**Module 3**

**Modular and Structured Software Development for Robust Intelligent Systems that run on HPC CI**

**Module Overview**

**Segment 1: Introduction**

**1. What is A.I.?**

**2. HPC and A.I.**

**Segment 2: Things to consider before building an AI**

**Segment 3: AI development workflow**

**Segment 4: Data integrity**

**Segment 5: Implementation (MNIST)**

1. **Introduction**

*1.1 What is Artificial Intelligence (AI)?*

There is no such thing as a universal or unified definition of AI. It is hard to define artificial intelligence because we first need to define *intelligence*. Moreover, if you ask people to define intelligence, the definitions you get vary from person to person and from context to context. The definition of intelligence is subjective rather than objective. However, for our purpose, we would follow the rational agent approach, given by *Stuart Russell* and *Peter Norvig* in their book, *Artificial Intelligence: A Modern Approach*. As defined in the book, “a **rational agent** is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.” What is the best outcome? Well, it depends on the objective provided to the agent. In a nutshell, AI is the study of rational agents.

So, what are the applications of artificial intelligence? The smart voice assistants that we have in our cell phones, like Alexa, Siri, Google Assistant, etc. When you ask your voice assistant to call someone or play music, it recognizes your voice and then follows your order to its best capabilities. In this example, the voice assistant acts as a rational agent. Other applications include assembly line robotics, self-driving cars, automated financial investing, fraud prevention systems, google search, health monitoring systems, and many more.

*1.2 High Performance Computing and AI:*

Now let us shift our focus to high-performance computing and its importance in the field of artificial intelligence. In general terms, **high-performance computing (HPC)** can be defined as the practice of combining multiple individual computers (nodes) into a group called cluster in such a way that collectively they can deliver much higher performance than a typical computer to solve large, computationally expensive problems.

So far, we have defined AI and HPC. Now let us examine the ways HPC fits into the field of AI. In the current landscape, most AI applications deal with large, complex datasets. For example, autonomous driving vehicles require training on thousands or millions of data samples before they can be used.

Yes, and you read it right. While some rational agents use algorithms without training, such as finding the shortest path from point A to point B, most rational agents are first trained with extensive sample data and expected outcomes. Then the rational agent tries to find patterns in the given data to reach the expected outcome with a higher accuracy percentage and thus ‘learns’ from the data. This is called **Machine Learning (ML)**, a subset of AI. While training a rational agent, if the input dataset that’s being used is small, a typical computer can handle the load. However, when it comes to the dataset, which has millions to billions of data points or samples, such as in the field of computer vision, it requires much higher computational power than a typical computer can deliver. Even after that, it could end up requiring days to even months to train a rational agent. Because of HPC, nowadays, we can process a large volume of data efficiently, enabling us to train complex rational agents faster and with higher accuracy, such as autonomous systems, which would have been intractable if not for the concept of HPC.

Hence, HPC plays a vital role in the advancement of AI. Especially in the modern world, where billions of data points are generated each day as technology is becoming an integral part of human life.

*1.3 What is HPC CI?*

High-performance computing cyber infrastructure*,* or HPC CI, is reference to the components necessary to build and apply HPC systems. HPC computing finds roots in the early-to-mid 20th century, with the term *supercomputer* initially emerging in response to work IBM was doing in the 1920's. However, it wasn't until the 1960's that we saw what many consider the first real supercomputer: CDC 6600. Developed by Seymour Cray for Control Data Corporation, the CDC 6600 was 3x faster than the IBM 7030 Stretch, the previous fastest computer [4]. Among the many techniques and strategies used by Cray and his team to build the CDC 6600, one of the most influential today is the concept of *parallelism*.



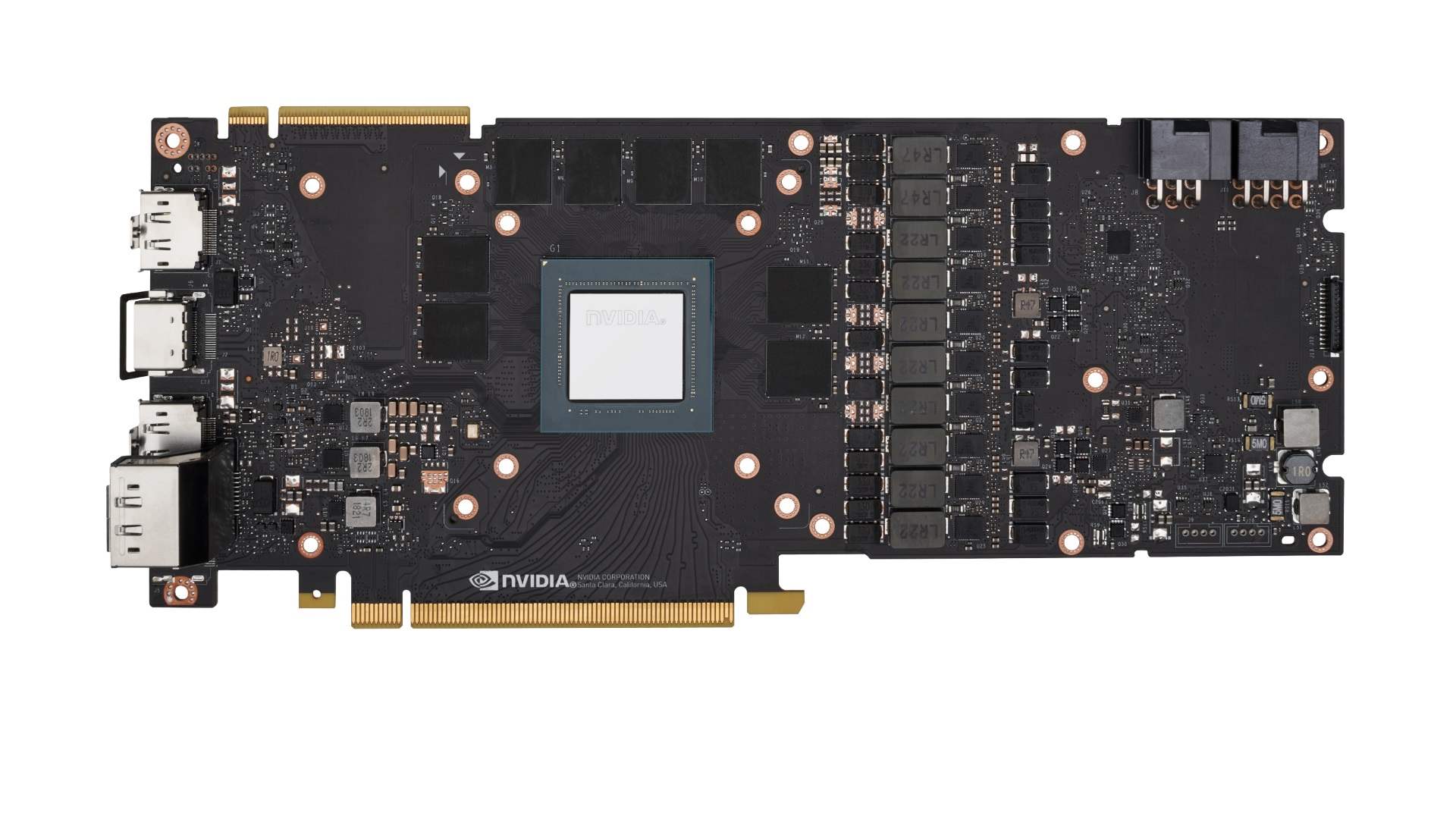
The CDC 6600

[Image Source: <https://www.computerhistory.org/timeline/1964/>]

Parallel computing is a strategy for computation that involves carrying out multiple calculations or processes simultaneously, either across multiple nodes of a distributed system, or across threads of a single processing unit, also known as a *core*. By decomposing tasks in to smaller, sub-divided tasks that can be spread across multiple different components of a system, allows for the new sub-tasks to be run concurrently, thus significantly increasing processing speed [3]. *Multi-core processors* are computer processors on a single integrated circuit with two or more separate cores, each of which reads and executes program instructions [5].

*1.4 What are GPUs?*

GPUs (Graphics Processing Units) are one of the most important hardware components of modern computer architecture. The GPU can be defined as "a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device." In less technical language, it's the part of your computer tasked with rendering the visual components of the user's computing experience [1]. The technology's earliest roots can be traced back to the MIT-built Navy flight simulator 'Whirlwind', a project famous for producing one of the first 3D graphics systems. The modern GPU's foundation wasn't laid until the mid 70's with the advent of video shifters and video address generators for graphics processing in arcade games. In the early 2000's, GPUs with many parallel cores found a broader market, expanding into the realm of general purpose and scientific computing as the demand grew for devices that could perform numerous complex simultaneous calculations [2].



Nvidia' Turing' GPU

Image Source: <https://www.pcgamesn.com/nvidia-turing-gpu-architecture-specs>

Today, GPU computing is an essential part of HPC systems, with three of the ten most powerful supercomputers in the world (as of 2019) using GPU acceleration to improve performance [1]. The GPU’s highly parallel structure makes them far more efficient than CPUs for running algorithms that process large amounts of data. As the field of artificial intelligence continues to evolve into a new paradigm characterized by the processing of massive datasets with complex statistical learning algorithms, it has become increasingly apparent that standard CPU or even single-GPU solutions are no longer viable to meet the computational needs of AI professionals. This is where something called 'clustering' comes into play.

A computer *cluster* is an integrated set of computers that behave, functionally, as a single system. Clusters are used to significantly improve computing performance. The origin of computer clustering can be traced back to the middle of the 20th century, corresponding closely with the development of networks that allowed for the linking of compute resources. GPU-based clusters are a newer type of computer cluster that equips each node of the cluster with a GPU.

*1.5 The Cloud & High-Performance Computing*

One of the most powerful tools in tech right now is cloud computing. Cloud computing is the delivery of a variety of services and resources via the internet. Some of these resources include data storage, networking, databases, and servers. Cloud computing gives users the ability to access an array of tools that allow for increased speed, efficiency, and cost savings. Most of the major tech companies offer cloud services in some form, with examples including Google Cloud, AWS, Microsoft Azure, and Alibaba Cloud [6]. Traditionally, HPC was limited to the traditional 'supercomputer' that requires local hardware with many processors running multiple tasks in parallel. Today, cloud computing allows for the outsourcing of storage, hardware, application software, etc., on an on-demand basis, accessible to anyone with an internet connection and the technical ability to leverage the tools.

References

[1] "GPU Computing." *WVU*, WVU, 2021, <https://docs.hpc.wvu.edu/text/43.UsingGPUs.html>.

[2] House, Bryan. "A Brief History of GPUs." *Medium*, Deep Sparse, 9 July 2019, <https://medium.com/neuralmagic/a-brief-history-of-gpus-27122d8fd45>.

[3] *What is parallel computing? definition and faqs*. OmniSci. (n.d.). Retrieved October 29, 2021, from <https://www.omnisci.com/technical-glossary/parallel-computing>.

[4] *Computer History Museum*. CDC 6600's Five Year Reign. (n.d.). Retrieved October 29, 2021, from <https://www.computerhistory.org/revolution/supercomputers/10/33>.

[5] *Multi-core processor*. HandWiki. (n.d.). Retrieved October 29, 2021, from <https://handwiki.org/wiki/Multi-core_processor>.

[6] Frankenfield, J. (2021, May 19). *How cloud computing works*. Investopedia. Retrieved October 29, 2021, from <https://www.investopedia.com/terms/c/cloud-computing.asp>.

*1.6 Robust AI:*

In the preceding sections, we defined AI and provided a couple of applications of AI. When developed and used correctly, AI is a powerful technology that makes human life better. However, if AI fails, its consequences can be devastating. For example, an autonomous driving vehicle failure can cause a road accident which can result in loss of life. (If you want to see more examples of AI failures, you can look them up in AI Incident Database here [https://incidentdatabase.ai](https://incidentdatabase.ai/).) Therefore, we need to ensure that current and future scientists who advance AI and practitioners who use AI understand the tech’s limitations and how to develop systems that are robust and dependable.

So, what does robustness mean? Within the context of this module, robustness means a program/software’s ability to behave and respond appropriately for all the defined/expected use cases, even with versions that have been scaled up or down, while also gracefully responding to the unexpected/undefined use cases for a given problem.

Now, a new question arises, how can we achieve robustness? A relatively introductory and straightforward answer would be to follow sound software engineering principles. Moreover, one of the approaches which can be used to build robust software would be to apply the principle of modular programming.

*1.7 What is modular programming, and how does one achieve it?*

According to the book, *Programming Fundamentals: A Modular Structured Approach* by *Kenneth* and *Dave*, m**odular programming** is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each module contains everything necessary to execute only one aspect of the desired functionality. Now, since we know what modular programming is, let us jump into how to achieve it. As a widely accepted rule of thumb, we can follow the guidelines below to write a modular program.

· A program should be divided into small, less complex sub-programs called modules.

· Each module should perform only one task, and the engineer should document its purpose, expected input, and output.

· Each module should be callable or useable without needing to understand any of its implementation details.

· As much as possible, each module should be kept independent of other modules.

· Each module should be reusable enough to be reused multiple times without modifications or customization to its body.

· Each module should hold the capability of being tested alone outside of the main program,

· Each module should handle any error that arises within, without causing other modules to be affected or the whole program to crash.

Now that we have a general idea of what modular and robust programming means, let's follow up on the development strategies used by some of the well-known names in the industry and academia such as Google, IBM, MIT, CMU, Microsoft, Stanford, and many other institutions for their AI development processes.

1. **Few things to consider before building an AI**

Even before we start our work building an AI system, we should first consider the tips provided by major players in AI development. A few of the following recommendations come from Google Research:[1]

1. Always go with a human-centered approach.
2. Try to set up multiple metrics for assessing the design, monitoring the training process and analyzing the results.
3. Try to go through your raw data and make sure that you can interpret and understand it before an unseen basis skews the model's results.
4. Perform rigorous unit testing throughout the isolated unit systems. Try to apply the principle of mistake-proofing ([poka-yoke](https://www.villanovau.com/resources/six-sigma/what-is-poka-yoke/)).
5. Continue to monitor the model throughout its lifecycle and always look back on how the updated model differs from previous iterations.
6. **AI Software Workflow and Best Practices**

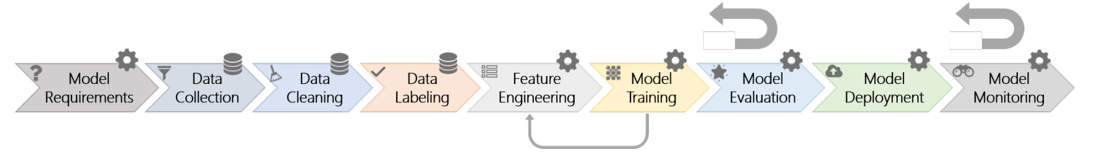


Figure 1: Nine-stage workflow for an ML model. The feedback loop denotes the necessity of moving back to an earlier stage [2]

Compared to traditional software engineering principles, AI/ML introduces a few new points to consider, such as [2]:

* Develop pipelines that seamlessly allow developers to change hyper-parameters for updating a dependency model so that the newer software can have more robust integration
* Promote data reusability within a corporation to reduce the duplication effort on collection and training
* Practice tagging the dataset with original information and which version of code is used for its extraction
* Consider frequent training and information sharing among the team members on what work
* Prevent using the black-box technique and try to add by adding more visualization techniques
* Consider applying modularization in a conventional, layered, and tiered software architecture to simplify error analysis and debuggability
* Develop a set of principles on fairness, accountability, transparency, and ethics that leads to a more robust and safer AI for everyone

*Optional Learner Exercise:* Develop an example that follows through all seven of the points above. A case study examining the process of building an SSR AI that gives concrete examples of actions taken to meet the best practice requirements.

**4. Data Integrity**

Data integrity is an essential part of developing safe, secure, and reliable artificial intelligence. A.I., after all, is only as good as the data used to train it. So, what exactly is data integrity? It's a broad concept with many distinct aspects, but, essentially, it is the maintenance of, and the assurance of, data accuracy and consistency over its entire life cycle. Three primary attributes of good data are completeness, accuracy and validity.

*Data Accuracy*

Accurate data, obviously, is data that corresponds to reality. Data is a representation of some real-world phenomenon and, if the data is to be useful, it must exist in such a way that it represents the real-world phenomenon true to form. For example, if some database is created to record employee salaries, the salaries attributed to each employee must be what the employee is actually paid. This may seem obvious, and, perhaps it is, but it's crucial to have systems in place to verify that the data being recorded is correct. Routine audits and rigorous review processes should be put in place to reduce the probability of inaccurate data being used to train A.I. systems.

*Data Completeness*

Accurate data, alone, is not sufficient for representational faithfulness, the information must be complete in both space and time. So, what does it mean for data to be complete? Data completeness is a simple concept that can be best illustrated by imagining our database of employee names and salaries. If our data table is missing the salaries of two employees, or perhaps a new employee had been hired a few weeks after the table was created so they have yet to be added, then our data is said to be incomplete. Missing data renders data incomplete, so data completeness is simply a reference to how much of necessary data is included in the database.

Achieving 100% real-time completeness isn't possible because of fundamental physical limitations to information processing systems, especially if the system or object being represented by data is highly dynamic. For example, if a database tracks every time a customer makes a purchase in some store, no matter what technique is being used to track this metric, there will exist some delay between when the actual act of purchasing occurs and when it is recorded in the database. Whether a person is hired to watch a security camera recording at the end of each day with the purpose of manually recording every purchase made, or if some system automatically records each purchase, there is some degree of latency that emerges. This is unavoidable but crucial to be aware of when designing AI systems.

*Data Validity*

Representational faithfulness of information requires more than just accurate correspondence with a physical condition. Valid data is data that is an appropriate and adequate device for understanding some phenomenon. You wouldn't train an NLP bot with image data. NLP requires natural language data. This is an obvious example, but it's not always so clear what type of data is best suited for the task at hand. In machine learning, choosing the right data for a model is called *feature selection.*Feature selection is a topic one could write books about, but here's a [link](https://towardsdatascience.com/why-data-integrity-is-key-to-ml-monitoring-3843edd75cf5)that offers a good starting point for beginning to understand how to select the best data for machine learning models.

 5**. Building a Robust AI design**

Now that we’ve studied a bit of the theory and learned a few new concepts, let's get some practical experience building a robust AI. Today, we’ll get started by constructing an intelligent system that can recognize one’s handwriting.

It’s highly recommended that you follow along with the mnist.pynb file that accompanies this module because it contains de-bugged, executable cells that are ready to run, but you’re welcome to read through this section to familiarize yourself with the lab as well.

As an introduction to AI programming, we have prepared a few basic functions that you can use to implement your very own Neural Network. You can import them via:

from run import \*

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.optimizers import Adadelta

from tensorflow.keras import utils

import numpy as np

First, to make a robust AI we need a dataset with a sufficiently large number of observations. Luckily, there is a package in Tensorflow that provides a large dataset of images that are ready to be processed.

(x\_train, y\_train), (x\_test, y\_test) = get\_dataset(num\_classes)

Here we are pulling a dataset from the internet and the function takes care of processing the data.

Now we need to create a Neural Network. Keras, a high-level API that acts as an interface for the TensorFlow library, makes building neural networks relatively easy.

model = get\_model(num\_classes)

We now need to compile our model with an optimizer, you can learn more about optimizers [here](https://keras.io/api/optimizers/). but for now, just go with the Adadelta optimizer:

optimizer = Adadelta(lr=learning\_rate)

model.compile(optimizer=optimizer,

            loss='categorical\_crossentropy',

            metrics=['accuracy'])

You can now train and save your model with:

model.fit(x\_train, y\_train,

            batch\_size=batch\_size,

            epochs=epochs,

            verbose=verbose,

            validation\_data=(x\_test, y\_test))

    model.save("test.h5")

loss, accuracy = model.evaluate(x\_test, y\_test, batch\_size=128)

print("loss:", loss)

print("accuracy:", accuracy)

Here, Keras' fit function is optimized to such a degree that it will always use the highest number of available cores. This would be especially useful to us if we were running this on an HPC system.

Now, let's take a look at a few hyperparameters. These values can be adjusted to the performance of your model.

batch\_size = 128

epochs = 5

num\_classes = 10

verbose = 1

img\_rows, img\_cols = 28, 28

learning\_rate=0.1

A single *epoch* represents one full cycle through the training data and the *learning rate* dictates how much to change the model in response to the estimated error each time the model weights are updated. Num\_classes represents how many different possible outputs there are. In this case, the 10 classes are images of digits 0 through 9. You can test with the hyperparameter settings above to see how your model performs and then change them to see if you can get better results. Give it a try. Note how the results vary depending on how many epochs you choose or what learning rate you choose.

Now that your model is ready let's make some preparation for predictions.

from PIL import Image

model = load\_model("test.h5")

img = Image.open('path to image').convert("L")

img = np.resize(img, (img\_rows, img\_cols))

im2arr = np.array(img)

im2arr = im2arr.reshape(1,img\_rows, img\_cols,1)

im2arr = im2arr / 255

yh = np.argmax(model.predict(im2arr.reshape(1,img\_rows, img\_cols,1)), axis=-1)

print(yh)

Here, we load the model, resize it to have the same dimension as during the training, and use the model.predict function to let the computer decide what the value is.

The final remaining task is to make 28x28 pixels black filled canvas. I recommend a brush of 4 pixels to draw your number. And finally, you can test your model to see if it can recognize the digit you painted.

Congratulations!! You have just made your very own Neural Network that can predict your handwritten digits.

The code design was inspired by the databricks design[7].

1. [**https://cloud.google.com/responsible-ai#section-2**](https://cloud.google.com/responsible-ai#section-2)
2. **https://www.microsoft.com/en-us/research/uploads/prod/2019/03/amershi-icse-2019\_Software\_Engineering\_for\_Machine\_Learning.pdf**
3. [**https://arxiv.org/pdf/1808.01664.pdf**](https://arxiv.org/pdf/1808.01664.pdf)
4. [**http://art360.mybluemix.net/?\_ga=2.265458054.825521404.1626898609-440520872.1626560578**](http://art360.mybluemix.net/?_ga=2.265458054.825521404.1626898609-440520872.1626560578)
5. [**https://medium.com/dataswati-garage/create-a-robust-ai-rest-api-71a8050ce314**](https://medium.com/dataswati-garage/create-a-robust-ai-rest-api-71a8050ce314)
6. **http://www.r-5.org/files/books/computers/algo-list/common/Heineman\_Pollice\_Selkow-Algorithms\_in\_a\_Nutshell-EN.pdf**
7. https://docs.databricks.com/\_static/notebooks/deep-learning/mnist-tensorflow-keras.html