

# Deploying Machine Learning Models in Production

In the fourth course of Machine Learning Engineering for Production Specialization, you will learn how to deploy ML models and make them available to end-users. You will build scalable and reliable hardware infrastructure to deliver inference requests both in real-time and batch depending on the use case. You will also implement workflow automation and progressive delivery that complies with current MLOps practices to keep your production system running. Additionally, you will continuously monitor your system to detect model decay, remediate performance drops, and avoid system failures so it can continuously operate at all times.

Understanding machine learning and deep learning concepts is essential, but if you're looking to build an effective AI career, you need production engineering capabilities as well. Machine learning engineering for production combines the foundational concepts of machine learning with the functional expertise of modern software development and engineering roles to help you develop production-ready skills.

## Week 4: Model Monitoring and Logging

### Contents

<b>Week 4: Model Monitoring and Logging</b>	<b>1</b>
Why Monitoring Matters	2
Observability in ML	5
Monitoring Targets in ML	7
Logging for ML Monitoring	10
Tracing for ML Systems	14
What is Model Decay?	17
Model Decay Detection	19
Ways to Mitigate Model Decay	21
Responsible AI	24
Legal Requirements for Secure and Private AI	27
Anonymization and Pseudonymization	32
Right to be Forgotten	35
References	39

## Why Monitoring Matters

### ML Lifecycle Revisited



- Welcome back, this week starts with a discussion of monitoring your models in production, beginning with a look at why monitoring matters, let's get started.
- By now you're familiar with this process which starts with building your models but **doesn't end with deployment**.
- The last task monitoring your model in production is an ongoing task for as long as your model is in production.
- The data that you gather by monitoring will guide the building of the next version of your model and make you aware of changes in your model performance.
- So, as you can see here, this is a cyclical iterative process which requires the last step monitoring in order to be complete.
- You should note here that this diagram is only looking at monitoring which is directly related to your model performance.
- But, you will also need to include monitoring of the systems and infrastructure which are included in your entire product or service such as databases and web servers.
- That kind of monitoring is only concerned with the basic operation of your product or service and not the model itself but is critical to your users experience.
- Basically, if the system is down, it really doesn't matter how good your model is.

## Why Monitoring Matters

*“An ounce of prevention is worth a pound of cure”*

- Benjamin Franklin

- Benjamin Franklin once wrote, an ounce of prevention is worth a pound of cure, or in metric terms maybe, a gram of prevention is worth a kilo of cure.
- In 1733, he visited Boston and was impressed with the fire prevention measures that the city had established. So when he returned home to Philadelphia, he tried to get his city to adopt similar measures, Franklin was talking about preventing actual fires.
- But in our case, you might apply this same idea to preventing fire drills, the kind of fire drills where your system is performing badly and it's suddenly an emergency to fix it.
- It's these kinds of fire drills that can happen if you don't monitor your model performance.

## Why do you need monitoring?

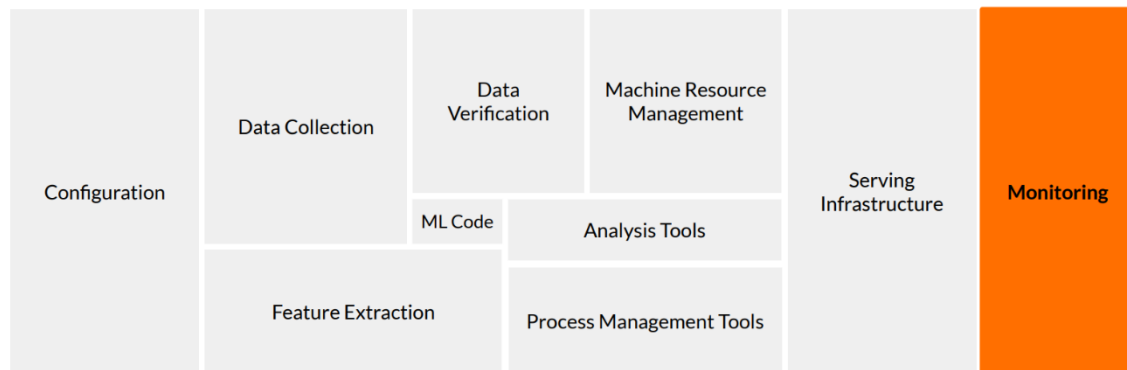
- Immediate Data Skews
    - Training data is too old, not representative of live data
  - Model Staleness
    - Environmental shifts
    - Consumer behaviour
    - Adversarial scenarios
  - Negative Feedback Loops
- If your training data is too old, even when you first deploy a new model, it can have **immediate data skews**.
  - Without monitoring right from the start, you may easily be unaware of the problem and your model will not be accurate even when it's new.
  - Of course, as previously discussed, models will also become stale or inaccurate because the world constantly changes and the training data you originally collected might no longer reflect the current state.
  - Again, without monitoring, you are unlikely to be aware of the problem.
  - You can also have negative feedback loops; this turns out to be a more complex issue that arises when you automatically train your models on data collected in production.
  - If this data is biased or corrupted in any way, then the models trained on that data will perform poorly.
  - Monitoring is important even for automated processes because they too can have problems.

## Monitoring in ML Systems

ML Monitoring (functional monitoring)	System monitoring (non-functional monitoring)
Predictive performance	System performance
Changes in serving data	System status
Metrics used during training	System reliability
Characteristics of features	

- ML monitoring, or functional monitoring, deals with keeping an eye on model predictive performance and changes in serving data.
- These include the metrics, the model optimized during training, and the distributions and characteristics of each feature in the serving data.
- System monitoring, or non-functional monitoring, refers to monitoring the performance of the entire production system, the system status and the reliability of the serving system.
- This includes queries per second, failures, latency, resource utilization etc.

## Why is ML monitoring different?



- You may ask yourself, why is ML monitoring different than software monitoring.
- **Unlike a pure software system, there are two additional components to consider in an ML system, the data and the model.**
- Unlike in traditional software systems, the accuracy of an ML system depends on how well the model reflects the world it is meant to model, which in turn depends on the data used for training and on the data that it receives while serving requests.
- It's not simply a matter of monitoring for system failures like SEG faults and out of memory or network issues.
- The model and the data require additional, very specialized monitoring as well.
- Code and config also take on additional complexity and sensitivity in an ML system due to two aspects, entanglement and configuration.
- **Entanglement refers to the issue where changing anything, changes everything.**
- Here, you need to be **careful with feature engineering and features selection** and understand your model sensitivity.
- Configuration can also be an issue because model hyperparameters, versions and features are often controlled in a system config and the slightest error here can cause radically different model behavior that won't be picked up with traditional software tests.
- Again, requiring additional very specialized monitoring.

### What is observability?

- *Observability* measures how well the internal states of a system can be inferred by knowing the inputs and outputs
  - Observability comes from control system theory
  - Observability and controllability are closely linked
- Next, let's explore the concept of observability and how it applies to ML.
  - First, what is observability?
  - Observability measures how well you can infer the internal states of a system by just knowing the inputs and outputs.
  - For ML, this means monitoring and analyzing the prediction requests and the generated predictions from your models.
  - Observability isn't a new concept; it actually comes from **control system theory** where it has been well established for decades.
  - In control system theory, observability and controllability are closely linked.
  - **You can only control a system to the extent that you can observe it.**
  - Looking at an ML-based product or service, this maps to the idea that controlling the accuracy of the results overall, usually across different versions of the model, requires observability.
  - **This also adds to the importance of model interpretability.**

### Complexity of observing modern systems

- Modern systems can make observability difficult
    - Cloud-based systems
    - Containerized infrastructure
    - Distributed systems
    - Microservices
- In ML systems, observability becomes a more complex problem since you need to consider multiple interacting systems and services such as cloud deployments, containerized infrastructure, distributed systems, and microservices.
  - This generally means that there are a **substantial number of systems which you need to monitor and aggregate.**

- This often means relying on vendor monitoring systems to collect and sometimes aggregate data because the observability of each instance can be limited.
- For example, monitoring CPU utilization across an autoscaling containerized application is much different than simply monitoring CPU usage on a single server.

## Deep observability for ML

- Not only top-level metrics
  - Domain knowledge is important for observability
  - TensorFlow Model Analysis (TFMA)
  - Both supervised and unsupervised analysis
- **Observability is about making measurements.**
  - Just like when you're analyzing your model performance during training, measuring top-level metrics is not enough and will provide an incomplete picture.
  - You need to slice your data to understand how your model performs for various data subsets.
  - For example, in an autonomous vehicle, you need to understand performance in both rainy and sunny conditions and measure them separately.
  - More generally speaking, **data slices** provide a useful way to analyze different groups of people or different types of conditions.
  - This means that domain knowledge is important in observing and monitoring your systems and production just like it is when you're training your models.
  - In general, **it's your domain knowledge that will guide how you slice your data.**
  - The TFX framework and TensorFlow model analysis (TFMA) are very powerful tools and include functionality for doing observability analysis on multiple slices of data for your deployed models.
  - This is true for both supervised and unsupervised monitoring of your models.
  - In a supervised setting, the true labels are available to measure the accuracy of your predictions.
  - In contrast, **in an unsupervised setting, you'll monitor for things like the means, medians, ranges, and standard deviations of each feature.**
  - In both supervised and unsupervised settings, you need to slice your data to understand how your system behaves for different subsets.
  - Going back to the autonomous vehicle example, slicing by weather condition is important to avoid things like making poor driving decisions in the rain.

## Goals of ML observability

- Alertable
    - Metrics and thresholds designed to make failures obvious
  - Actionable
    - Root cause clearly identified
- 
- The main goal of observability in the context of monitoring is to prevent or act upon system failures.
  - For this, the observations need to provide **alerts** when a failure happens, and ideally provide recommended actions to bring the system back to normal behavior.
  - More specifically, alertability refers to designing metrics and thresholds that make it very clear when a failure happens.
  - This may include defining rules to link more than one measurement to identify a failure.
  - Knowing that your system is failing is a good start, but an **actionable recommendation** based on the nature of the failure is way more helpful to correct this behavior.
  - Actionable alerts clearly define the root cause of the system's failure.
  - At a bare minimum, your system should gather sufficient information to enable root cause analysis. Both alertability and actionability are goals, and the effectiveness of your system is a reflection of how well it achieves these goals.

## Monitoring Targets in ML

### Basics: Input and output monitoring

- Model input distribution
  - Model prediction distribution
  - Model versions
  - Input/prediction correlation
- 
- Let's look now at the kind of things that you can observe and monitor in an ML system.
  - Starting with the basics, you can monitor the inputs and outputs of your system.
  - The inputs in a deployed system are the prediction requests, each of which is a feature vector.
  - You can use statistical measures of each feature, including their distributions and look for changes that may be associated with failures.

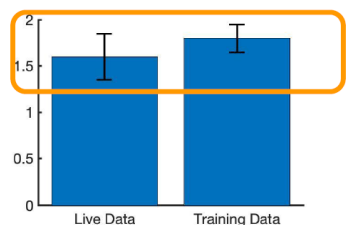
- Again, this should not be just top-level measurements, but **measurements of slices** that are relevant to your domain.
- The outputs are the model's predictions which you can also monitor and measure.
- This should include an understanding of the deployment of different model versions to help you understand how different versions perform.
- You should also consider performing **correlation analysis** to understand how changes in your inputs affect your model outputs.
- And again, this should be done on **slices** of your data
- For example, correlation analysis can help you detect how seemingly harmless changes in your inputs cause prediction failures.

## Input Monitoring

<p><b>Do these check out?</b></p>	<ul style="list-style-type: none"> <li>• Errors: Input values fall within an allowed set/range?</li> <li>• Changes: Distributions align with what you've seen in the past?</li> <li>• Per slice, e.g., marital status (single/married/widowed/divorced)</li> </ul>
---------------------------------------	--

- The prediction requests, whether you're doing real time or batch predictions, form a large part of the observable data that you have for deployment
- For each feature, you should monitor for errors such as values falling outside an allowed range or a set of categories, where these error conditions are often defined based on domain knowledge.
- You should also **monitor how each feature distribution changes over time** and compare those to the training data,
- Monitoring for errors and changes is better done with sliced data so that you can better understand and identify potential system failures.

## Prediction Monitoring

<p><b>Statistical significance</b></p>	<div data-bbox="826 1570 1244 1794">  <p>SEM</p> </div> <ul style="list-style-type: none"> <li>• Unsupervised: Compare model prediction distributions with statistical tests <ul style="list-style-type: none"> <li>○ e.g., median, mean, standard deviation, min/max values</li> </ul> </li> <li>• Supervised: When labels are available</li> </ul>
--	--



- Statistical testing and comparisons are the basic tools that you can use to analyze your data.
- Typical **descriptive statistics** include median, mean, standard deviation, and range values.
- For monitoring model predictions, you can also use statistical testing
- Sometimes in scenarios such as predicting click through where labels are available, you can also do comparisons between known labels and model predictions
- In this figure, you can see that if the variables are normally distributed, then you would expect the mean values to be **within the standard error of the mean interval**.
- It's also important to consider that if you have **altered the distributions** of the training data to correct for things like class imbalance or fairness issues, then you need to take that into account when comparing to the distributions of the input data gathered through monitoring prediction requests

## Operational Monitoring

ML engineering	Software engineering
Latency	Receiving an HTTP request
IO / Memory / Disk Utilisation	Entering/leaving a function
System Reliability (Uptime)	A user logging in
Auditability	Reading from net / writing to disk

- Monitoring in the realm of software engineering is far more well established.
- The operational concerns around our ML system may include monitoring system performance, in terms of measures like latency, or IO, and memory, or disk utilization, or system reliability in terms of up time, and monitoring can even happen while taking auditability into account.
- In software engineering, talking about monitoring is strictly speaking, **talking about events**
- Events can be almost anything, ranging from receiving an http request, entering or leaving a function (which may or may not contain ML code), a user logging in, reading from network or writing to the disk, and so on,
- All of these events listed here have some context.
- Having all of the context for all of the events would be great for debugging and understanding how your systems perform in both technical and business terms.
- But collecting all the context information is often not practical, as the amount of data to process and store could be very large
- So it's important to understand the most relevant context and try to gather that information.

### Steps for building observability

- Start with the out-of-the-box logs, metrics and dashboards
  - Add agents to collect additional logs and metrics
  - Add logs-based metrics and alerting to create your own metrics and alerts
  - Use aggregated sinks and workspaces to centralize your logs and monitoring
- Logging is almost always the basis for collecting the data that you will use to monitor your models and systems.
  - **A log is an immutable time stamped record of discrete events** that happened over time for your ML system, along with additional information.
  - Let's take a deeper look at logging
  - To avoid making the same mistake twice, it's important to learn from history. For ML systems, the same logic applies.
  - This is where logging becomes really handy and more so when building observability.
  - Let's explore how to do that.
  - You can start with the out of the box logs and metrics.
  - These will usually give you some basic overall monitoring capabilities, which you can then add to.
  - For example, in Google's compute engine platform, if you need additional application logs, you can install agents to collect those logs.
  - Cloud monitoring collects metrics from all of the cloud services by default, which you can then use to build dashboards.
  - When you need additional application or business level metrics, you can use those custom metrics to monitor over time.
  - Then using **aggregate sinks and workspaces** allows you to **centralize your logs** from many different sources or services in order to create a unified view of your application.

### Logging

**Log:** An event log (usually just called “logs”) is an immutable, time-stamped record of discrete events that happened over time.

- To clarify exactly what I mean a log is an immutable time stamped record of discrete events that happened over time.

- This also includes debugging or profiling messages that are printed to the log from your application, as well as automatically generated warning errors and debug messages, depending on the verbosity settings for your logging.

## Tools for building observability

- Google Cloud Monitoring
- Amazon CloudWatch
- Azure Monitor



Google Cloud Monitoring

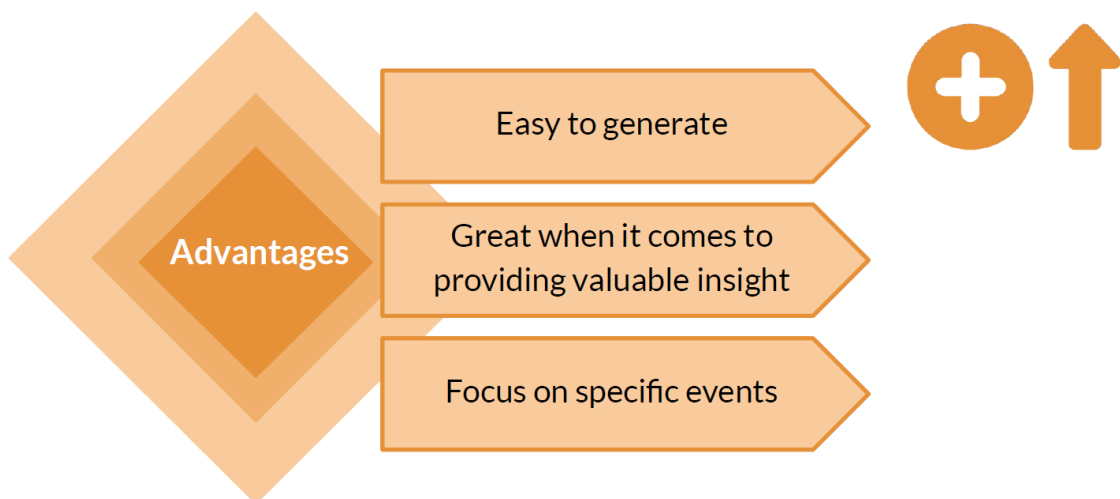


Amazon CloudWatch



- Cloud providers also offer managed services for logging of cloud based distributed services.
- These include Google Cloud Monitoring, Amazon Cloudwatch, and Azure Monitor, as well as several managed offerings from 3rd parties.

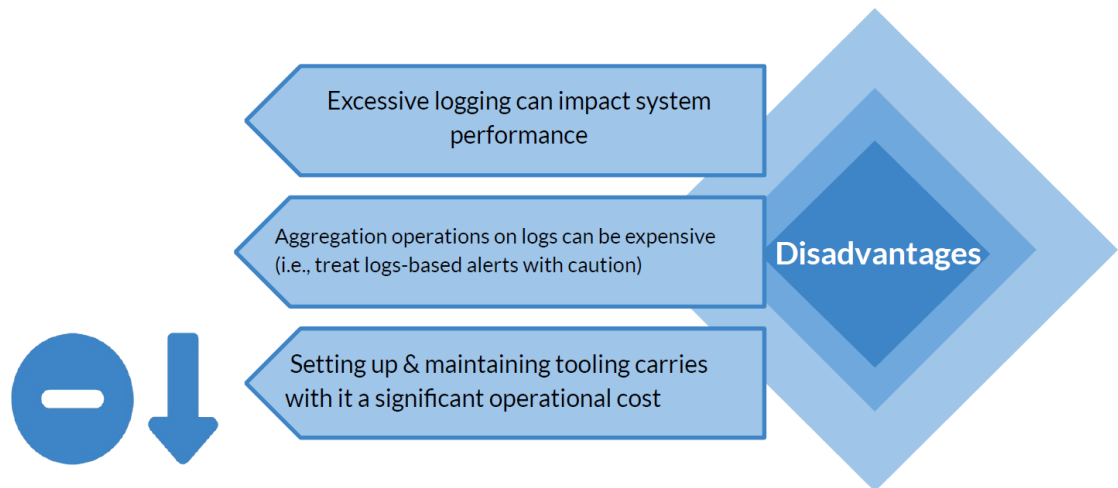
## Logging - Advantages



- Log messages are very easy to generate since it is just a string, a blob of JSON, or typed key value pairs.
- Event logs provide valuable insight along with context providing detail that averages and percentiles don't surface.
- However, it's not always easy to provide the right level of context without obscuring the really valuable information in too much extraneous detail.
- While metrics show the trends of a service or an application, **logs focus on specific events**.

- This includes both log messages printed from your application as well as warnings, errors or debug messages which are generated automatically.
- The information logs can be used to investigate incidents and to help with root cause analysis.

## Logging - Disadvantages



- But logging isn't perfect. For example, **excessive logging can negatively impact system performance**.
- As a result of these performance concerns, aggregation operations on logs can be expensive, and for this reason alerts based on logs should be treated with caution.
- On the processing side, raw logs are almost always normalized, filtered and processed by a tool like LogStash, FluentD, or Scribe before they are persisted in a data store like elastic search or Big Query.
- Setting up and maintaining this tooling carries with it a **significant operational cost**.
- One of the key advantages of managed services is that they remove this cost.

## Logging in Machine Learning

### Key areas

- Use logs to keep track of the model inputs and predictions

### Input red flags

- A feature becoming unavailable
- Notable shifts in the distributions
- Patterns specific to your model

- Much of the discussion so far has centered around how you could use metrics to monitor your input data and predictions in an ML system.
- This is usually the basic way to collect data to monitor an application.
- Some of the **red flags** to watch out for may include basic things like a feature becoming unavailable, especially when you're including historical data in your prediction requests which needs to be retrieved from a data store.
- In other cases, **notable shifts in the distribution** of key input values are important.
- For example, a categorical value that was relatively rare in the training data becomes more common.
- Pattern specific to your model for example, in an NLP scenario, a sudden rise in the number of **words not seen in the training data**.
- That can also be another sign of a potential change which can lead to problems.

## Storing log data for analysis

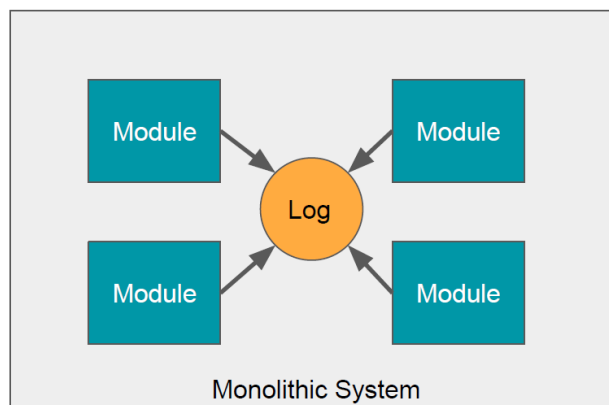
- Basic log storage is often unstructured
  - Parsing and storing log data in a queryable format enables analysis
    - Extracting values to generate distributions and statistics
    - Associating events with timestamps
    - Identifying the systems
  - Enables automated reporting, dashboards, and alerting
- How you store your log data can have a significant impact on how easily it can be queried for analysis.
  - At this point, you should consider parsing out and storing your input and prediction data along with any labels that you're able to gather in queryable data store, such as a database or a search engine based tool like Elasticsearch.
  - This enables analysis for things like generating the distributions and statistics of your features which can be tracked and compared over time.
  - **By associating each item with a time stamp**, you can also order the data, which is important for identifying trends and seasonality.
  - In addition, by identifying the systems involved, you can help with root cause analysis of system failures.
  - Having this data in queryable data store also enables **offline automated reporting** dashboards and alerting.

## New Training Data

- Prediction requests form new training datasets
  - For supervised learning, labels are required
    - Direct labeling
    - Manual labeling
    - Active learning
    - Weak supervision
- 
- Log data is of course also the basis for your next training data set.
  - At the very least, collecting prediction requests should provide the feature vectors that are representative of the current state of the world that your application lives in.
  - So this data is very valuable. Let's consider labeling issues and labeling techniques for a moment.
  - If you're lucky in your domain, you'll be able to use **direct labelling**. For example, for recommender systems, **you can usually capture the user behavior after a recommendation is made to determine if the right options were recommended.**
  - In other cases, you will need to use **manual labeling**, which can be slow and expensive, but is also sometimes the only viable option.
  - Techniques like **active learning** help reduce cost by only selecting the most important examples to label. And that includes shaping your data set for issues like class imbalance and fairness.
  - Finally, **weak supervision** is a powerful technique with significant advantages but also challenges.
  - What's most important here is that you capture this valuable data so that you can keep your model in sync with a changing world

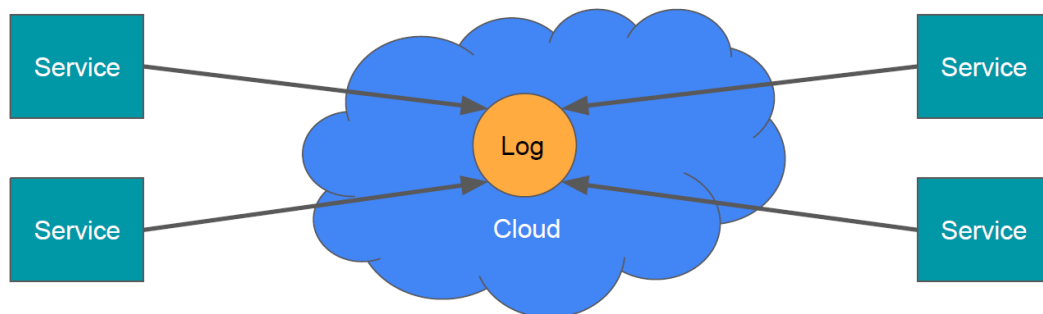
## Tracing for ML Systems

### Distributed Tracing



- **Tracing** focuses on monitoring and understanding system performance, especially for **microservice-based applications**.
- Things get more interesting when considering distributed systems.
- Suppose you're trying to troubleshoot a prediction latency problem, suppose your system is made of many independent services and the prediction is generated through many downstream services, you have no idea which of those services are causing the slowdown.
- You have no clear understanding of whether it's a bug, an integration issue, a bottleneck due to a poor choice of architecture, or poor networking performance.
- In monolithic systems, it's relatively easy to collect diagnostic data from different parts of a system.
- All the modules might even run within one process and share common resources for logging.

## Distributed Tracing



- Solving this problem becomes even more difficult if your services are running as separate processes in a distributed system.
- **You can't depend on the traditional approaches that help diagnose monolithic systems.**
- You need to have **finer grained visibility** into what's going on **inside each service** and how they interact with one another over the lifetime of a user request.
- It becomes harder to follow a call starting from the front-end web server to all its back-ends until the prediction is returned back to the user
- You'll notice here that we're really focusing on online serving.

## Tools for building observability

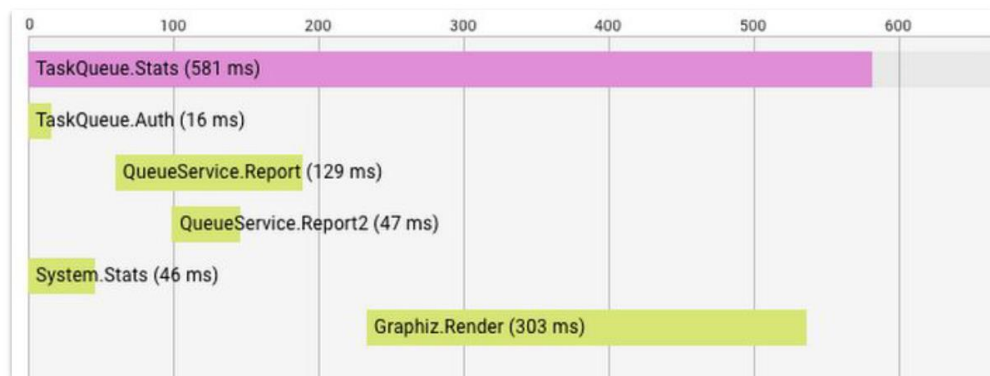
- Sequencing and parallelism of service requests
- Distributed tracing
  - Dapper
  - Zipkin
  - Jaeger



- To properly inspect and debug issues with latency for requests in distributed systems, you need to understand the sequencing and parallelism of the services, and the latency contribution of each to the final latency of the system.
- To address this problem, Google developed the **distributed tracing system, Dapper**, to instrument and analyze its production services.
- The Dapper paper has inspired many open-source projects, such as Zipkin and Jaeger
- Dapper style tracing has emerged as an industry wide standard.

## Dapper-Style Tracing

- Propagate trace between services
- A trace is a call tree, made up of one or more spans



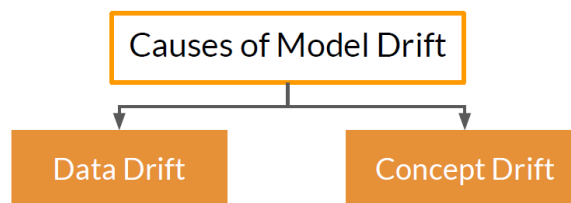
- In service-based architectures, Dapper-style tracing works by **propagating tracing data between services**.
- Each service annotates the trace with additional data and passes the tracing header to other services until the final request completes.
- Services are responsible for uploading their traces to a tracing back-end.
- The tracing back-end then puts related latency data together like pieces of a puzzle.
- Tracing back-ends also provide UIs to analyze and visualize traces.
- Each trace is a **call tree**, beginning with the entry point of a request and ending with the server's response, including all of the RPCs along the way.
- Each trace consists of small units called spans



## What is Model Decay?

### Model Decay

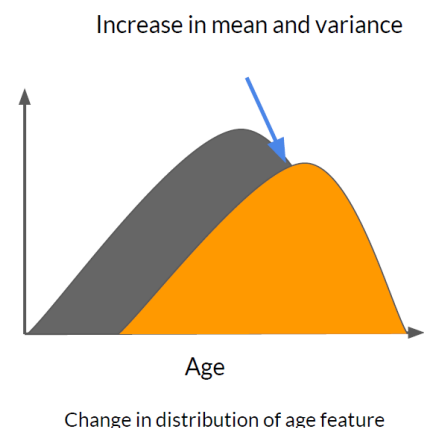
- Production ML models often operate in a dynamic environments
- The ground truth in dynamic environments changes
- If the model is static and does not change, then it gradually moves farther and farther away from the ground truth



- One of the key problems in many domains is model decay. Let's explore this briefly now to get a better understanding of why this might happen and how to prevent it.
- Production ML models often operate in dynamic environments.
- Over time, dynamic environments change. That's what makes them dynamic.
- Think of a recommender system that is trying to recommend which music to listen to.
- Music changes constantly, with new music becoming popular and taste changing.
- If the model is static and continues to recommend music that has gone out of style, then the quality of the recommendations will decline.
- The model is moving away from the current ground truth.
- It doesn't understand the current styles because it hasn't been trained for them.
- So there are **two main causes of model drift - Data drift and concept drift**. Let's talk about each of them.

### Data Drift (aka Feature Drift)

- Statistical properties of input changes
- Trained model is not relevant for changed data
- For eg., distribution of demographic data like age might change over time

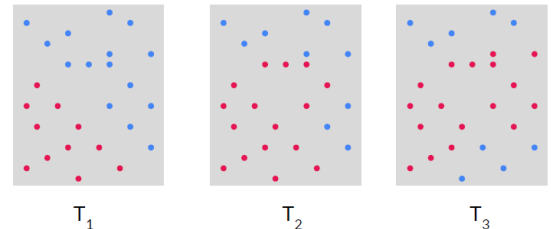


- Data drift occurs when **statistical properties of the input**, the features, changes.

- As the input changes, the prediction requests, the input moves farther away from the data that the model was trained with, and model accuracy suffers.
- Changes like these often occur in **demographic features** like age, which may change over time.
- The graph on the right shows how there is an increase in mean and variance for the age.
- This is data drift.

## Concept Drift

- Relationship between features and labels changes
- The very meaning of what you are trying to predict changes
- Prediction drift and label drift are similar

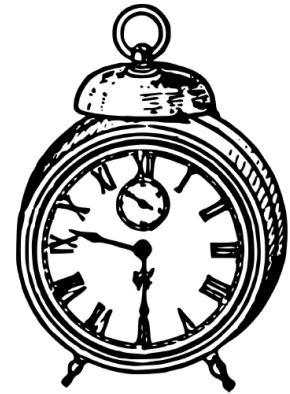


Change in relationship between the features and the labels

- Concept drift occurs when the **relationship between the features and the labels changes**.
- When a model is trained, it learns a relationship between the inputs and ground truth or labels.
- If the relationship between the inputs and the labels changes over time, it means that the very meaning of what you are trying to predict changes.
- The world has changed, but our model doesn't know it.
- For example, take a look at the graph on the right side. You can see that the distribution of the features for the two classes, the blue and red dots, changes over time intervals, T1, T2, and T3.
- If your model is still predicting for T1 when the world has moved to T3, many of its predictions will be incorrect.
- Example of concept drift: Suppose that you have a very accurate model for a social app that uses several features to predict whether a user is a spammer or not. You trained the model with a particular idea of what a spammer was, for example, a user who sends ten messages in one minute. Over time, the app grew and became more popular, but the outcome of the predictions has drastically changed. As people are chatting and messaging more, now sending ten messages in a minute becomes normal and not something that only spammers do. The model's original idea about what it means to be a spammer has changed, and now sending ten messages in a minute becomes normal. In other words, the concept of spammers has drifted. Consequently, since you haven't updated your model, it will predict these non-spammers as spammers (a false positive).
- I should also mention here that there are related forms of drift known as **prediction drift** (where drifts solely in your model's predictions) and **label drift**, but I won't be discussing them in detail in this course.

## Detecting Drift on Time

- Drift creeps into the system slowly with time
- If it goes undetected, model accuracy suffers
- Important to monitor and detect drift early



- If you don't plan ahead for drift, it can slowly creep into your system over time.
- How fast your system drifts depends on the nature of the domain that you're working in.
- Some domains like markets can change within hours or even minutes. Others change more slowly.
- If drift, either data drift or concept drift or both, is not detected, then your model accuracy will suffer, and you won't be aware of it.
- This can lead to emergency retraining of your model, which is something to avoid.
- So, monitoring and planning ahead are important. Knowing that you've planned and have systems in place just might make it easier for you to sleep at night.

## Model Decay Detection

### Detecting Concept and Data Drift

#### Log Predictions (Full Requests and Responses)

- Incoming prediction requests and generated prediction should be logged
  - If possible log the ground truth that should have been predicted
    - Can be used as labels for new training data
  - At a minimum log data in prediction request
    - This data is analysed to detect data drift that will cause model decay
- So far you've seen that model decay is a problem, but how do you detect it? Well, I'm glad you asked. Let's talk about that now.
  - Detecting drift, whether it's data drift or concept drift or both, **starts with collecting current data**.
  - You should collect all of the data in the incoming prediction request to your model, along with the predictions that your model makes.

- If it's possible in your application, also collect the correct label or ground truth that your model should have predicted.
- This is also extremely valuable for retraining your model, but at a minimum, you should capture the prediction request data, which you can use to **detect data drift using unsupervised** statistical methods.

## Detecting Drift

- Detected by observing the statistical properties of logged data, model predictions, and possibly ground truth
- Deploy dashboards that plot statistical properties to observe how they change over time
- Use specialized libraries for detecting drift
  - TensorFlow Data Validation (TFDV)
  - Scikit-multiflow library
- The process is really straightforward. Once you're set up to continuously monitor and log your data, you employ tools which use **well-known statistical methods** to compare your current data with your previous training data.
- You also use **dashboards** to monitor for trends and seasonality over time.
- Essentially, you'll be working with time series data since you have an ordered data that is associated with a time component.
- You don't have to reinvent the wheel here, there are good tools and libraries available to help you do this kind of analysis.
- These include TensorFlow Data Validation or TFDV, and the **scikit-multiflow** library (now known as [River](#)).

## Continuous Evaluation and Labelling in Vertex Prediction



- **Vertex Prediction** offers continuous evaluation
- **Vertex Labelling Service** can be used to assign ground truth labels to prediction input for retraining
- Azure, AWS, and other cloud providers offer similar services

- Cloud providers, including Google, offer managed services such as Google's Vertex Prediction, that help you perform continuous evaluation of your prediction requests.
- Continuous evaluation regularly **sample's prediction input and output** from trained machine learning models that you've deployed to Vertex prediction.
- Vertex **data labeling service** then assigns **human reviewers** to provide ground truth labels to your prediction input, or alternatively, you can provide your own ground truth labels.
- The data labeling service compares your model's predictions with the ground truth labels to provide continual feedback on how well your model is performing over time.
- Azure, AWS, and other cloud providers provide similar services

## Ways to Mitigate Model Decay

### Mitigating Model Decay

#### When you've detected model decay:

- At the minimum operational and business stakeholders should be notified of the decay
  - Take steps to bring model back to acceptable performance
- Now you've detected drift, which has led to model decay, so what can you do about it?
  - First, the basics. When you detect model decay, you need to let others know about it.
  - That means **informing** your operational and business stakeholders about the situation, along with some idea about how severe you think the drift has become.
  - Then you work on bringing the model back to acceptable performance, which is what I'll discuss now.

### Steps in Mitigating Model Decay

- What if Drift is Detected?
  - If possible, determine the portion of your training set that is still correct
  - Keep the good data, discard the bad, and add new data - OR -
  - Discard data collected before a certain date and add new data - OR -
  - Create an entirely new training dataset from new data
- Now that you've detected drift, what can you do about it?
- Well, first, try to determine which data in your previous training data set is still valid by using unsupervised methods such as clustering or statistical methods that look at divergence.

- Many options exist, including Kullback-Leibler or KL divergence, Jensen-Shannon or JS divergence or the Kolmogorov-Smirnov or K-S test.
- This step is optional, but especially when you don't have a lot of new data, it can be important to try to keep as much of your old data as possible.
- Another option is to simply discard that part of your training data set that was collected before a certain date and add your new data.
- Or if you have enough newly labeled data then you can just create an entirely new data set.
- The choice between these options will probably be dictated by the realities of your application and your ability to collect new labeled data.

## Fine Tune, or Start Over?

- You can either continue training your model, fine tuning from the last checkpoint using new data - OR -
- Start over, reinitialize your model, and completely retrain it
- Either approach is valid, so it really depends on results
  - How much new labelled data do you have?
  - How far has it drifted?
  - Try both and compare
- Now that you have a new training data set, you've basically two choices for how to train your model, fine tuning or starting over.
- You can either continue training your model, fine tuning it from the last checkpoint using your new data, or start over by re-initializing your model and completely retraining it.
- Either approach is valid and the choice between these two options will largely be dictated by the amount of new data that you have, and how far the world has drifted since the last time you trained your model.
- Ideally, if you have enough new data, you should try both approaches and compare the results.

## Model Re-Training Policy

On-Demand	<ul style="list-style-type: none"> <li>• Manually re-train the model</li> </ul>
On a Schedule	<ul style="list-style-type: none"> <li>• New labelled data is available at a daily, weekly or monthly basis</li> </ul>
Availability of New Training Data	<ul style="list-style-type: none"> <li>• New data available on ad-hoc basis, when it is collected and available in source database</li> </ul>

- It's usually a good idea to establish policies around when you're going to retrain your model.
- There's no right or wrong answer here, so it will depend on what works in your particular situation.
- You could simply choose to retrain your model whenever it seems to be necessary.
- That includes situations where you detect a drift, but it also includes situations where you may **need to add or remove class labels or features**, for example.
- You could also always retrain your model according to a schedule, whether it needs it or not.
- **In practice, this is what many people do because it's simple to understand and in many domains it works fairly well.**
- It can, however, incur higher training and data gathering costs unnecessarily, or alternatively, it can allow for greater model decay that might be ideal depending on whether your schedule has your model training too often or not often enough.
- Finally, you might be limited by the availability of new training data. This is especially true in circumstances where labeling is slow and expensive.
- As a result, you may be forced to try to retain as much of your old training data as possible for as long as possible and avoid fully retraining your model.

## Automating Model Retraining

Model Performance  
Degradation

- Manually retrain the model

Data Drift

- When you notice significant changes in the data

- If you can automate the process of detecting the conditions which require model retraining, that's ideal.
- That includes being able to detect model performance degradation and triggering retraining, or when you detect significant data drift.
- In both cases, in order to automate retraining, you should have data gathered and labeled automatically using a separate process and only retrain when sufficient data is available.
- Ideally, you also have continuous training, integration and deployment setup as well, to make the process fully automated.
- For some domains where change is fast and frequent retraining is required, these automated processes become requirements instead of luxuries.

## Redesign Data Processing Steps and Model Architecture

- When model performance decays beyond an acceptable threshold you might have to consider redesigning your entire pipeline
  - Re-think feature engineering, feature selection
  - You may have to train your model from scratch
  - Investigate on alternative architectures
- 
- When your model decay is beyond an acceptable threshold, or when the meaning of the variable you are trying to predict deviates significantly, or you need to make changes like adding or removing features or class labels, you might have to redesign your data pre-processing steps and model architecture.
  - I like to think of this as an opportunity to make improvements.
  - You may have to rethink your feature engineering, feature selection, and so forth, in order to make your model work with current data, and retrain your model from scratch rather than applying fine tuning.
  - You might have to investigate other potential model architectures, which personally I find is a lot of fun.
  - The point here is that **no model lives forever**, and periodically, you need to go back to the drawing board and start over, applying what you've learned since the last time you updated your model.

### Responsible AI

## Responsible AI Practices

- Development of AI **creates new opportunities** to improve the lives of people around the world
    - Business, healthcare, education, etc.
  - But it also **raises new questions** about implementing responsible practices
    - Fairness, interpretability, privacy, and security
    - Far from solved, active areas of research and development
- 
- Now let's look into some of the emerging issues concerning responsible AI and what you as a developer can do to ensure that your models and applications are as responsible as possible.
  - The development of AI is creating new opportunities to improve the lives of people around the world from business to healthcare to education and beyond.



- But at the same time, it's also raising new questions about the best way to build fairness, interpretability, privacy and security into these systems.
- These questions are far from solved, and are extremely active areas of research and development.
- I encourage you to commit to following the development of this field and working to make sure that your models and applications are as responsible as you can make them.
- They will never be perfect, but there is already a lot that you can do with more tools and techniques being developed constantly.

## Human-Centered Design

Actual users' experience is essential

- Design your features with appropriate disclosures built-in
  - Consider augmentation and assistance
    - Offering multiple suggestions instead of one right answer
  - Model potential adverse feedback early in the design process
  - Engage with a diverse set of users and use-case scenarios
- 
- The way actual users experience your system is essential to assessing the true impact of its predictions, recommendations and decisions.
  - For example, you should design your features with appropriate disclosures built in
  - Clarity and control is crucial to a good user experience.
  - Often, it's also a good idea to consider augmentation and assistance, producing a single answer can be appropriate where there is a high probability that the answer satisfies a diversity of users and use cases, but in other cases it may be better for your system to suggest a few options to the user.
  - In fact, it can even be easier since it's often much more difficult to achieve good precision at one answer top one versus precision at a few answers like top three.
  - Try to plan for modeling potential adverse feedback early in the design process, followed by specific live testing and iteration for a small fraction of traffic before full deployment.
  - And finally, engage with a diverse set of users and different use case scenarios and incorporate that feedback both before and throughout your project development.
  - This will build a rich variety of user's perspectives into the project and increase the number of people who benefit from the technology, and help you catch potential issues early.

# Identify Multiple Metrics

- Using several metrics helps you understand the tradeoffs
    - Feedback from user surveys
    - Quantities that track overall system performance
    - False positive and false negative sliced across subgroups
  - Metrics must be appropriate for the context and goals of your system
- A fairly simple technique is to use several metrics rather than a single one, which can help you understand tradeoffs between different kinds of errors and experiences.
  - Consider metrics including feedback from user surveys, quantities that track overall system performance, and short- and long-term product health (for example, click through rate and customer lifetime value respectively), and false positive and false negative rates **sliced across different subgroups**.
  - Of course, the metrics that you select are important, you should try to ensure that your metrics are appropriate for the context and goals of your system.
  - For example, **a fire alarm system should have high recall** even if that means the occasional false alarm.

# Analyze your raw data carefully

- For sensitive raw data, respect privacy
    - Compute aggregate, anonymized summaries
  - Does your data reflect your users?
    - Example: will be used for all ages, but all data from senior citizens
  - Imperfect proxy labels?
    - Relationship between the labels and actual targets
    - Using label X as a proxy for target Y - any problematic gaps?
- Of course, as always it all comes back to the data
  - ML models will reflect the data that they're trained on, so analyze your raw data carefully to ensure you understand it

- In cases where this is not possible, for example with sensitive raw data, understand your input data as much as possible while respecting privacy. For example, by computing aggregate anonymized summaries.
- Consider whether your data was **sampled in a way that represents your users**.
- For example, if your application will be used by people of all ages, but you only have training data from senior citizens, it might not work that well for other age groups.
- Imagine doing music recommendations when all of your data is from senior citizens. My guess is that it might not perform that well for teens
- Sometimes you're using your model to predict a proxy label for the actual target that you're interested in, because labelling for the actual target is difficult or impossible.
- In these cases consider the **relationship between the data labels that you have and the actual thing that you're trying to predict**.
- Are there problematic gaps? For example, if you're using data label X as a proxy to predict target Y, in which case is the gap between X and Y problematic?

## Legal Requirements for Secure and Private AI

### Legal Implications of Data Security and Privacy

Companies must comply with data privacy protection laws in regions where they operate

General Data Protection Regulation (GDPR)

California Consumer Privacy Act (CCPA)

- A legal side to practicing responsible AI. They are already legal requirements in some countries and regions, and this trend is growing.
- Exposure to civil liability is another concern. Let's explore some of the issues now.
- Training data, prediction requests, or both, can contain very sensitive information about people.
- For prediction request, those people are your users. Privacy of sensitive data should be protected.
- This includes not only respecting the legal and regulatory requirements, but also considering social norms and typical individual expectations.
- What **safeguards do you need to put in place to ensure the privacy of individuals** considering that ML models may remember or reveal aspects of the data that they've been exposed to?
- What steps are needed to ensure users have adequate transparency and control of their data?
- It's not just up to you to decide what is required. In Europe, for example, you need to comply with the General Data Protection Regulations, or **GDPR**, and in California, you need to comply with the California Consumer Privacy Act, or **CCPA**.

# General Data Protection Regulation (GDPR)

- Regulation in EU law on data protection and privacy in the European Union (EU) and the European Economic Area (EEA)
  - Give control to individuals over their data
  - Companies should protect the data of employees and consumers
  - When the data processing is based on consent, the data subject has the right to revoke their consent at any time
- The General Data Protection Regulation, or GDPR, was enacted by the EU in 2016 and became a model for many national laws outside the EU, including Chile, Japan, Brazil, South Korea, Argentina, and Kenya.
  - It regulates the data protection and privacy in the European Union and the European Economic Area.
  - The GDPR **gives individuals control over their personal data** and requires that companies should protect the data of employees and consumers.
  - When data processing is based on consent, the data subject, usually an individual person, has the **right to revoke their consent** at any time.



## California Consumer Privacy Act (CCPA)

- Similar to GDPR
  - Intended to enhance privacy rights and consumer protection for residents of California
  - User has the right to know what personal data is being collected about them, whether the personal data is sold or disclosed, and to whom
  - User can access the personal data, block the sale of their data, and request a business to delete their data
- In California, Consumer Privacy Act, or CCPA, was **modeled after the GDPR** and has similar goals, including enhancing the privacy rights and consumer protections for residents of California.
  - It states that users have the right to know what personal data is being collected about them, including whether the personal data is sold or disclosed in some way, who supplied their data and who received their data.
  - Users can access the personal data which a company has for them, block the sale of their data, and request a business to delete their data.

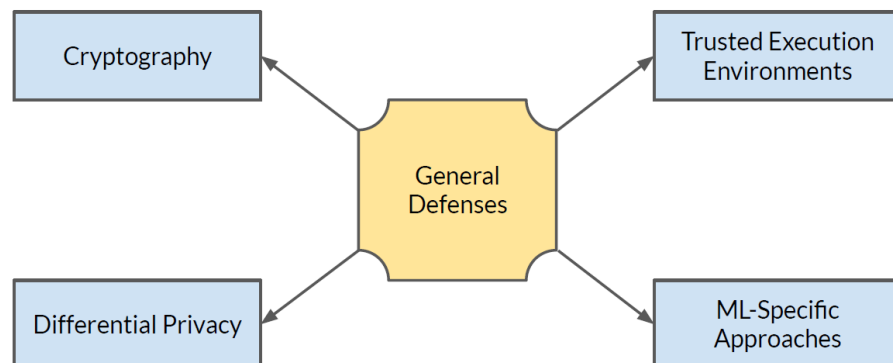


## Security and Privacy Harms from ML Models

Informational Harms	Behavioural Harms
Relate to unintended or unanticipated leakage of information	Relate to manipulating the behavior of the model itself, impacting the predictions or outcomes of the model

- Security and privacy are closely linked for some problems or harms in machine learning.
- Informational harms are caused when information is allowed to **leak** from the model.
- There are at least three different types of **informational harms**, including **membership inference**, (where an attacker can determine whether or not an individual's data was included in the training set), **model inversion** (where the attackers actually able to recreate the training set), and **model extraction** (where an attacker is able to recreate the model itself).
- **Behavioral harms** are caused when an attacker is able to change the behavior of the model itself.
- This includes **poisoning attacks** (where the attacker is able to insert malicious data into the training set), and **evasion attacks** (where the attacker makes small changes to prediction requests to cause the model to make bad predictions with adversarial examples).

## Defenses



- It's important to defend your model against attacks, as well as ensuring privacy and security of user data.
- Let's discuss a few approaches for defending against attacks.

# Cryptography

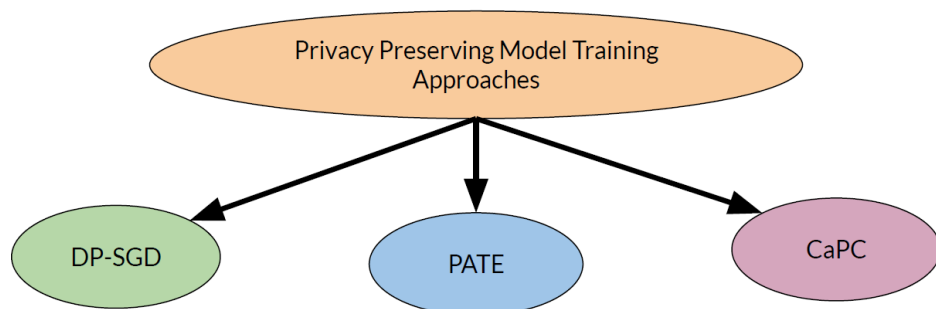
- Privacy-enhancing tools (like SMPC and FHE) should be considered to securely train supervised machine learning models
- Users can send encrypted prediction requests while preserving the confidentiality of the model
- Protects confidentiality of the training data



- You should consider privacy enhancing technologies such as Secure Multi-Party Computation, or SMPC, or Fully Homomorphic Encryption, or FHE, when training and serving your models.
- Briefly, SMPC enables multiple systems to collaborate securely to train and/or serve a model while **keeping the actual data secure through the use of shared secrets**.
- **FHE, on the other hand, enables developers to train their models on encrypted data without decrypting it first.**
- FHE in particular allows users to send an encrypted prediction requests and receive back an encrypted results.
- During the entire process, **the data is never decrypted except by the user.**
- However, you should be aware that currently, FHE is very computationally expensive.
- The goal here is that using cryptography, you can protect the confidentiality of your training data.

## Differential Privacy

System for publicly sharing information about a dataset by describing the patterns of groups within the dataset while withholding information about individuals in the dataset



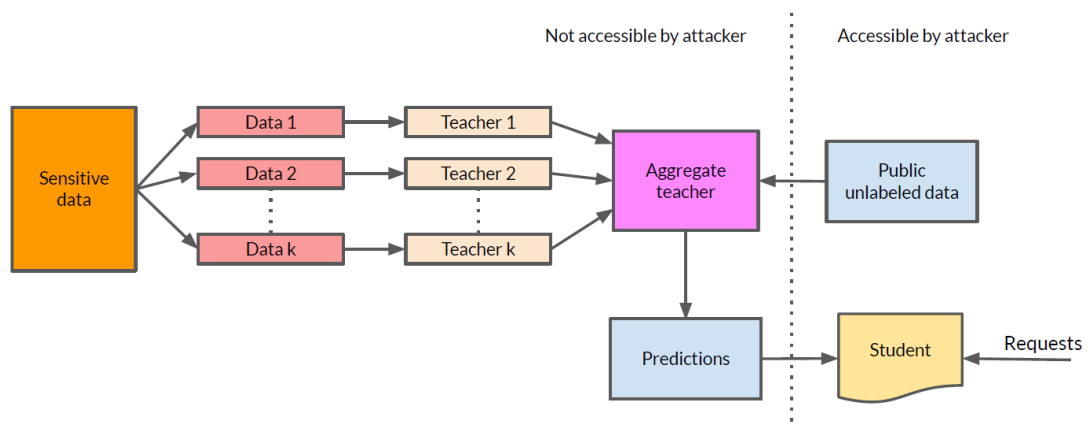
- Roughly, a model is **differentially private** if an attacker seeing its predictions cannot tell if a particular user's information was included in the training data.

- By implementing differential privacy, you can responsibly train models on private data.
- It provides provable guarantees of privacy, mitigating the risk of exposing sensitive training data.
- Let's briefly discuss three different approaches to implementing differential privacy - Differentially-Private Stochastic Gradient Descent, or DP-SGD, Private Aggregation of Teacher Ensembles or PATE, and Confidential and Private Collaborative learning, or CaPC.

## Differentially-Private Stochastic Gradient Descent (DP-SGD)

- Applies differential privacy during model training
  - Modifies the minibatch stochastic optimization process
  - Trained model retains differential privacy because of the post-processing immunity property of differential privacy
- If an attacker is able to get a copy of a normally trained model, then they can use the weights to extract private information.
  - Differentially-Private Stochastic Gradient Descent, or DP-SGD, eliminates that possibility by applying differential privacy throughout training.
  - It does that by modifying the minibatch stochastic optimization process by **adding noise**.
  - The result is a trained model which retains differential privacy because of the post-processing immunity property of differential privacy.
  - **Post-processing immunity** is a fundamental property of differential privacy. It means that regardless of how you process the models predictions, you can't affect their privacy guarantees.

## Private Aggregation of Teacher Ensembles (PATE)



- Next, let's take a look at Private Aggregation of Teacher Ensembles, or PATE.
- PATE begins by dividing up sensitive data into **k partitions** with no overlaps.
- It then trains k models on that data separately as teacher models, and then aggregates the results in an aggregate teacher model.
- **This is the same teacher-student used for knowledge distillation.**
- During the aggregation for the aggregate teacher, you will **add noise** to the output in a way that won't affect the resulting predictions.

- All of these models and the sensitive data are not available to end users, including attackers.
- **For deployment, you will create a student model.**
- To train the student model, you'll take unlabeled public data and feed it to the aggregate teacher model.
- The output of this process is labeled data, which maintains privacy.
- You use this data as the training set for the student model.
- After training, you will discard everything on the left side of this diagram and **deploy only the student model** for use.

## Confidential and Private Collaborative Learning (CaPC)

- Enables models to collaborate while preserving the privacy of the underlying data
  - Integrates building blocks from cryptography and differential privacy to provide confidential and private collaborative learning
  - Encrypts prediction requests using Homomorphic Encryption (HE)
  - Uses PATE to add noise to predictions for voting
- Confidential and Private Collaborative learning, or CaPC, enables multiple developers using different data to collaborate to improve their model accuracy without sharing information.
  - This preserves both privacy and confidentiality.
  - To do that, it applies techniques and principles from both cryptography and differential privacy.
  - This includes using Homomorphic Encryption, or HE, to encrypt the prediction requests that each collaborating model receives so that information in the prediction request is not leaked.
  - It then uses PATE to add noise to the predictions from each of the collaborating models and uses voting to arrive at a final prediction, again, without leaking information.
  - A great example of how CaPC can be used is to consider a group of hospitals who want to collaborate to improve each other's models and predictions.
  - Because of healthcare privacy laws, they can't share information directly.
  - But using CaPC, they can achieve better results while preserving the privacy and confidentiality of their patients.

## Anonymization and Pseudonymization



# Data Anonymization in GDPR

- GDPR includes many regulations to preserve privacy of user data
- Since introduction of GDPR, two terms have been discussed widely

Anonymization

Pseudonymisation

- Anonymization and pseudonymisation are some of the most well-established ways of protecting privacy. So let's discuss them now
- The GDPR includes many regulations to preserve privacy of user data and includes the definitions of many of the terms that it uses.
- This includes two terms that I'll discuss now – anonymization and pseudonymization.

## Data Anonymization

### Recital 26 of GDPR defines Data Anonymization

True data anonymization is:

- Irreversible
- Done in such a way that it is impossible to identify the person
- Impossible to derive insights or discrete information, even by the party responsible for anonymization

GDPR does not apply to data that has been anonymized

- Anonymization removes personally identifiable information or PII from datasets, so that people who the data describes remain anonymous
- GDPR Recital 26 defines acceptable data anonymization to be **irreversible** and done in such a way that is **impossible to identify** the person.
- It's impossible to derive insights or discrete information even by the party responsible for anonymization.
- **Once data has been acceptably anonymized, the GDPR no longer applies to that data.**

## Pseudonymisation

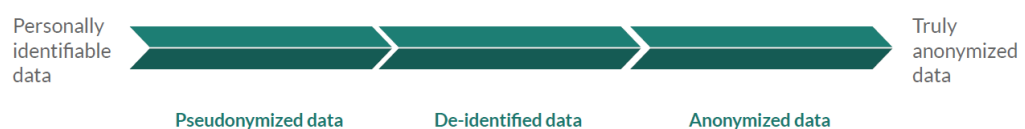
- GDPR Article 4(5) defines pseudonymisation as:  
“... the processing of personal data in such a way that the data can no longer be attributed to a specific data subject without the use of additional information”
  - The data is anonymized by switching the identifiers (like email or name) with an alias or pseudonym
- Pseudonymization is a bit different. This is a **reversible** process, meaning that it's still possible to identify the individual if the right additional information is included.
  - Pseudonymization can be implemented with **data masking or encryption or tokenization**.
  - It relies on careful control of access to the additional identifying information.

## Pseudonymisation v Anonymization

Information	Pseudonymized	Anonymized
Chelsea	Puryfrn	*****
Kumar	Xhzne	*****
Zaed	Mnrq	*****
John	Wbua	*****
Doe	Qbr	*****
Alex	Nyrk	*****

- So, to be clear the biggest difference between anonymization and pseudonymization is that pseudonymized data **can be reversed** using an additional set of information or an encryption key, while **anonymization is irreversible**.

## Spectrum of Privacy Preservation



- A lot of methods, mechanisms and tools have been developed over the years that produce data with various levels of both anonymity and the capability of being identified.
- It ranges from personally identifiable to truly anonymous data
- Personally identifiable data contains name, address, phone, email etc.

- Data which is purely anonymous and in accordance with GDPR guidelines does not include personally identifiable information or PII, and cannot be connected to PII, even with additional information
- Pseudonymized and de-identified data form the intermediary category of the spectrum.
- They are indeed a way of preserving certain aspects of data privacy but not to the level of truly anonymous data.
- Note however, that the **difference between de-identified data and pseudonymized data is not well defined**, and many discussions will group them together as one thing.

## What Data Should be Anonymized?

- Any data that reveals the identity of a person, referred to as identifiers
  - Identifiers applies to any natural or legal person, living or dead, including their dependents, ascendants, and descendants
  - Included are other related persons, direct or through interaction
  - For example: Family names, patronyms, first names, maiden names, aliases, address, phone, bank account details, credit cards, IDs like SSN
- So, what part of your data should you anonymize? Basically, everything that is part of PII.
  - That includes any data that reveals the identity of a person, which are known as identifiers.
  - With the term identifiers, I mean any natural or legal person, living or dead, including their dependents, their ascendants and descendants.
  - This also includes other related persons who might be identifiable through either direct or indirect relationships.
  - For example, this includes features such as family names, patronyms, first names, maiden names, aliases, address phone, bank account details, credit cards, tax IEDs and so forth.

## Right to be Forgotten

### What is the Right to Be Forgotten?

“The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay”

- Recitals 65 and 66 and in Article 17 of the GDPR
- It turns out that you have a right to be forgotten. Let's discuss that now, shall we?
  - First, some clarification of terms. When the GDPR refers to a data **subject**, it means a person, and when it refers to a **controller**, it means a person or organization who has control over a dataset containing Personally Identifiable Information or PII.

- Now that we've got that out of the way, let's ask the question. When does a person have the right to be forgotten?
- Well, there's a fairly long list of reasons for why an individual has the right to have their personal data erased.
- Rather than trying to remember these, I encourage you to refer to the GDPR.
- The list includes: the personal data is no longer necessary for the purpose an organization originally collected it or processed it, or an organization is relying on individual's consent as the lawful basis for processing the data and that individual withdraws their consent, or an organization is relying on legitimate interests as its justification for processing an individual's data and the individual objects to this processing and there is no overriding legitimate interest for the organization to continue with the processing, or an organization is processing personal data for direct marketing purposes and the individual objects to this processing, or an organization processed an individual's personal data unlawfully, or an organization must erase personal data in order to comply with a legal ruling or obligation, or an organization has processed a child's personal data to offer them information society services (an information society might be a social network, for example).
- If any of those conditions are met, you must delete the person's data.
- In general, these are mostly common sense.
- However, **in some cases, an organization's right to process someone's data might override their right to be forgotten.**
- Here are the reasons cited in the GDPR that **override** the right to be forgotten.
- The data is being used to exercise the right of freedom of expression and information, or the data is being used to comply with a legal ruling or obligation, or the data is being used to perform a task that is being carried out in the public interest or when exercising an organization's official authority, or the data being processed is necessary for public health purposes and serves in the public interest, or the data being processed is necessary to perform a preventative or occupational medicine (this only applies when the data is being processed by a health professional who is subject to a legal obligation of professional secrecy), or the data represents important information that serves the public interest, scientific research, historical research, or statistical purposes where the eraser of the data would be likely to impair or halt progress towards the achievement that was the goal of the processing, or the data is being used for the establishment of a legal defense or in the exercise of other legal claims.
- Furthermore, an organization can request a reasonable fee or deny a request to erase personal data if the organization can justify that the request was unfounded or excessive.
- But, in general, you should **avoid overriding** an individual's right to be forgotten unless you strongly meet one of these conditions.
- **When in doubt, err on the side of privacy.**

## Right to Rectification

“The data subject shall have the right to obtain from the controller without undue delay the rectification of inaccurate personal data concerning him or her. Taking into account the purposes of the processing, the data subject shall have the right to have incomplete personal data completed, including by means of providing a supplementary statement.”

- Chapter 3, Art. 16 GDPR

- You also have the right to have your personal information corrected or rectified.
- This might be important in situations like your credit history, health history, or employment history.

## Other Rights of the Data Subject

Chapter 3 defines a number of other rights of the data subject, including:

- Art. 15 GDPR – Right of access by the data subject
  - Art. 18 GDPR – Right to restriction of processing
  - Art. 20 GDPR – Right to data portability
  - Art. 21 GDPR – Right to object
- The GDPR also defines a number of other rights which people or data subjects have.
  - These include the right of access by the data subject, the right to restriction of processing, the right to data portability, and the right to object.
  - As a general rule, it's **best to err on the side of privacy, and consider any personal information that you have in your data as sensitive**.
  - You should restrict access to it and keep it safe.
  - Above all, you should think of it as the property of the person whose information it is and honor their wishes.

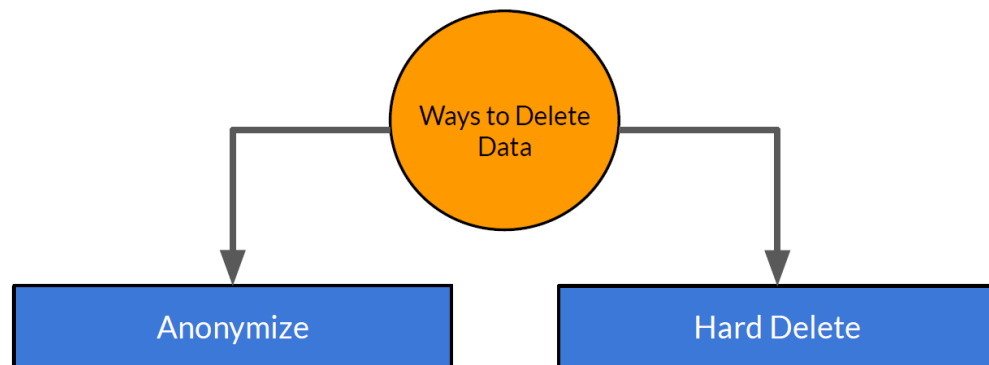
## Implementing Right To Be Forgotten: Tracking Data

For a valid erasure claim

- Company needs to identify all of the information related to the content requested to be removed
- All of the associated metadata must also be erased
  - Eg., Derived data, logs etc.

- When you receive a valid request to have personal information deleted, you need to identify all of the information related to the content requested to be removed
- You also need to identify and remove all of the metadata associated with that person.
- **If you've run any analysis or trained any models**, the derived data and logs, and models must also be removed or corrected.
- The goal here is **as much as possible to make it as if you never had their data**.

## Forgetting Digital Memories



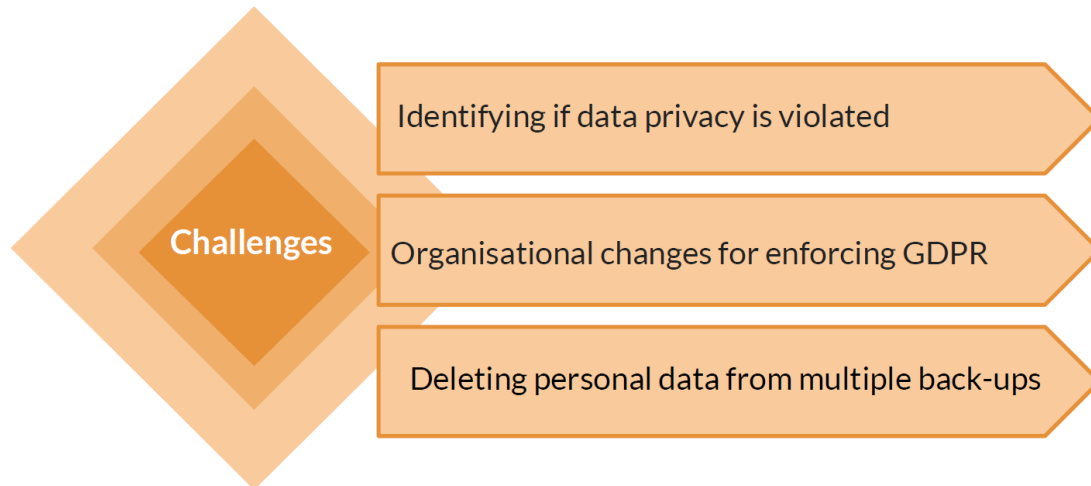
- There are basically two ways to delete data which will satisfy the requirements of the GDPR
- First, you can anonymize the data, which as you saw previously, will make it **non personally identifiable** under the terms of the GDPR, and the GDPR will no longer apply to anonymized data.
- Second, you can do a hard delete of the data, meaning actually delete the data, including any rows in your database which might contain it.
- **Normally, your first impulse might be to always just do a hard delete, but often there are issues with that.** Therefore, anonymization is another option.

## Issues with Hard Delete

- Deleting records from a database can cause havoc
  - User data is often referenced in multiple tables
  - Deletion breaks the connections, which can be difficult in large, complex databases
  - Can break foreign keys
  - Anonymization keeps the records, and only anonymizes the fields containing PII
- In a database or any other similar relational datastore, deleting records can cause havoc.

- **Part of this is because user data is often referenced in multiple tables, so deleting those records breaks the connections, which can be difficult, especially in large complex databases.** For example, it can **break foreign keys**.
- On the other hand, anonymization keeps the records and only anonymizes the fields containing PII while still satisfying the requirements of the GDPR.

## Challenges in Implementing Right to Be Forgotten



- There are several challenges in implementing the right to be forgotten
- The process of identifying whether or not data privacy has been violated is itself a challenging task,
- In order to enforce the GDPR, several organizational changes are needed, including **policy changes** and **training employees** in how to enforce the right to be forgotten
- And one last consideration, that can be tricky; if your organization maintains multiple backups of your data, which actually you should, **making sure that your personal data has been deleted from all of your backups is challenging.**
- You might very well have to change your data storage and backup implementation to maintain compliance with a GDPR.
- Issues like GDPR and the right to be forgotten are already important to operating in a business environment.
- Understanding the machine learning issues around them will only get increasingly important.
- We've given you a basic understanding of today's reality in this area, but I strongly encourage you to keep watching for new developments.
- **Also, I think it's always important to respect the privacy of your customers and treat any information or PII that you have with great care, and I strongly encourage you to do so.**

## References

- [Hidden Technical Debt in Machine Learning Systems](#)
- [Monitoring Machine Learning Models in Production](#)
- [Google Cloud Monitoring](#)
- [Amazon CloudWatch](#)
- [Azure Monitor](#)

- [Dapper](#)
- [Jaeger](#)
- [Zipkin](#)
- [Vertex Prediction](#)
- [Vertex Labelling Service](#)
- [Retraining Model During Deployment: Continuous Training and Continuous Testing](#)
- [How “Anonymous” is Anonymized Data?](#)
- [Pseudonymization](#)
- [Responsible AI Practices](#)