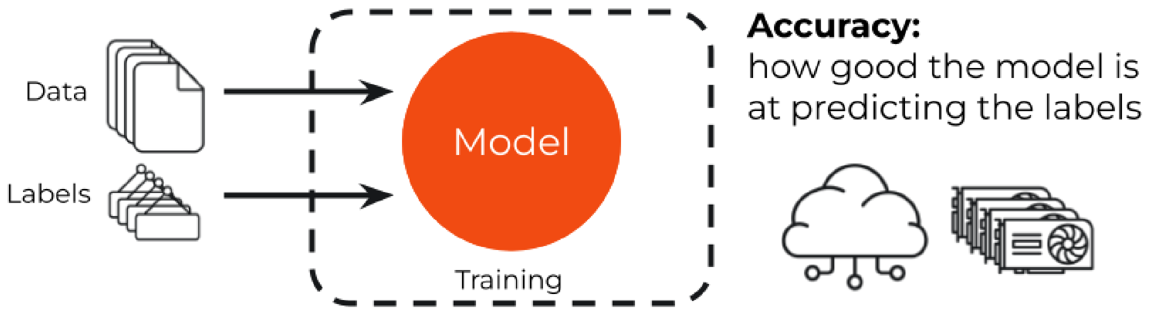


Examples :-

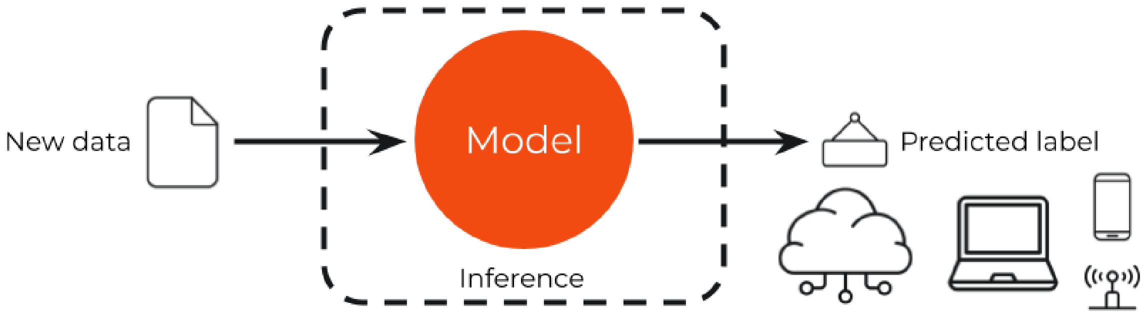
* Edge detection - filters are used to extract meaning from images
* Sorting algorithms are popular with search engines to present web search results.
* Fourier transform is used in signal processing to convert time series to freq
* Advanced Encryption Standard (AES) - to keep data secret during transcription

Machine learning training and deployment

In **machine learning (ML)**, data is fed into the training process. For supervised learning, the ground-truth labels are also provided along with each sample. The training algorithm automatically updates the parameters (also known as "weights") in the ML model.



During each step of the training process, we evaluate the model to see how good it is at predicting the correct label given some data. Over time, we ideally want this accuracy to increase to some acceptable level. In most cases, training a machine learning model is computationally expensive, and training does not need to be performed on an edge device. As a result, we can do model training in the cloud with the help of powerful accelerator hardware, such as graphics processing units (GPUs). Once we are happy with the performance of the model, we can deploy it to our end device. At this point, the model accepts new, never-before-seen data and produces an output. For supervised learning and classification, this output is a label that the model believes most accurately represents the input data. In regression, this output is a numerical value (or values). This process of making predictions based on new data after training is known as *inference*. In traditional, cloud-based ML model deployment, inference is run on a remote server. Clients connect to the inference service, supply new data along with their request, and the server responds with the result. This cloud-based inference process is known as *prediction serving*.



In the majority of cases, inference is not nearly as computationally intensive as training. As a result, we could run inference on an edge device instead of on a powerful cloud server.Because edge devices often offer less compute power than their cloud counterparts, ML models trained for the edge often need to be less complex. With that in mind, edge AI offers several benefits over cloud AI.

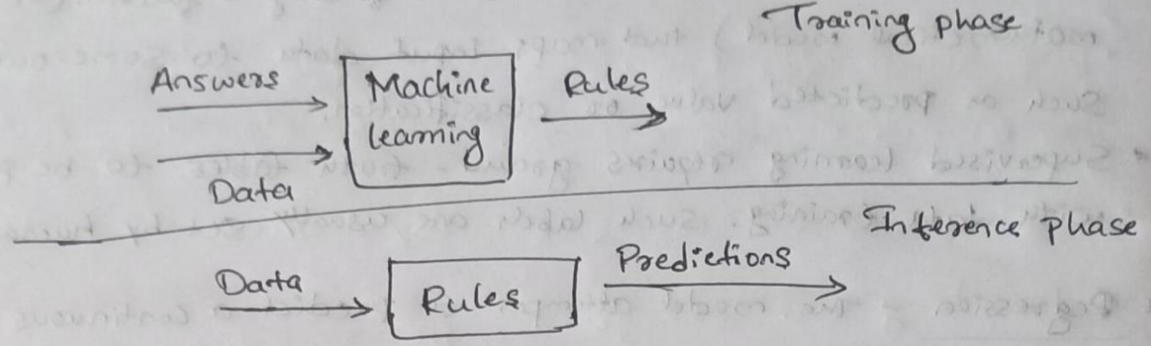


Fig: Machine Learning Rules Creation & Deployment.

* <u>Use Cases</u>

Edge AI works on data collected from sensors.

- <u>Time-Series Sensor data</u>. (ex- Sleep patterns, Mech equip. failu)
  (Slow sample rates (1Hz to 1000Hz)
- <u>Audio</u> - Identify wake words (ex- animals. machinery.)
  (higher sample rates 10KHz to 40KHz)
- <u>Image Classification</u> - Identify Image (ex- object. animal. Person)
  (Response time - 1fps to 60fps+)
- <u>Object detection</u> - Detect one or more targets

`Design constraints

Whether you are building an edge AI device for sale or buying off-the-shelf (OTS) components to solve a business need, you should consider your environmental and use constraints.

- **Interfaces** - Does your device connect to sensors? Will it need a connection to the internet (e.g. WiFi) or a smartphone (e.g. Bluetooth)? Does it need to have a user interface (screen, buttons, etc.), or can it be embedded in another device without human interaction?
- **Power constraints** - If the device is battery-powered, how long does it need to operate on a single charge? Even if the device can be plugged into the wall, optimizing for energy savings means you can save money on electricity usage.
- **Form factor** - Do you have the space for a large, powerful server? If not, can you mount a small box containing your device somewhere? Alternatively, is the device wearable, or does it need to conform to some unique shape?
- **Operating environment** - Most electronics work best in a climate-controlled environment, free from moisture and debris. Can you place your device in climate-controlled room like a server room or office? If not, does your device need to be hardened for a specific operating environment, like the outdoors, vehicle, or in space?
- **Code portability** - If you are designing an edge AI application, you should weigh your available options for code portability. Code optimized for a particular piece of hardware can often execute faster and with less energy usage. However, optimized code can often be difficult to port to different hardware and may require unique expertise and extra time to develop. Portable code, on the other hand, usually requires some overhead in the form of an operating system, but it is often easier to run on different hardware (i.e. port to a different device).

Off the shelf (OTS) versus do it yourself (DIY)

You have the option of buying any or all parts of an edge AI solution from a third-party provider. OTS usually involves a higher unit price, as vendors have overhead and profit margins built in. However, purchasing the device, software, or framework likely means faster setup and time-to-market. Additionally, some of the support/maintenance needs can be passed on to the vendor.If you are developing or selling an electrical device, OTS options often include compliance testing, such as **UL**, **FCC**, and **CE**. Such testing can be expensive and time-consuming, but they are almost always necessary for selling devices in a given country. On the other hand, developing the device or solution yourself requires more up-front time and costs in engineering, programming, and compliance testing. However, the device can be customized and optimized for particular use cases and environments. You also gain economies of scale if you plan to manufacture and sell hundreds or thousands of devices. The following chart summarizes the tradeoffs between OTS and DIY.

Buy (OTS)	Build (DIY)
Time efficiency	More engineering effort
Ease of use	Customization
Higher unit cost	Potential hidden costs
Third-party support	Independence from third-party vendors

Choosing hardware

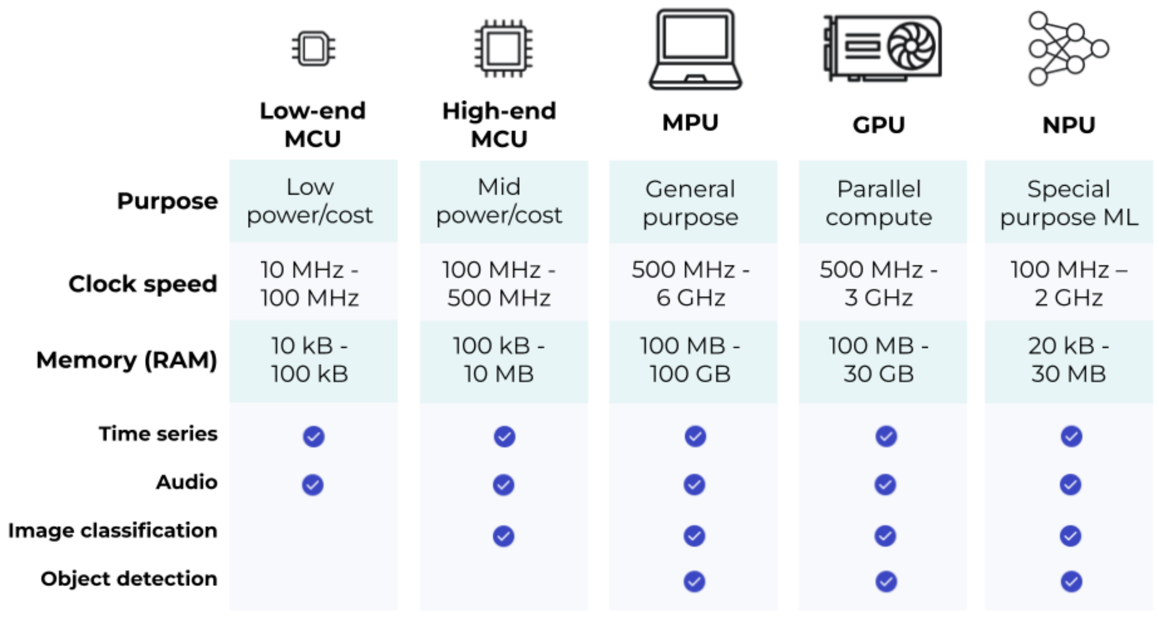
Once you have an idea of your problem scope and design constraints, you can choose the appropriate hardware. Most edge AI is performed by one of the following hardware categories:

- **Low-end microcontroller** - A microcontroller (also known as a microcontroller unit or MCU) is a self-contained central processing unit (CPU) and memory on a single chip, much like a tiny, low-power computer. Low-end microcontrollers are often optimized for a single or few tasks with a focus on collecting sensor data or communicating with other devices. Such MCUs usually have little or no user interface, as they are intended to be embedded in other equipment. Examples include controllers for microwave ovens, fitness trackers, TV remote controls, IoT sensors, modern thermostats, and smart lights.
- **High-end microcontroller** - High-end MCUs offer more powerful CPUs, more memory, and more peripherals (built-in WiFi, sensors, etc.) than their low-end counterparts. You can find high-end microcontrollers in vehicle engine control units (ECUs), infotainment systems in cars, industrial robotics, smart watches, networking equipment (e.g. routers), and medical imaging systems (e.g. MRI, X-ray).You can read more about microcontrollers **here**.
- **Microprocessor unit (MPU)** - An MPU is a CPU (often more than one CPU core) packaged on a single chip for general purpose computing. MPUs can be found in laptops, tablets, and smartphones. Unlike MCUs, they require external memory (e.g. RAM, hard drive) to function. They are almost always more powerful than MCUs and capable of crunching numbers at a faster rate. However, they also generally require more energy to function versus MCUs. You can read more about microprocessors **here**.
- **Graphics processing unit (GPU)** - Graphics processing units were originally designed to render complex 2D and 3D graphics to a computer screen. They are sold either as coprocessors on the same motherboard as an MPU (known as *integrated graphics*) or as a separate graphics card that can be plugged into a motherboard. In both cases, they

require another processor (usually an MPU) to handle the general computing needs. Because graphics are generally created using parallel matrix operations, GPUs have also seen success performing similar matrix operations for activities like cryptocurrency mining and machine learning. **NVIDIA** is the most popular GPU maker. You can read more about GPUs **here**.

- **Neural processing unit (NPU)** - NPUs are special-purpose AI accelerator chips designed to perform neural network calculations quickly and efficiently. Like GPUs, they almost always require a coprocessor in the form of an MCU or MPU to handle the general purpose computing needs. NPUs range from tiny coprocessors in the same chip as an MCU to powerful, card-based options that can be plugged into a motherboard. The Google **Tensor Processing Unit (TPU)** is one example of an NPU. You can read more about AI accelerators and NPUs **here**.

The boundary between low- and high-end microcontrollers is not clearly defined. However, we try to differentiate them here to demonstrate that your choice of hardware can affect your ability to execute different edge AI tasks.



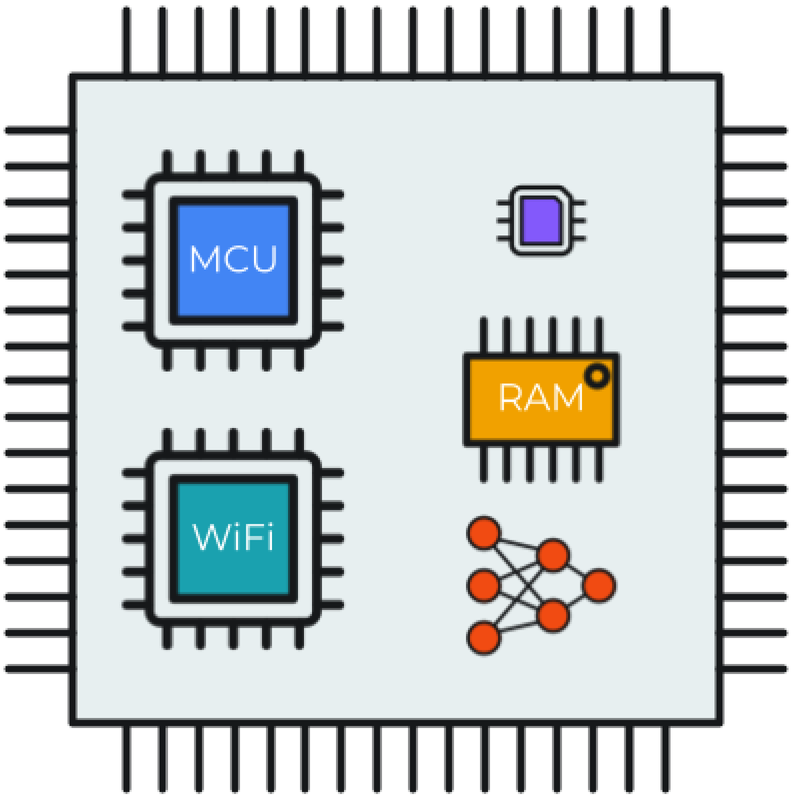
	Low-end MCU	High-end MCU	MPU	GPU	NPU
Purpose	Low power/cost	Mid power/cost	General purpose	Parallel compute	Special purpose ML
Clock speed	10 MHz - 100 MHz	100 MHz - 500 MHz	500 MHz - 6 GHz	500 MHz - 3 GHz	100 MHz – 2 GHz
Memory (RAM)	10 kB - 100 kB	100 kB - 10 MB	100 MB - 100 GB	100 MB - 30 GB	20 kB - 30 MB
Time series	✓	✓	✓	✓	✓
Audio	✓	✓	✓	✓	✓
Image classification		✓	✓	✓	✓
Object detection			✓	✓	✓

Which hardware works best for which edge AI task

The above chart makes general suggestions for which class of hardware is best suited for each edge AI task. Not all AI tasks are included, as some are better suited for cloud AI, and AI is an evolving field where such needs are constantly changing.

Hardware combinations

As noted, many of the processor types are not intended to operate alone. For example, GPUs are optimized for a particular type of operation (e.g. matrix math) and need to be paired with another processor (e.g. MPU) for general purpose computing needs. In some cases, you can create processor-specific modules, such as GPUs and NPUs on cards that easily slot into many personal computer (PC) motherboards.In some cases, you may come across single-chip solutions that contain multiple processors and various peripherals. For example, a chip might contain a high-end MCU for general processing, a specialized radio MCU for handling WiFi traffic, a low-end MCU for managing power systems, random-access memory (RAM), and a specialized NPU for tackling AI tasks. This type of chip is often marketed as a [system on a chip (SOC)](#).



Example of system on a chip (SOC)

Edge AI lifecycle

The edge AI lifecycle includes the steps involved in planning, implementing, and maintaining an edge AI project. It follows the same general flow as most engineering and programming undertakings with the added complexity of managing data and models.

Identify need and scope

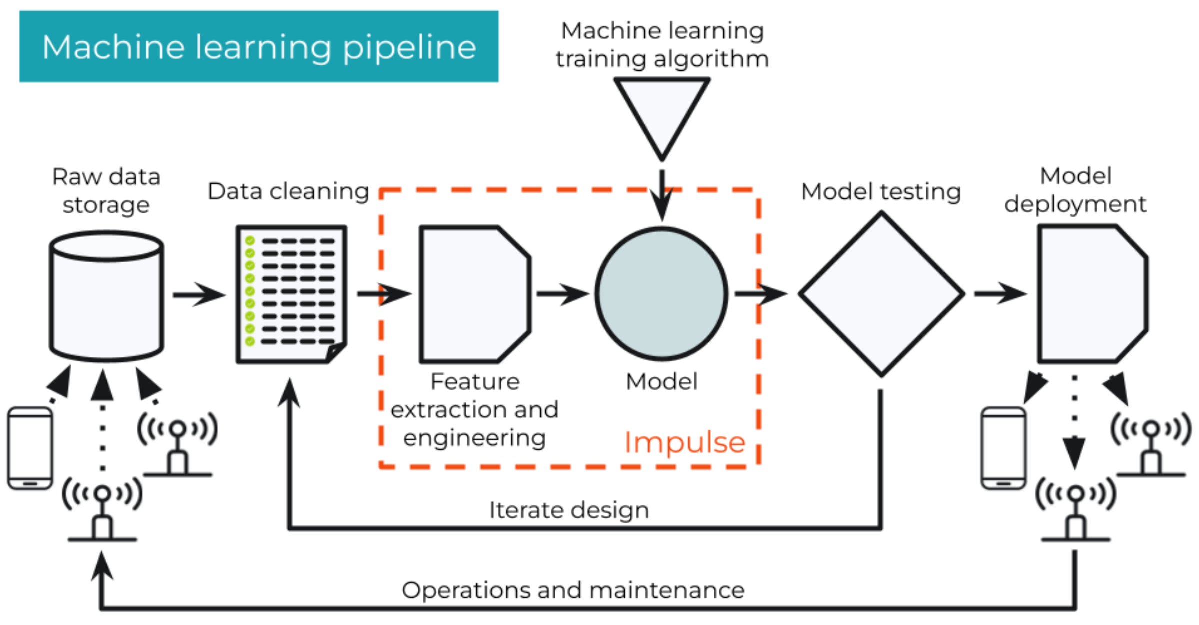
Before starting a machine learning project, it is imperative that you examine the actual need for such a project: what problem are you trying to solve? For example, you could improve user experience, such as creating a more accurate fall detection or voice-activated smart speaker. You might want to monitor machinery to identify anomalies before problems become unmanageable, which could save you time and money in the long run. Alternatively, you could count people in a retail store to identify peak times and shopping trends. Once you have identified your requirements, you can begin scoping your project:

- Can the project be solved through traditional, rules-based methods, or is AI needed to solve the problem?
- Is cloud AI or edge AI the better approach?
- What kind of hardware is the best fit for the problem?

Note that the hardware selection might not be apparent until you have constructed a prototype ML model, as that will determine the amount of processing power required. As a result, it can be helpful to quickly build a proof-of-concept and iterate on the design, including hardware selection, to arrive at a complete solution.

Machine learning pipeline

Most ML projects follow a similar flow when it comes to collecting data, examining that data, training an ML model, and deploying that model.


Machine learning pipeline
Machine learning training algorithm
Raw data storage
Data cleaning
Feature extraction and engineering
Model
Impulse
Model testing
Model deployment
Iterate design
Operations and maintenance

Machine learning pipeline and workflow

This complete process is known as a *machine learning pipeline*.

Data collection

To start the process, you need to collect raw data. For most deep learning models, you need a lot of data (think thousands or tens of thousands of samples). In many cases, data collection involves deploying sensors to the field or your target environment and let them collect raw data. You might collect audio data with a smartphone or vibration data using an IoT sensor. You can create custom software that automatically transmits the data to a **data lake**.

Data cleaning

Raw data often contains errors in the forms of omissions (some fields missing), corrupted samples, or duplicate entries. If you do not fix these errors, the machine learning training process will either not work or contain errors.

A common practice is to employ the **medallion architecture** for scrubbing data, which involves copying data, cleaning out an errors or filling missing fields, and storing the results into a different bucket. The buckets have different labels: bronze, silver, gold. As the data is successively cleaned and aggregated, it moves up from bronze to silver, then silver to gold. The gold bucket is ready for analysis or to be fed to a machine learning pipeline. The process of downloading, manipulating, and re-uploading the data back into a separate storage is known as *extract, transform, load* (ETL). A number of tools, such as **Edge Impulse transformation blocks** and **AWS Glue**, can be used to build automated ETL pipelines once you have an understanding of how the data is structured and what cleaning processes are required.

Data analysis

Once the data is cleaned, it can be analyzed by domain experts and data scientists to identify patterns and extract meaning. This is often a manual process that utilizes various algorithms (e.g. unsupervised ML) and tools (e.g. Python, R). Such patterns can be used to construct ML models that automatically generalize meaning from the raw input data. Additionally, data can contain any number of **biases** that can lead to a biased machine learning model. Analysing your data for biases can create a much more robust and fair model down the road.

Feature extraction

Sometimes, the raw data is not sufficient or might cause the ML model to be overly complex. As a result, manual features can be extracted from the raw data to be fed into the ML model. While feature engineering is a manual step, it can potentially save time and inference compute resources by not having to train a larger model. In other words, feature extraction can simplify the data going to a model to help make the model smaller and faster.

Train machine learning model

With the data cleaned and features extracted, you can select or construct an ML model architecture and train that model. In the training process, you attempt to generalize meaning in the input data such that the model's output matches expected values (even when presented with new data). Deep neural networks are the current popular approach to solving a variety of supervised and unsupervised ML tasks. ML scientists and engineers use a variety of tools, such as **TensorFlow** and **PyTorch** to build, train, and test deep neural networks. Pretrained models can be retrained using custom data in a process known as **transfer learning**. Transfer learning is often faster and requires less data than training from scratch. The combination of automated feature extraction and ML model is known as an *impulse*. This combination of steps can be deployed to cloud servers and edge devices. The impulse takes in raw data, performs any necessary feature extraction, and runs inference during prediction serving.

Model testing

In almost all cases, you want to test your model's performance. Good ML practices dictate keeping a part of your data separate from the training data (known as a *test set*, or *holdout set*). Once you have trained the model, you will use this test set to verify the model's functionality. If your model performs well on the training set but poorly on the test set, it might be **overfit**, which often requires you to rethink your dataset, feature extraction, and model architecture. The process of data cleaning, feature extraction, model training, and model testing is almost always iterative. You will often find yourself revisiting each stage in the pipeline to create an impulse that performs well for your particular task and within your hardware constraints.

Model deployment

Model deployment is about getting your trained AI model ready to be used to make predictions in the real world.

Cloud vs. Edge Deployment

- **Cloud-Based AI**: This is simpler. Tools like SageMaker help deploy your model onto a server that waits for data requests over the internet, processes them, and sends back the results.
- **Edge AI**: This is trickier because the model runs directly on a local device (like a phone or sensor). You must optimize the model to run fast and efficiently on that specific, often low-power, hardware.
    - Optimization: This involves processes like pruning (removing unnecessary parts of the model) and quantization (making the math simpler) so the model uses less memory and power.

The Need for an Application

A raw AI model is just math; it can't do anything on its own. It needs a surrounding application (software) to:

1. **Collect** the incoming data.
2. **Feed** that data to the model.
3. **Take action** based on the model's prediction (inference).

- **Cloud** apps typically just serve predictions over the web.

- **Edge** apps need a tighter link between the prediction and a physical action (e.g., stopping a machine, notifying a user).

Programmers and software engineers build this application and ensure the model works correctly on the final hardware.

Operations and maintenance (O&M)

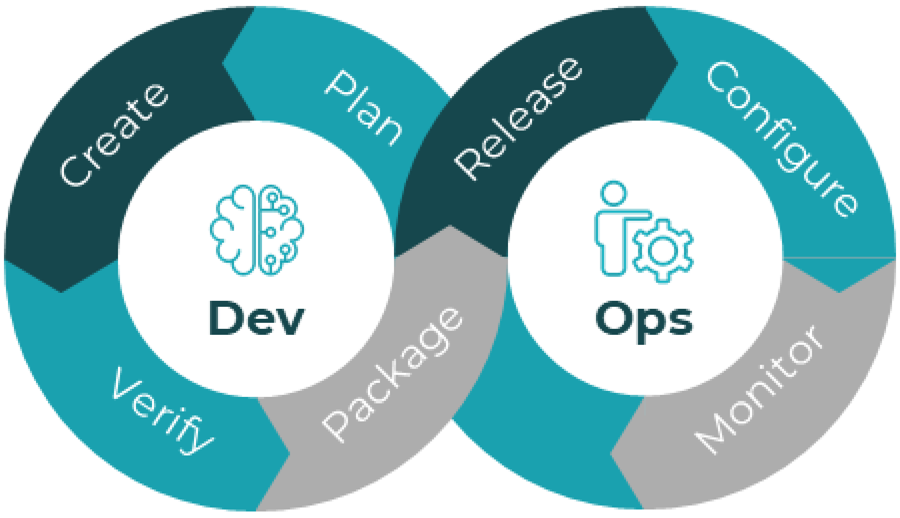
As with any software deployment, operations and maintenance is important to provide continuing support to the edge AI solution. As the data or operating environment changes over time, model performance can begin to degrade. As a result, such deployments often require monitoring model performance, collecting new data, and updating the model.

What is edge MLOps?

Edge machine learning operations (MLOps) is the set of practices and techniques used to automate and unify the various parts of machine learning (ML), system development (dev), and system operation (ops) for edge deployments. Such activities include data collection, processing, model training, deployment, application development, application/model monitoring, and maintenance. Edge MLOps follows many of the same principles of MLOps but with a focus on edge computing.

DevOps

DevOps is the collaboration between software development teams and IT operations to formalize and automate various parts of both cycles in order to deliver and maintain software.

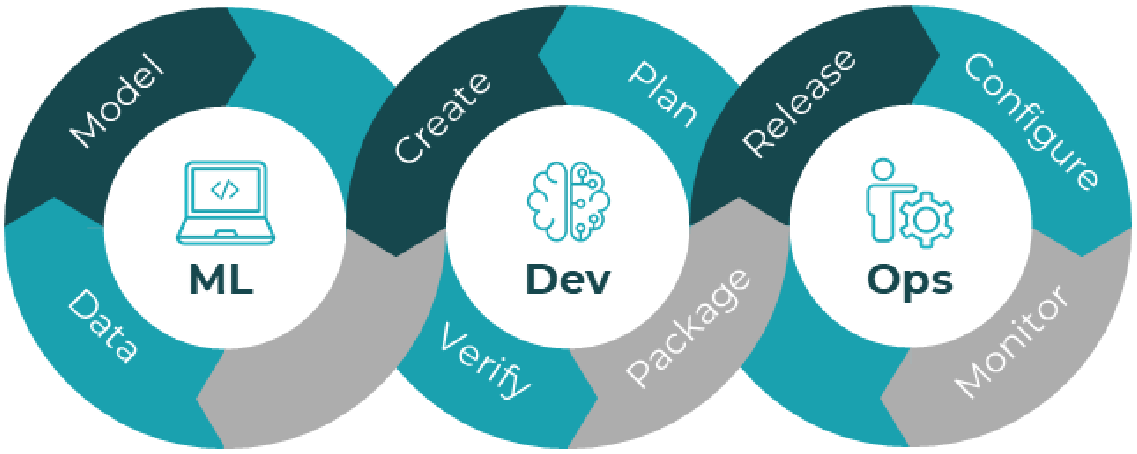


DevOps cycle

In this cycle, the software development team works with management and business teams to identify requirements, plan the project, create the required software, verify the code, and package the application for consumption. In many instances, this packaged software is simply "thrown over the fence" to the operations team to manage the release, which consists of pushing the software to users, configuring and installing the software for users, and monitoring the deployment for any issues.The concept of DevOps comes into play when these two teams work together to ensure smooth delivery and operation of the software. Many aspects of the packaging and delivery can be automated in a process known as **continuous integration and continuous delivery (CI/CD)**.

MLOps

Machine learning operations extends the DevOps cycle by adding the design and development of ML models into the mix.



MLOps cycle

Data collection, model creation, training, and testing is added to the flow. The machine learning team must work closely with the software development and operations teams to ensure that the model meets the needs of the customer and can operate within the parameters of the application, hardware, and environment. For cloud-based deployments, the application may be a simple prediction serving web interface, or the model may be fully integrated into the application.

Building frameworks for inter-team operation and lifecycle automation offers a number of benefits:

- Shorter development cycles and time to market
- Increased reliability, performance, scalability, and security
- Standardized and automated model development/deployment frees up time for developers to tackle new problems
- Streamlined operations and maintenance (O&M) for efficient model deployment

Team effort

In most cases, implementing an edge MLOPs framework is not the work of a single person. It involves the cooperation of several teams. These teams can include some of the following experts:

- **Data scientists** - analyze raw data to find patterns and trends, create algorithms and data models to predict outcomes (which can include machine learning)
- **Data engineers** - build systems to collect, manage, and transform raw data into useful information for data scientists, ML researchers/engineers, and business analysts
- **ML researchers** - similar to data scientists, they work with data and build mathematical models to meet various business or academic needs
- **ML engineers** - build systems to train, test, and deploy ML models in a repeatable and robust manner
- **Software developers** - create computer applications and underlying systems to perform specific tasks for users
- **Operations specialists** - oversee the daily operation of network equipment and software maintenance
- **Business analysts** - form business insights and market opportunities by analyzing data

Principles

Edge MLOps is built on three main principles: version control, automation, and governance.

Version control

In software development, the ability to track code versions and roll back versions is incredibly important. It goes beyond simply "saving a copy," as it allows you to create branches to try new features and merge code from other developers. Tools like git and GitHub offer fantastic version control capabilities.

Automation

Automation means setting up processes and software to automatically handle repetitive steps in the AI life cycle, like data collection, model training, and deployment.

- **Payoff:** It costs time/money up front but saves work later, letting your team focus on other tasks.
- **Key Stages (The "Continuous" steps):**
    - **Continuous Collection:** Automatically gathering data.
    - **Continuous Training:** Automatically training/testing models.
    - **Continuous Integration (CI):** Automatically testing code changes.
    - **Continuous Delivery (CD):** Automatically releasing new software versions.
    - **Continuous Monitoring:** Automatically checking performance and security.

- **Triggers:** These automated steps can be started by things like a set Time, new Data changes, changes to the Code, a user Request, or a model issue (Model monitoring like drift).

Governance

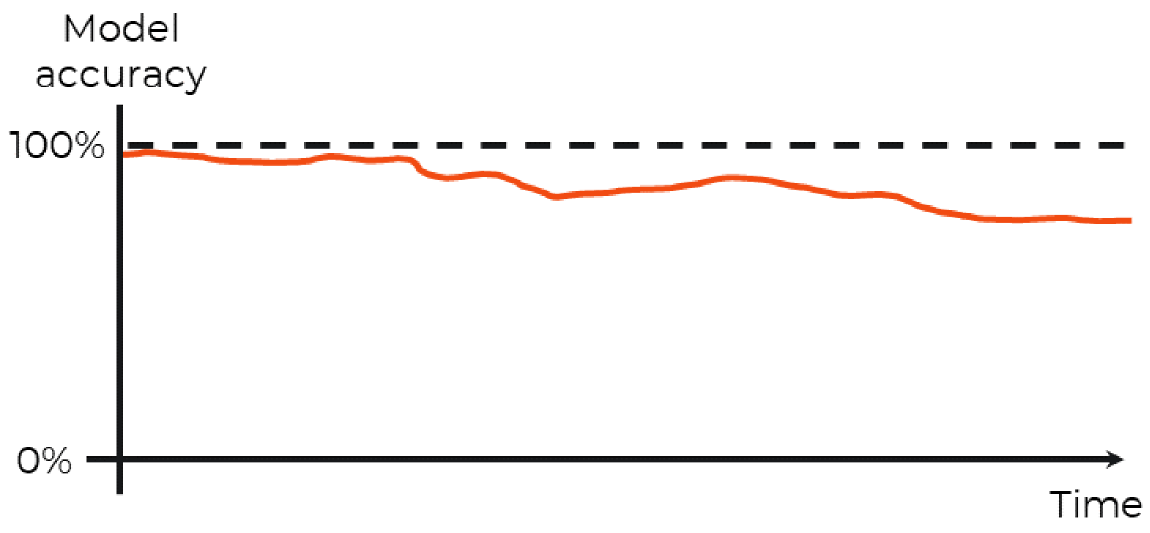
Governance is about ensuring your AI systems (data and processes) are safe, fair, and legal.

- **Compliance:** You must follow laws on data privacy (like GDPR or HIPAA) and emerging AI regulations (like the EU AI Act).
- **Fairness:** Check your data and model for bias. Remember: "Garbage in, garbage out"—biased data creates a biased model.
- **Security:** Implement best practices to ensure:
    - **Confidentiality:** Protect sensitive data from unauthorized access.
    - **Integrity:** Ensure data hasn't been changed.
    - **Availability:** Make sure authorized users can access the data when needed.

As an AI developer, it's your responsibility to ensure the systems are compliant, unbiased, and secure.

Model drift

Model drift occurs when an ML model's loses accuracy over time. This can happen over the course of days or years.



Machine learning model drift

In reality, the model does not lose accuracy. Instead, the data being fed to the model or the relationships that data represents in the physical world change over time. Such drift can be placed into two categories:

- **Data drift** occurs when the incoming data skews from the original training/test datasets.
- **Concept drift** happens when the relationship between the input data and target changes.

One way to combat model drift is to consistently monitor the model's performance over a period of time. If the accuracy dips below a threshold or users notice a decline in performance, then you may be experiencing such drift. At this point, you would need to collect new data (either from scratch or supplement your existing dataset), retrain the model, and redeploy.