Key Challenges

What makes deployment hard?

- ML/statistical issues
- Software engineering issues

Concept Drift and Data Drift

One of the challenges of deployment is concenpt and data drift. Loosely, it means **what if your data changes after your system has already been deployed?**

- Example: detecting scratches on smartphone manufacturing. You've trained the model on images with a certain lighting conditions but when deployed the images you receive have a different lighting condition (probably because the lighting in the factory changes).

- **Example: Speech recognition**

    - Training set:
        - Purchase data, historical user data with transcripts
    - Test set:
        - Data from a few months ago (relatively recent data)
    - How has the data changed?
        - The language changes
        - People are using a newer smartphone model which has a different microphone and sounds different.
        - This can result in the performance of the speech recognition system to degrade.

- It is important to recognize if the data has changed and if you need to update your learning algorithm as a result.

- Data changes:

    - Gradual: English language, it changes but very slowly (with new vocabulary introduced, etc.)
    - Suddenly: shock to the system. When COVID happened, a lot of credit card fraud detection systems started not working because the purchase patterns of individuals suddenly changed.

Concept drift: if the desired mapping $X \rightarrow y$ changes.
- *Example:* before COVID-19 perhaps for a given user, a lot of surprising online purchases should have flagged that account for fraud. But after COVID started, maybe those same purchases would not have really been any cause for alaram.
- *Example:* Let's say x is the size of a house and y is the home price. If because of inflation (or changes in the

market), houses become more expensive over time, then the same size house will end up having higher price.

So, in **concept drift** refers to when y from the $X \rightarrow y$ mapping changes (i.e. the definition of what is y given X changes), whereas **data drift** refers to when the input data distribution, X, changes (i.e. the distribution of X changes). In example above, data drift would be that people start building houses in different sizes.

Note: When you deploy a ML system, one of the important tasks is to make sure you are able to detect and manages any changes to the data.

Software Engineering Issues

Let's say you want to deploy a prediction service which receives X and returns prediction y. You have a lot of design choices as to how to implement this piece of software.

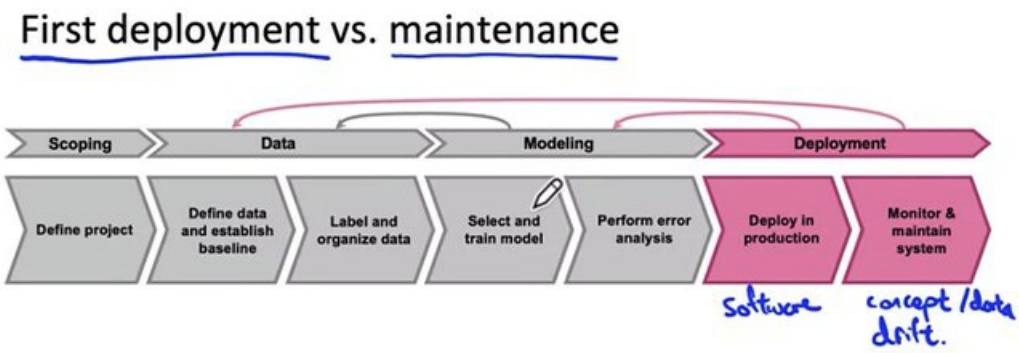
Here's a checklist of questions to help you make appropriate decisions:

- **Realtime or batch systems**
- **Cloud vs. edge/browser**
- **Compute resources (CPU/GPU/memory)**
- **Latency, throughput (QPS $\rightarrow$ Query Per Second)**
- **Logging**
- **Security and privacy**

Note: Advantages of edge deployment vs cloud deployment

- Lower latency
- Can function even if the network connection is down
- Less network bandwidth needed

Note: One of the things you see when you're building ML systems is that the practices for the very deployments will be quite different compared to when you're updating or maintaining a system that has already been deployed. Deployment is not about getting to the finish line. First deployment is maybe just half of the way. The other half starts after your first deployment with tasks like feeding the data back and keep updating the model, or monitoring and managing possible concept or data drift cases.



Note: A full cycle of a ML project is an **iterative process** where during the later stage we might go back to an earlier stage.

Deployment Patterns

Common deployment cases:

1. **New product/capability**
    - Example: You're offering a speech recognition service that you have not offerred before.
        - In this case, a common design pattern is to start up a small amount of traffic and then gradually ramp it up.
2. **Automate/assist with manual task**
    - Something that's already done be a person, but we'd now want to use a learning algorithm to either automate or assist with that task.
    - Example: people in the factory inspecting smartphones scratches, but now you want to use a learning algorithm to assist/automate that task.
    - The fact that people were doing this gives a few more options for how to deploy.
    - Shadow model deployment takes advantage of this.
3. **Replace previous ML systems**
    - You've already been doing a task with previous implementation of a ML system, but now you want to replace it with hopefully a better one.

Two key ideas and recurring themes here are:

- **Gradual ramp up with monitoring**
- **Rollback** $\rightarrow$ if for some reason you ML model is not working, roll back to earlier system.

Shadow Mode Deployment

People are inspecting smartphone for scratches. Now, you want to automate some of this work with a learning algorithm.

Note: When you have people initially doing a task, one common deployment pattern is ***shadow mode deployment***:

- ML system shadows the human (or even an older ML system) and runs in parallel.
- ML system's output not used for any decisions during this phase, i.e. in discrepancy cases between human and ML, we always consider human to be correct.

The *purpose* of shadow mode deployment is that it allows you to gather data of how the learning algorithm is performing and how it compares to the human judgement. You can use that maybe to allow the ML take some real decisions in the future.

Canary Deployment

When you're ready to let a ML system start making real decisions, a common deployment pattern is **canary deployment**.

In canary deployment:

- Roll out to small fraction (say 5%) of traffic initially.
- Monitor the system and ramp up traffic gradually.

Canary deployment allows to spot problems, hopefully, early on before there are maybe overly large consequences to a system.

Blue Green Deployment

Blue refers to the old version of your software and *green* is the new version. In blue green deployment, you have an old service (blue), you spin up a new service (green), and you'd have the router ***suddenly*** switch the traffic from the old one to the new one.

The advantage of this deployment method is that there's an easy way to enable rollback. If something goes wrong, you can just very quickly have the router send the traffic back to the older system (assuming that you've kept the older version).

Degrees of Automation

One of most useful frameworks on how to deploy a system is to think about deployment not as a 0, 1 i.e.

either deploy or not deploy, but instead to design a system thinking about ***what is the appropriate degree of automation***.

Example: in visual inspection of smartphones, here's a range of automation you could have: