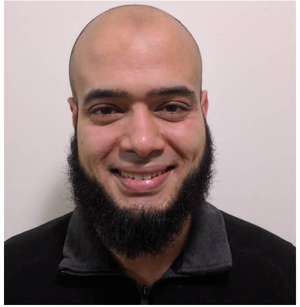
Machine Learning
Model Deployment

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Building Machine Learning Systems

- D. Sculley, et al, Hidden [Technical Debt] in Machine Learning Systems
  - *"ML systems have a special capacity for incurring **technical debt**, because they have all of the maintenance problems of traditional code **plus** an additional set of ML-specific issues"*

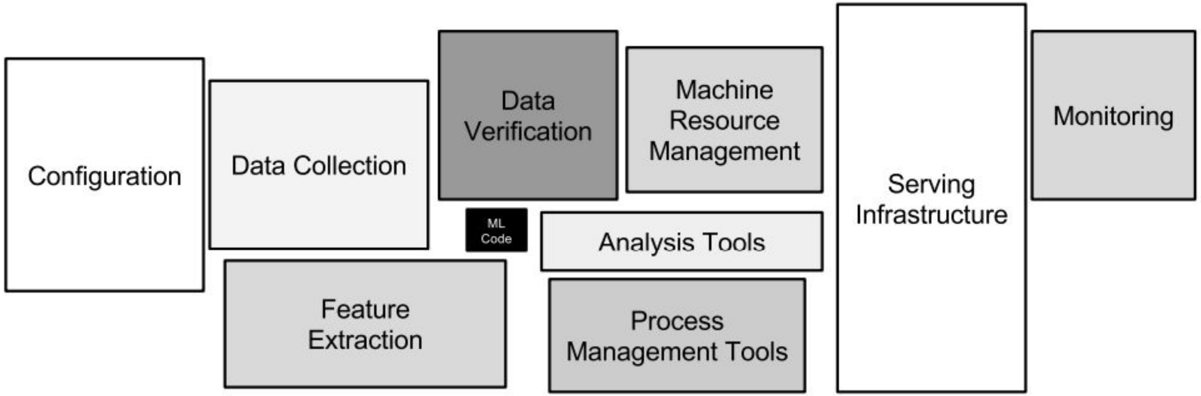


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Machine Learning Operations (MLOps)

- MLOps (Machine Learning Operations) are **processes and practices** that **automate** the end-to-end (e2e) ML lifecycle and **ensure efficient and effective** model development, deployment, and maintenance.
  - Continuous **Integration** and Continuous Deployment (CI/CD)
    - Automate the building, testing, and deployment
    - Continuous Model **Retraining**
  - Model **Versioning** for data, code, parameters, and configurations
  - **Monitoring** and Logging: performance, data quality, and system health
  - **Validation** and Testing: for both models and the data pipelines
  - Compliance and Security

MLOps: Job postings

Responsibilities

- You will work closely with our data scientists, software engineers, and DevOps team to ensure that our MLOps services are reliable, scalable, and performant
- Design, build, and maintain solutions that support our MLOps services on cloud (AWS, Azure, GCP)
- Collaborate with the data science, engineering, and DevOps teams to ensure that our MLOps services are integrated with the infrastructure
- Automate the deployment, scaling, and monitoring of MLOps services using Kubernetes and related technologies such as Helm, Istio, and Prometheus
- Implement security and compliance controls MLOps services
- Optimize the performance and reliability of MLOps services through monitoring, logging, and performance testing
- Develop and maintain documentation, training materials, and best practices for the solution
- Collaborate with other teams and stakeholders to ensure that MLOps services meet their requirements and expectations
- To be considered for this position you will be required to complete a technical assessment as part of the selection process

Responsibilities

- As a MLOps Engineer, your expertise will be crucial in designing and executing our machine learning operations strategy
- You will work closely with data scientists, software engineers, and Data Engineering teams to create and enhance complete machine learning pipelines and frameworks
- Create scalable MLOps frameworks and infrastructure to support the full machine learning lifecycle, from data ingestion to model deployment and monitoring
- Work closely with data scientists and software engineers to seamlessly integrate machine learning models into production systems, prioritizing robustness, scalability, and performance
- Implement automation and CI/CD practices to streamline model deployment, version control, and testing, ensuring efficient and reliable updates and rollbacks
- Develop and maintain monitoring and alerting systems to track model performance, data drift, and system health, enabling proactive issue detection and resolution
- Optimize resource utilization and cost-efficiency by establishing scalable and efficient infrastructure for training and inference, leveraging cloud platforms
- Establish and enforce best practices for data management, ensuring data quality, security, and privacy throughout the machine learning pipeline

From DevOps to MLOps

- MLOps has bigger scope than DevOps
- DevOps Engineers ⇒ In addition Data Engineers, ML Engineers
- Version code ⇒ In addition version data and models
- Monitor throughput, latency, resources ⇒ in addition performance metrics, data drift, etc
- CI/CD ⇒ In addition, continuous training

Machine Learning Infrastructure

- Machine learning infrastructure refers to the **hardware, software, tools and systems** needed to develop, train, deploy, and manage models
  - **Hardware**: CPUs, GPUs, TPUs, Memory and Storage, Networking
  - **Software**:
    - **Operating Systems**: Mainly linux
    - **Programming Languages**: Python doe dev, C++ for deploy
    - **Libraries and Frameworks**: TensorFlow, PyTorch, scikit-learn, Keras, OpenCV
    - **ETL** Tools: **e**xtraction, **t**ransformation, and **l**oading of data
  - **Development Tools**
    - IDEs: Jupyter Notebooks, PyCharm, Visual Studio
    - Version Control: Git / Testing Frameworks: pytest
  - **Deployment**
    - Platforms (MLflow, Kubeflow), Containers (Docker, Kubernetes), AutoML / Monitoring

Machine Learning Infrastructure vs MLOps

- Efficient and effective MLOps is built on top of strong infrastructure
- Examples
  - Infra team: creates a cluster of 120 GPUs and tools to submit and monitor a task
    - Also created a tool that can version different the **data**
  - MLOps set process & practices and do necessary developments **to automate**
    - New coding practices
      - pytest any updated coding modules
      - Create new AI updated models (CI/CD)
    - New data ⇒ must version it and be able to follow GDPR
    - Do in garage testing regularly for new features not just internal evalautions
    - Automate e2e as much as possible
    - QA test every quarter new features
    - Use docker to avoid software version problems and easily scale
    - Use tools like MLflow, TensorBoard to log and track experiments.

Machine Learning Infrastructure

- Startups
  - Simple features can be done without complications until serious growth
- Medium-level companies
  - Utilize open source as much as you can
  - Don’t reinvent the wheel
- Large scale companies
  - Develop and customize your own tools
  - Employees cons: disconnected from market tools and frameworks

Deployment: Keep in mind

- Deployment involves many challenges (e.g. scalability, real-time processing, and robustness, data storage, etc) and is a **continuous** activity
- We put an effort to make sure our locally running model will run the same in production!
  - More worse, models in production may degrade over time (e.g. data shift)
  - We need to **monitor and update** the model
- Training features sometimes mayn't be available during production!
- Sometimes, we need to take **security** into considerations!
- While the ML team develops the model, Others may deploy, not you!
- ML can be expensive to productionize

Deployment

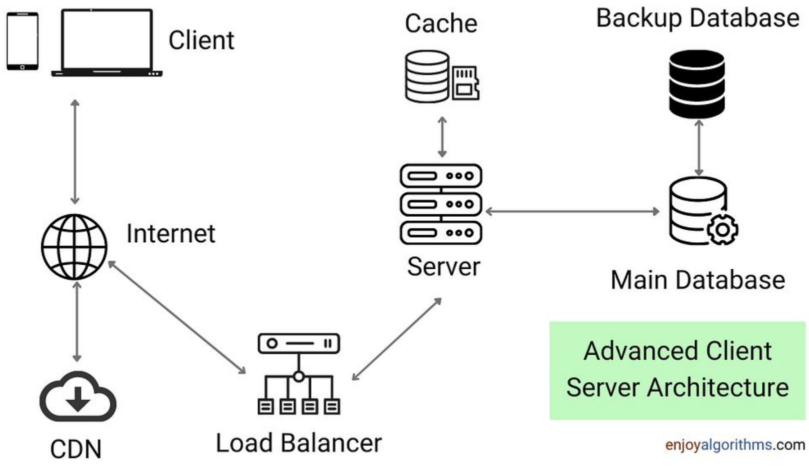
- Model deployment is the process of putting machine learning models into production for the usage
- Assume, We trained a computer vision model that can act as your fitness coach
- Upload 1-min video doing an exercise and get a feedback
- We developed the model
- But, where will we deploy the model?!

Deployment: Where?!

- First, we need to decide, will we provide a mobile app, web app, or both?
- Let's say we will provide a mobile app.
- The model can be:
  - On the mobile itself (example of edge computing)
    - Critical to boost model inference time / limited libraries / different inference hardware
  - On a remote server
    - On-Premises Deployment (Privacy, but big effort / financial challenges)
    - Cloud Deployment (Platform-as-a-Service (PaaS), Infrastructure-as-a-Service (IaaS))
      - Can be so expensive. Not an infinite resource (elastic)
    - Hybrid Deployment of the above choices
    - Multi-Cloud: Minimize your cost
    - **Issues**: Network latency, Round-trip time, Bandwidth, Privacy, Cost, GDPR, Security

Deployment: Challenges with Servers

- Consider the Round-trip time (RTT) when you deploy on a server
    - RTT: duration it takes for a network request to go and get back the response
    - **Network Latency**: depends on factors such as the **geographical distance** between the client and the server, the number of intermediate networks or **hops** the data must pass through, and the **quality** and **speed** of the internet connections involved
    - **Server Processing Time**
    - Server Overhead (infrastructure time to handle the request and response)
    - **Data Transfer Time**
    - **Client Processing Time**
        - post-processing or rendering



Img src

Deployment: Online vs Offline Prediction

- Do you want your model to be real time response (online)?
  - For example, the user uploads his squat video and get immediate response in a minute
- Or is it fine to upload the video and get check it out later?
  - For example, at night you get results of your automatic coach evalautions

Deployment: Online Prediction

- Process and reply back directly to the caller application
  - **Synchronously**: application is **holding** until it receives the prediction
  - **Asynchronously**. Submit and wait fast response preparation
    - **Push**. The system directly push to the caller app
    - **Poll**. The model stores the prediction and the caller polls them
- **Concerns**
  - **Low Latency**: Responses must be generated quickly, often in milliseconds.
    - Model need to be light to allow fast response (or highly hardware optimized)
    - But this create limitations on ML performance (e.g. tradeoff: bigger DNN is better)
  - **System** challenges: Resource-Intensive ⇒ Scalability (many requests) and Availability
  - In some cases, less usage for resources (e.g. GPU with batch of size 1)
- **Examples**: Self-driving cars, Recommendation Systems, Fraud Detection, Chatbots

Deployment: Offline Prediction

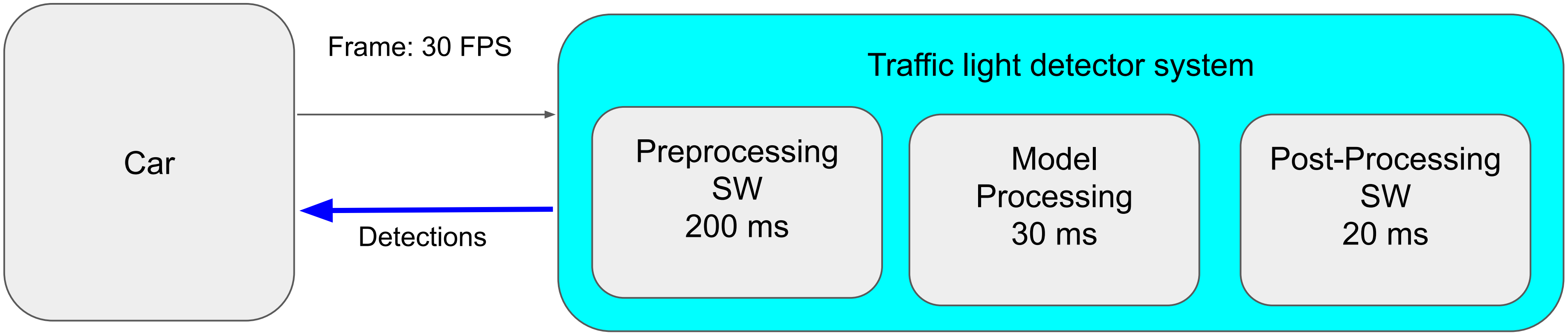
- Aggregate the requests and process (or schedule) them later!
  - It is a **batch** processing
  - No server peak times
  - High Throughput
  - Use more complex models and run on servers
- Examples: Data Analysis tasks (e.g. campaigns), Processing user uploads (e.g. extract objects from images for future utilization)
- Compute resources
  - We need to think about the CPU/GPU/memory needs for both styles

Question!

- Choose the most logical choice between online or batch prediction
- 1) Autonomous Vehicles
- 2) Chatbots
- 3) Youtube
- 4) Fraud Detection
- 5) Stock Market Analysis
- 6) Game Playing
- 7) Siri or Google Assistant
- 8) Large data Text Summarization

Serving Queries: Latency and Refresh Rate

- Latency: **time** for an ML system to process query and generate results
- The car sends 30 FPS frames to a ML model
- The model processes only 10 out of 30 frames (compute budget limitation)
  - Latency of a single request: 200 + 30 + 20 = 250 ms
    - E.g. the answer we send back is about something happened 280 ms ago!
  - Refresh rate: 1000 ms / 10 frames = 100 ms (every 100 ms we refresh the results)



Observation Window

- Assume our self-driving car has a binary classifier: are there traffic lights?
- In practice, the classifier does mistakes and cause **flippings** (1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 ⇒ 2 wrong 0s) but we need it high confidence
- A common approach is to use a sliding window to observe the events
- For example, your system analyze the last 9 classifier outputs and use a voting based on the frequency (if 1s >= ceil(9/2), then 1, else 0)
  - Recall the model processes only 10 out of 30 frames
  - Then we see a new frame each 100 ms
  - For 9 frames, our observation window is 900 ms
  - E.g. Latency of a single request: 200 + 30 + 20 + 900 ms

Observation Window

- The observation window time should be added to the latency
    - Assume there were no traffic lights (000000000)
    - Now we started seeing traffic lights. Here are the next frames
    - GT=Traffic light. Classifier=Traffic light ⇒ 000000001 (voting 1 ⇒ no traffic lights)
    - GT=Traffic light. Classifier=Traffic light ⇒ 000000011 (voting 2 ⇒ no traffic lights)
    - GT=Traffic light. Classifier=Traffic light ⇒ 000000111 (voting 3 ⇒ no traffic lights)
    - GT=Traffic light. Classifier=!Traffic light ⇒ 000001110 (voting 3 ⇒ no traffic lights)
    - GT=Traffic light. Classifier=Traffic light ⇒ 000011101 (voting 4 ⇒ no traffic lights)
    - GT=Traffic light. Classifier=Traffic light ⇒ 000111011 (voting 5 ⇒ traffic lights)
    - GT=Traffic light. Classifier=!Traffic light ⇒ 001110110 (voting 5 ⇒ traffic lights)
    - GT=Traffic light. Classifier=Traffic light ⇒ 011101101 (voting 6 ⇒ traffic lights)
- In best case, we need 5 frames to reflect the change on the system
- In the worst case, we will need the whole observation window (9 frames)
    - Hardware deeper: in the worst case, the event starts directly after the camera capture the current frame (so we have to add an extra frame for latency!)

Deployment Strategies

- Assume you have a human task (e.g. judge an x-ray) or an old ML model
- You would like to **replace** this task with a new ML model
  - **Full Automation**: New AI system will completely replace the human process
  - **Partial Automation**: automating certain parts of a process and remaining for the human
    - Automated parts must be doable in full by ML
    - Examples: Manufacturing processes, data entry automation
  - **AI Assistance**: assists but doesn't replace human actions
    - ChatGPT is an AI assistance / Siri / AI-driven customer support
- But you don’t know how good is such new model?
- There are some strategies for that!

ML Shadow Deployment

- An interesting way is to shadow the task
- A machine learning model operates **in parallel** with human decision-makers, "shadowing" their decisions but not actively contributing to the decision-making process.
- This technique is often used to validate or refine a model before deploying
- Examples: Medical Diagnosis, Financial Decision Making, **Planning in self-driving**

A/B Deployment (Testing)

- Facebook ads system makes billions of dollars yearly
- They have a new model with 3 different configurations all seems will bring more money (*click-through rates*). But which version to deploy?!
- A/B testing evaluate and **compare** two or more versions (e.g., A and B) to determine which one performs better according to business **metrics**
  - Use enough period to gather enough data to make **statistically** significant comparisons
  - Careful from running in parallel if affecting each others (e.g. prices effect on ride sharing)
- It is an example of a "Data-Driven Decision"
  - You may hear your manager says, I like your logic in the decision, but I want a data-driven decision (collect data, run experiments and decide based on metrics)

A/B Testing

- Split the Audience
  - Decide sample size
  - Divide users randomly (avoid bias) into two [**statistically similar]** groups
    - Don’t test ads sample on Egypt vs France (bias)
  - One group sees Version A (old), and the other group sees Version B (new).
  - Consider **Confounding variables** [external factors that might affect the outcome]
- Select a **statistical test** that matches the data distribution, scale of measurement, and hypothesis being tested (e.g., t-test, chi-squared test)
  - Set **Confidence Level** and **Significance Level** (typically 95% and a 0.05)
  - Be careful from running **multiple A/B** tests simultaneously
    - Bonferroni *correction* to adjust the significance level
  - Evaluate **Power of the Test** (ability to detect a true difference between A & B)
    - **Null hypothesis**: there is no difference between A and B
    - Will the hypothesis test correctly reject it? TP, FP, TN, **FN**

Other Deployments

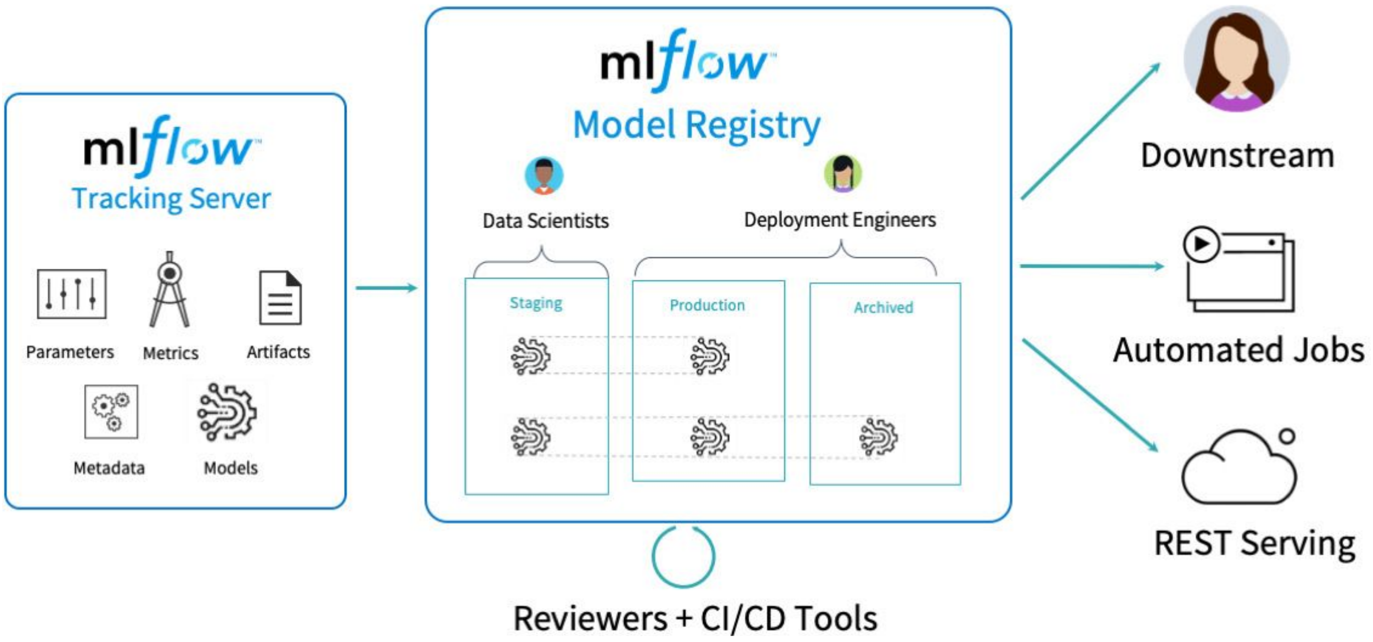
- When we roll out the model to a **small subset** of users before full-scale deployment, we call it **Canary** Deployment
- If we do the deployment in stages **gradually increasing** the number of users using the new model till the full scale, this is called **Rolling** Deployment
- If you can **switch** between **running** systems (old, new), we may call it Blue/Green Deployment (e.g. new system is good - disaster roll back)
- Tips
  - Don’t focus much on wording of the techniques. Just get the **possibilities**
    - Old vs New - Gradual replacements - can roll back to old model - comparing
  - In practice, we can build a hybrid style

Monitoring Deployed Models

- When we develop a model, we watch graphs for its loss/accuracy
- When we deploy a model, we may watch many things (**dashboards**)!
  - Monitor inputs (ranges, missing items, etc)
  - Monitor resources (e.g. memory usage, cpu)
  - Monitor business metrics (e.g. clicks on ads, revenue)
  - Monitor technical metrics (e.g. latency)
  - In general: think in all direct/indirect concerns that is relevant to your ML model
- Automatically, use **thresholds** to fire alerts if something is wrong
- In practice, we may have a pipeline of models that need all to be monitored

Model Registry

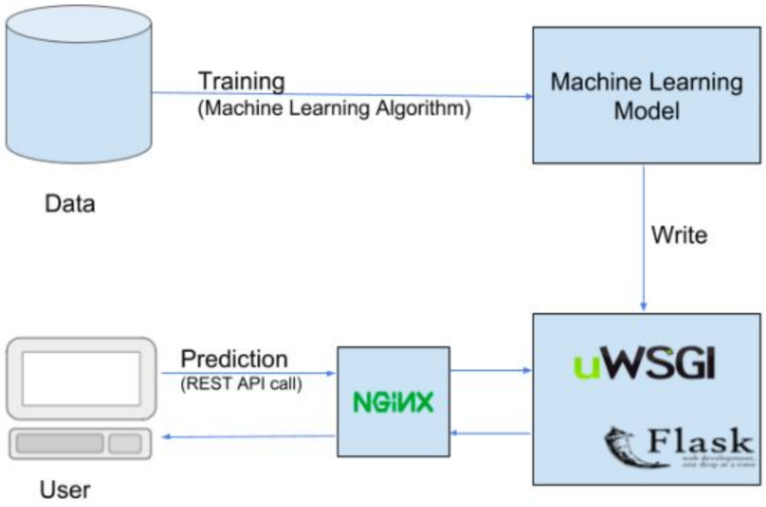
- A centralized repository that stores machine learning models and their associated metadata



Img src

REST API

- We can wrap our ML model using a RESTful web service and make it accessible over **HTTP**
    - Frameworks: Flask, FastAPI, Django REST Framework, uWSGI + Nginx



Img src

Online ML Services

- There are many services that mid-level companies can leverage for supporting e2e/MLOPs
- **Sagemaker**
  - provides tools to build, train, and deploy machine learning models
  - Built-in Algorithms and Frameworks
  - tools for data exploration, cleaning, and preprocessing
  - Automatic Model Tuning
  - Distributed Training
  - Pipelines for CI/CD
- **Azure**
  - Azure ML Studio, AutoML, MLOps capabilities, pre-trained models, Scalability, Azure Kubernetes, Workflow

Online Learning

- Sometimes, we deploy and monitor the system for the degradation
- However, in some problems we know it will degrade in a day/week/month, etc
  - Ads system updated daily (to reflect the new added advertisements)
    Search engines might be updated every month
  - Tip: the degradation period may change over time!
- OL: The model is updated **in real-time** as new **streamed** data points arrive
  - In Offline Learning, we just learn and deploy. We later may refine later on collected data
- Stochastic Gradient Descent (**SGD**): One of the simplest and most widely-used online learning algorithms.
  - It can update the model's parameters using each new example
  - Careful with Catastrophic Forgetting in Neural Networks
- Challenges: deployment management, Drift, New Data Quality, natural labels needs, etc

Relevant Materials

- Andrew NG: Machine Learning Engineering for Production (MLOps) Specialization
- Udemy - Deployment of Machine Learning Models - **Dr. Ahmad ElSallab**
- MLOps Channel
- Model compression and optimization
- Optimizing Models for Deployment and Inference
- Automated Canary Analysis at Netflix with Kayenta
- Udemy has some content on deployment/azure/MLOps
- Edge Computing: link link

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."