**MLOPs (Machine Learning Operations)** is a set of practices that combines machine learning, DevOps, and data engineering to automate and manage the entire ML lifecycle — from data preparation and model training to deployment, monitoring, and updating. It ensures models are production-ready, scalable, reproducible, and continuously improving, enabling faster and more reliable delivery of ML solutions.

Below are the problems MLOPS solves, or these are the aspects of the MLOPS.

**1. Data Management**

a. Data Collection

b. Data Pre-processing

c. Data Validation

d. Data Security

e. Data Compliance

f. Feature Store

**2. Development Practices**: Modular coding, developing the code modules for different parts and then combining.

**3. Version Control:**

- Data versioning

- Code Versioning

- Model versioning

**4. Experimental Tracking:**

- Tracking Experiments

- Test and validation

- Model registry

**5. Model Serving and CI/CD:**

- Continuous integration

- Containerization

- Continuous deployment

**6. Automation:**

- **Pipeline automation** [Data ingestion pipeline, model training pipeline, model validation and testing, model deployment, model monitoring and retraining].

- **Orchestration:** coordinating and automating the various steps involved in the ML lifecycle.

**7. Monitoring and Retraining:**

- Monitoring the metrics of the deployed model.

- Drift Detection

- Retraining

**8. Infrastructure Management:**

- Cloud based solutions to handle scalability concerns

- Cost management

- Managing multiple vendors

**9. Collaboration and operations:**

- Unified workspace

- Role based access

**10. Governance and Ethics**

MLOPs refers to the practice and discipline within machine learning that aims to unify and streamline the machine learning system development (Dev) and machine learning system operation (Ops). It involves collaboration between data scientists, ML engineers, and IT professionals to automate and optimize the end-to-end lifecycle of machine learning applications.

**Benefit of MLOPs:**

1. Scalability

2. Improved performance

3. Reproducibility

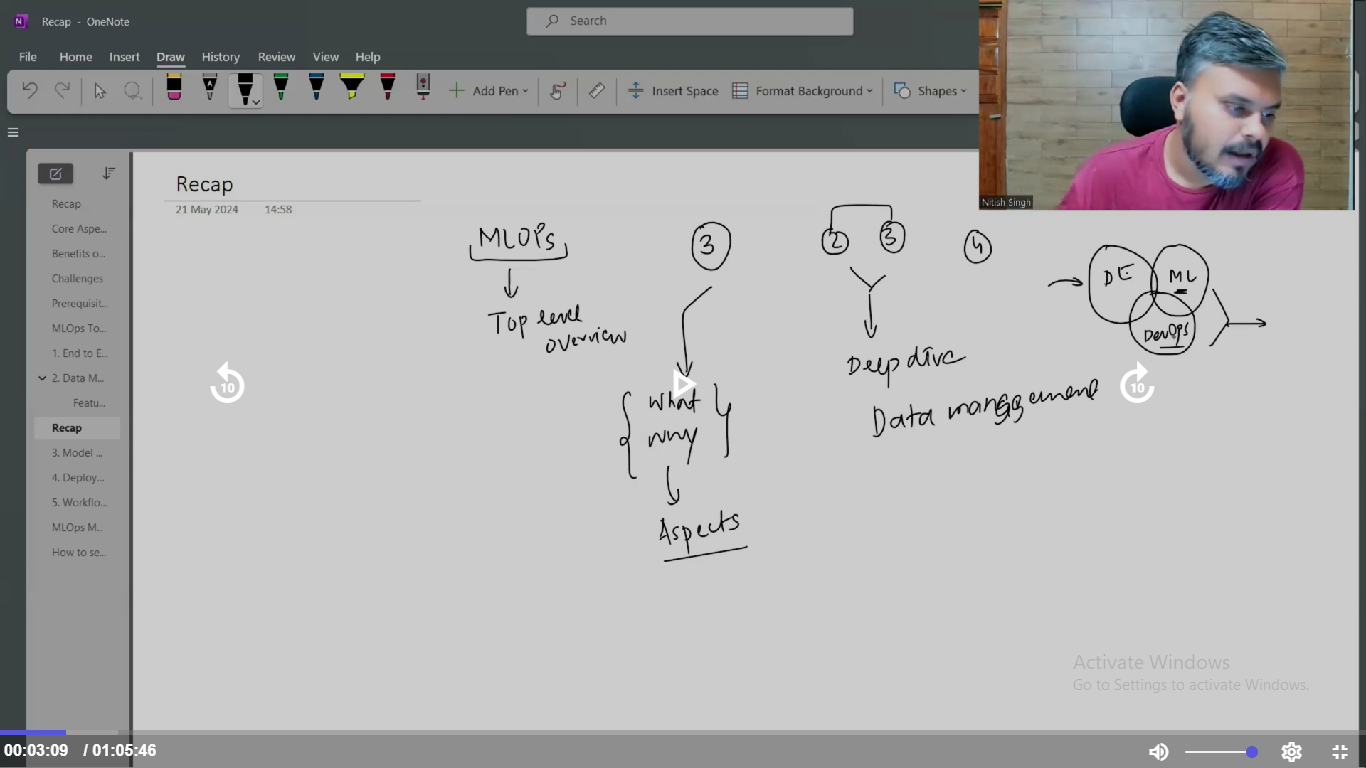
4. Collaboration and efficiency

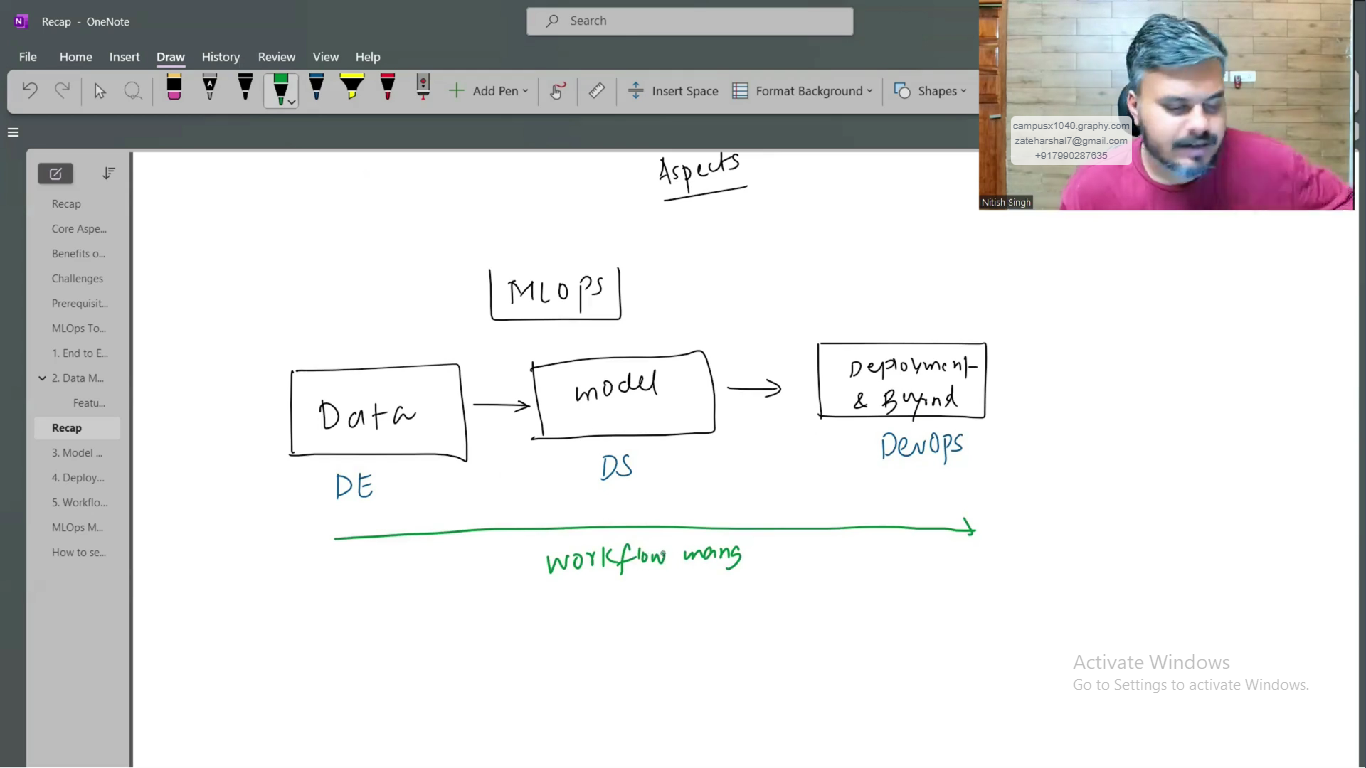
5. Risk reduction

6. Cost Savings

7. Faster time to market

8. Better compliance and governance





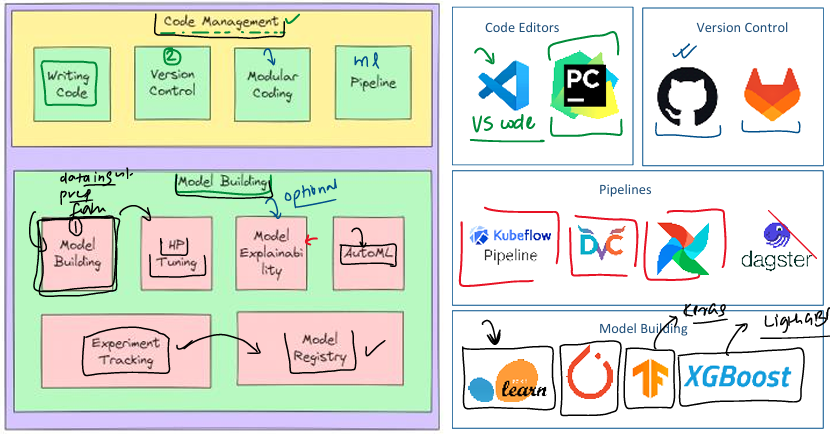
1. **Data Engineering**

**ETL (Extract, Transform, and Load)** is a data integration process used to move data from multiple sources into a centralized system, typically a data warehouse. It involves three key steps:

* **Extract**: Pulling raw data from various sources (e.g., databases, APIs, files)
* **Transform**: Cleaning, filtering, and converting the data into a desired format or structure
* **Load**: Storing the transformed data into a target system for analysis and reporting

Rest is to be refereed from the CampusX notes.

1. **Model Building:**



* **Modular coding** is a software development practice where a program is divided into small, self-contained units called modules, each responsible for a specific task. This approach enhances code **reusability**, **readability**, and **maintainability**, as modules can be developed, tested, and updated independently without affecting the entire system. It also promotes **scalability**, allows for **faster development** through parallel work, and improves **debugging and testing** by isolating functionality. By organizing code into logical, well-defined components, modular coding supports clean architecture, better collaboration among developers, and more robust, flexible software systems.
* **CookieCutter** supports modular coding by automating the creation of structured, modular project templates. It allows developers to scaffold projects with predefined folders and files for separate modules (e.g., core/, utils/, api/), ensuring consistent architecture and separation of concerns. This promotes reusability, easier maintenance, and faster development by eliminating repetitive setup tasks. By enforcing a standard modular layout across projects and teams, CookieCutter helps streamline development while aligning with clean, maintainable coding practices.
* **Pipelines:** A **machine learning pipeline** is a structured and automated sequence of steps that takes raw data through various stages—like pre-processing, training, evaluation, and deployment—to produce a final model. Instead of running each step manually, the pipeline handles everything in the correct order, ensuring consistency and saving time. It tracks inputs, outputs, and changes, so only the necessary steps are re-executed when something is updated. This makes the workflow **reproducible**, **efficient**, and easy to **collaborate** on, helping teams build and manage ML models in a reliable, scalable way. Tools use are Kubeflow, DVC, dagstar
* ML Model development: Everything is known, and stuff is similar.
* Hyper parameter tuning: Process is similar, tools are Optuna and HyperOpt
* **Model Explainability:** is a set of techniques that make AI model decisions **understandable and transparent** to humans. It helps users see **why** a model made a prediction, **which features influenced it**, and ensures **trust, accountability**, and **fairness**, especially in critical fields like healthcare, finance, and law.

Many powerful AI models (like deep learning) are "black boxes" — they make accurate predictions but **don’t clearly show how or why** they reached a conclusion. XAI helps answer questions like:

* Why did the model make this prediction?
* What features influenced the decision?
* Can I trust this output?

Tools like SHAPLEY, Lime are used.

* **AutoML (Automated Machine Learning)** is the process of automating the end-to-end steps of building machine learning models — including **data preprocessing**, **feature selection**, **model selection**, **training**, and **hyperparameter tuning** — with minimal human intervention. It helps both experts and non-experts quickly build high-performing models, saving time and reducing manual effort while ensuring consistency and scalability. Tools, Cloub AutoML (google), PYCARET, Katib, Auto Sklearn etc.
* **Experiment tracking** in machine learning is the process of logging and managing all details of model training runs — such as datasets used, code version, hyperparameters, metrics, and results. It helps ensure **reproducibility**, **easy comparison**, and **collaboration** by keeping a clear record of what was tried and what worked best. Tools like **MLflow** or **Weights & Biases** automate this process, making model development more organized and efficient.
* **Model registry** in MLOPs is a centralized repository or system that manages the storage, versioning, and lifecycle of machine learning models. It serves as a catalogue where models are tracked from development to deployment, ensuring that the best-performing models are readily available for production. The model registry is essential for managing the complexity of deploying and maintaining models in a reliable and reproducible manner.

1. **Model Versioning:** Track different versions of a model as it evolves over time.
2. **Metadata Management:** Store and manage metadata associated with each model.
3. **Model Lineage Tracking**: Track the lineage and provenance of models to understand their development history.
4. **Stage Management:** Manage the lifecycle stages of a model (e.g., development, staging, production).
5. **Model Storage and Retrieval:** Store and provide access to model artifacts, It means the model registry **saves all the files needed to use or redeploy the model,** and makes them **easily accessible** to others or to automated systems.
6. **Devops:**

A virtual machine is an **isolated computing environment** created on a single physical host, where each VM operates independently and securely from others—even when sharing the same underlying hardware. This isolation ensures that the software, operating system, and data within one VM cannot interfere with or access those of another VM, maintaining performance, security, and stability

For next, ref notes.

**Actual Coding start:**

1. Code modularity and DVC pipeline creation.

DVC: Complete the entire notes in detail.

**DVC:** DVC (Data Version Control) is an open-source tool that helps manage and version control large datasets, machine learning models, and experiments alongside code by working with Git. It replaces large files with small metafiles tracked by Git, while storing actual data in remote storage like S3, GCP, or shared drives. DVC uses files like dvc.yaml to define the ML pipeline stages (commands, inputs, outputs), params.yaml to track hyperparameters, and .dvc files to manage individual data files or directories. These files store metadata and are versioned in Git, enabling reproducible workflows by allowing you to restore any stage of the project reliably and consistently.

**How does it work?**

* SRC folder in project folder, which stores all the modules, these modules will be connected by pipeline and how to connect this modules that info is kept in dvc.yaml.
* Each module is staged by **dvc stage add** where four flags are set –n name –d dependencies –p parameters –o output which determines how the pipelines is formed and connected.
* **Dvc repro** to reproduce or run pipeline.
* **Dvc dag** for connection visualization
* **Dvc remove** for removing the stage
* **Dvc metrics** show to see the metrics.

Current directories keep it on root project folder level make sure to specify the exact location of the module while creating a folder and specifying dependencies because everything will have project folder as a root folder.

Idea is you split the entire code into specific modules. Where output of each module is stored specifically, in case of dataset is the output of the module then that dataset is stored in a specific data directory. And the next module will load the data from this location specifically and further processed the data and we have pipeline functioning properly with module connection.

**Improvements:** (a new file with the improvements is maintained)

1. **Params.yaml for parameter changes directly through the params.yaml file.**
2. **Implementing function in code.**
3. **Type hinting**
4. **Error handling**
5. **Logging**
6. **Cookie cutter for folder template.**

**Type hinting:** All you do is specify the argument and return’s data type.

e.g. def load\_data(**path : str) -> pd.DataFrame**:

pass

**Logging:** Logging in MLOPs refers to the systematic capture of information about every stage of the machine learning lifecycle—ranging from data ingestion and model training to deployment and serving—so that teams can monitor, debug, and trace both routine events and critical incidents. There elements are logger, handler, formatter etc. set these all three and then connect formatter to handler and then handler to logger. Meanwhile do set the levels of the logger and handler while creating the objects of these. Once a level is set at the start, only a level above can be tracked/called after.

**Version Control:**

After DVC pipeline creation and improvements in the pipelines by use of functions, exception handling, logging etc.

Now focus on Version Control, where CookieCutter directory template is used and each SRC file is pushed to the git repo while maintaining versions. (follow the video and steps docs to replicate)

**Data Version:** GIT can not version data due it’s inability to take data snapshots (it does it by doing the row by row compare) i.e. we use DVC for

# MLOPS (Andrew Ng)

Manages the lifecycle of the machine learning model. In entire lifecycle, model gets build in to far fewer lines compared to entire code of the ml lifecycle of the model.

Edge system: local system which processes the data locally, interact with deployed models and get the predicted response.

ML project lifecycle:

Scoping: Define project

Data: Define data and established baseline, label and organize

Modelling: Select and train model, perform error analysis

Deployment: Deploy in production, monitor and maintain system.

e.g. Voice recognition in Lifecycle: How does it works?

Few issues in data level step, first is the data correct labeling, volume normalization etc.

Issues in modeling, in modeling there are two approaches of improving models,

First is kind of research base approach to change the,

Change the code(algorithm/model)

Change the Hyperparameters

Data Constant

Whereas the second approach which is more product team approach is to

Keep the code/model/algorithm same

But change or tune the hyperparameters

And improve the data, (based on the data improve quality of the data to the model,)

**Data drift:** Data distribution changes, i.e. statistical properties of the input variable (features changes), with same relationship between X and Y.

e.g. A retail model trained on customer data from one region is deployed in another region with different age groups and income levels, causing the input distributions to shift, though the relationship between the input and output remained the same

**Concept Drift:** Relationship between the input variable (X) with label target variable (Y) changes.

e.g. in spam detection, suppose now the spammers start using different techniques meaning the relationship between the input changes, what was spam earlier is not spam now.

Software engineering issues:

* Realtime or Batch
* Cloud vs edge/browser
* Computer resources (CPU’s / GPU’s)
* Latency, throughput
* Logging
* Security and Privacy

**Deployment Patterns:**

Common deployment cases:

1. New product/capability: deploying a new product.
2. Automate/ assist with manual task: Add on to already deployed product/ running product, like smart phone monitoring product.
3. Replace previous ML system: replacing existing ml system

Key Ideas:

* Gradual ramp up with monitoring: Have small traffic initially then monitor and ramp up the traffic gradually.

Rollback:

* Revert to the previous version if the current one gets you errors.

**Types of the deployment:**

* Shadow mode deployment: a new model is deployed but does not handle queries directly, existing model only process the queries, and new model is tested for the same queries. Facilitate the testing of the model on the live data before handling actual queries.
* Canary deployment: New version is released to small user first while other users still use the old one and then gradually if no errors and performance drop in new model then the new one is rolled out to more users.
* Blue green deployment: In Blue green deployment, blue as in old model with green as in new model both maintained in two environments, only one is live at a time, and when testing is done the entire is switched entirely. Easy to rollback as both are maintained in separate environments.

**How to monitor ML production model:**

Dashboard metrics are monitored, now which metrics to monitor?

* Software metrics, like Memory, compute, latency, throughput, server load etc.
* Input metrics: certain metrics about the input data can be monitored which describes the statistics of the input data (like data proportionate). E.g. for speech recognition model input avg input length, avg input volume, num missing etc. can be checked with pre-decided thresholds
* Output thresholds: Output quality measuring metrics, like in case of the speech recognition null return, timer users redo the search (indicating the inefficiency in detecting the speech first time). This metrics are finalized as you run the model, initially you can start with lot of these metrics.
* Like model development, deployment is iterative process, you develop the model, deploys it, test it as the traffic gets on the model, do the performance analysis makes the modifications and its loop now. Retraining is either manual or automatic.

ML pipeline monitoring: entire project is based on the multiple steps which are combined in a pipeline, each step could be a ML algorithm itself where the input of one step is fed to the next step. So, pipeline could have multiple elements combined which could be ML elements or non-ml elements combined, it’s important to monitor the metrics at multiple stages as the next stage element is dependent on the first one.

Software metrics, drifts could be measured at each stage’s ML element.

Best practices for building for building the ML model:

* Model(code) centric approach
* Data Centric approach

Key challenges:

* Doing well on the train set.
* Doing well on test set.
* Doing well on business metrics/project goals.

Mainly the ML projects are driven by the test set accuracy, but there could be discrepancy between the test set and business logic. How?

**How low-test error is not good enough?**

* Fairness, bias and legal risk, if the model performs poorly on the specific groups like gender, age, ethnicity then in that case, business application won’t allow the model, though it would be giving the good result on the test set. (sort of like imbalanced set).
* Cost of mistake varies, like the mistake made on certain classes is far expensive than then other one (e.g. mistake on recommending wrong shirt is not that but approving fraudulent transaction is very high)

**How to establish the baseline model for the data? (structured and unstructured data)**

* Human level performance (comparing with human performance)
* Literature search (open-source model or work indicating the performance)
* Performance on older system
* Quick and implementation (like Harshal builds the quick baseline model and then start improving the model).

Tips for getting started.

Good data is better than good algorithm. Always takes deployment/compute constraint into account while developing the model.

Sanity check: before start training a model, check with very few datapoints and try to overfit the model with very smaller data and see if the accuracy is good or not, this gives you the idea whether is it worth going deeper and worth taking the efforts to train and optimized this model on this data.

**Error analysis example:**

Analyzing the error, error patterns, quantify, understand the errors made. E.g. for speech recognition is,

For the misclassified speech texts, creates the different features stating the reasons of the noise, like car noise, human noise, low bandwidth and mark for each of the text, the columns don’t have to be mutually exclusive. This indicates the speech with its unique reason of error and help you understand which kind of error is what and which one to prioritize and how? the column can be added as you move in the data. This column I’m referring are tags,

So, this error analysis or tagging process is the iterative process like the other process like model building as well as the deployment process.

**Error Analysis for Skewed dataset (imbalanced):**

Use metrics like Precision, recall and F1 scores, F1 scores harmonic mean penalized the score far more for smaller value of any of the recall or F1 score.

**Performance Auditing:** even after the error analysis, one last time before deploying, check for fairness and bias,

1. Brainstorm the ways the system might go wrong.

* Performance on subset of data, (e.g. ethnicity, gender)
* How common are certain error.
* Performance on rare classes.

1. Established metrics to assess performance on appropriate **slices of the data**.
2. Get business/product owner buy-in.

Speech recognition example.

1. Brainstorm the ways the system might go wrong:

* Accuracy on different genders and ethnicities.
* Accuracy on different devices.
* Prevalence of the rude mis-transcriptions. (Like GAN could be interpreted as GUN or GANG, could mean the data would have lot of content related to gun violence and could be misleading.

1. Established metrics to assess performance against these issues on appropriate slices of data.

* Mean accuracy for different genders and major accents.
* Check for prevalence of offensive words in output.

Now just, imagine your error analysis has pushed you to improve your model on certain slice (category or tag etc.)

Model centric approach – have better model (most of the research development in the field of AI/ML is based on the same approach).

Data centric approach- Improves the quality of the model (Fixed code, change the quality of the data). Best approach to improve the quality of the data is by data augmentation.

**Conceptual picture of thinking about the data augmentation:**

Let’s take an example of the speech recognition.

Different types of the speech input noise:

* Car noise
* Plane noise
* Train noise
* Machine noise
* Caffe noise
* Library noise
* Food court noise

**Graphical intuition,** just imagine graph, with y-axis being the performance with X having different noise kinds, now suppose some noise kinds have lower performance on them and some have high, compare the performance with human level performance, so now we have two curves one indicating the actual model performance on the different noises and human level performance curve as well, the max height or difference between them gives you the highest chance or scope of the improvement, so you tend to improve that one, improve as in the data can be augmented for this noise, by improving the data for this noise does not reduces the performance on the far extreme/ good noises but does improves performances on the similar/nearby noise datapoints as well.

Just like a rubber strings, you tend to stretch the rubber string of the model towards the rubber string of the human level performance from the point where there is max scope of the improvement. By doing the nearby points/ noises also improves or gets stretches, nearby ones get stretched the most compared to the far ones.

**How to do data augmentation?**

Let’s take a similar voice detection example.

Voice signal + Caffe noise = synthetic training example

Voice signal + library noise = synthetic training example

Goal:

Create a realistic example, that algorithm does poorly on, but humans or another baseline does well on.

Checklist:

* Does it sound realistic?
* Is the X🡪 Y mapping clear (e.g. can the humans recognize speech?)
* Is the algorithm currently doing poorly on it?

e.g. for the scratch detection in phone manufacturing?

- ways to augment would be contrast/ color change, dark, light obviously not too dark not too light.

- Even GANs can be used.

You fit the data augmentation in the data iterative loop, i.e. you do augmentation then you train, error analysis and depending on the error analysis repeat the process.

**Can Adding data by data augmentation hurt?**

mostly not.

* If the model is large enough
* The mapping X🡪 is clear (given only the X humans can accurately predict).

Adding data rarely hurts.

**Structured data:** represented in the structured format like tables, databases etc.

Whereas unstructured data does not follow a structured format. E.g. Images, audios, NLP etc.