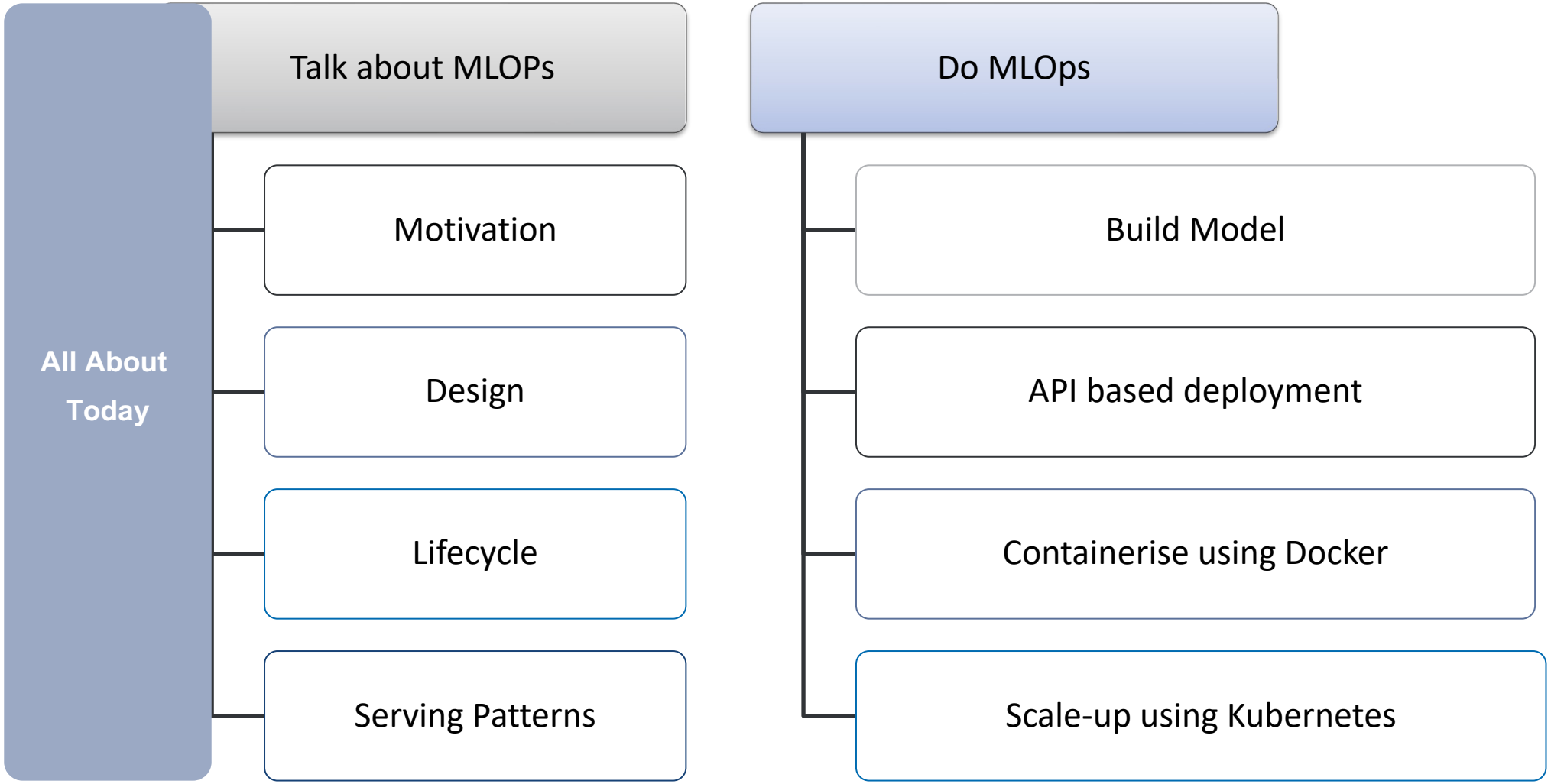
infogain
Engineering Business Outcomes

MLOps – ⚙ of life

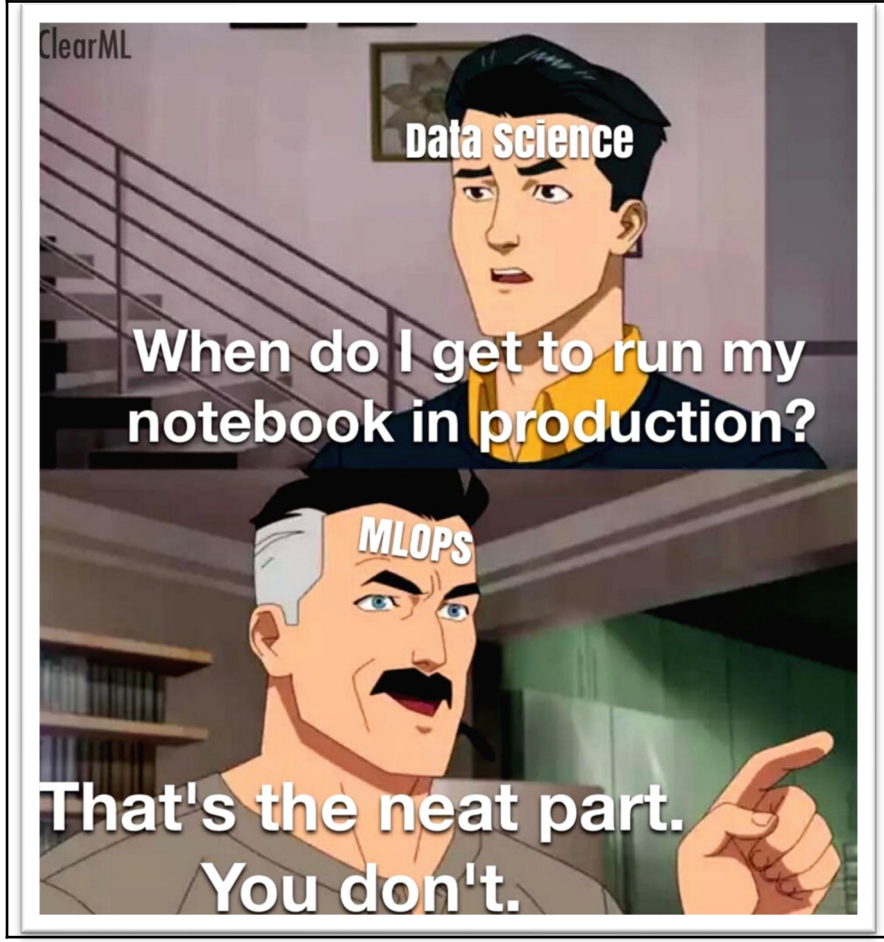
Sumit Tyagi



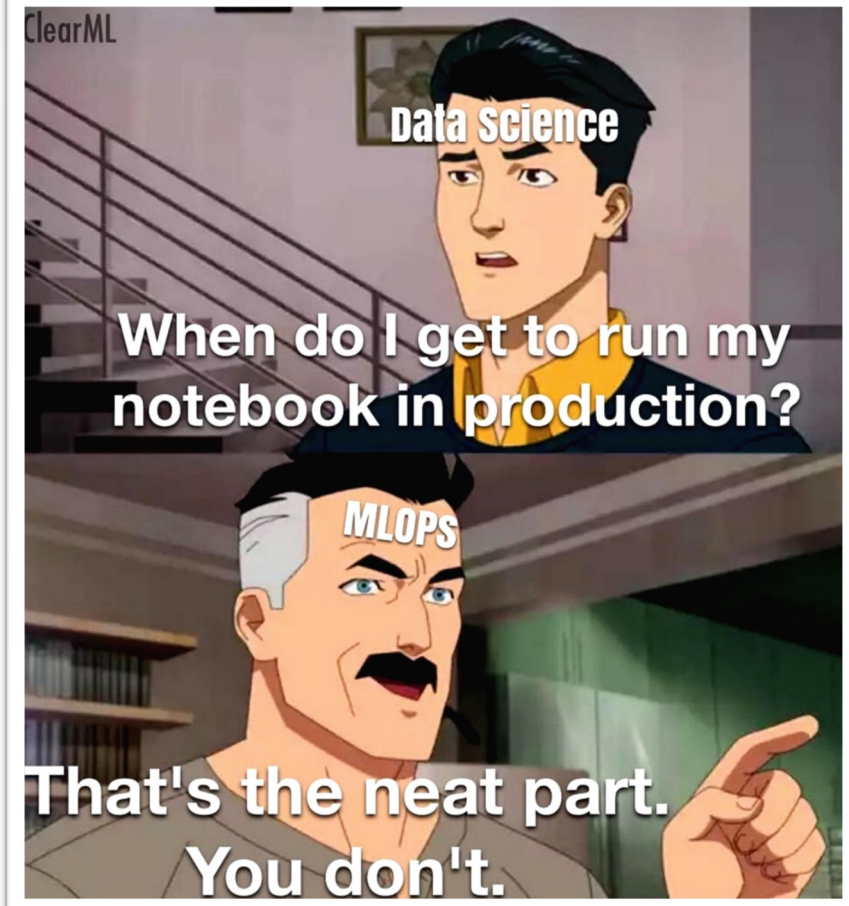
MLOPS

Motivation

Motivation

What is MLOps?	
For all my analyst/ consultant friends	
	

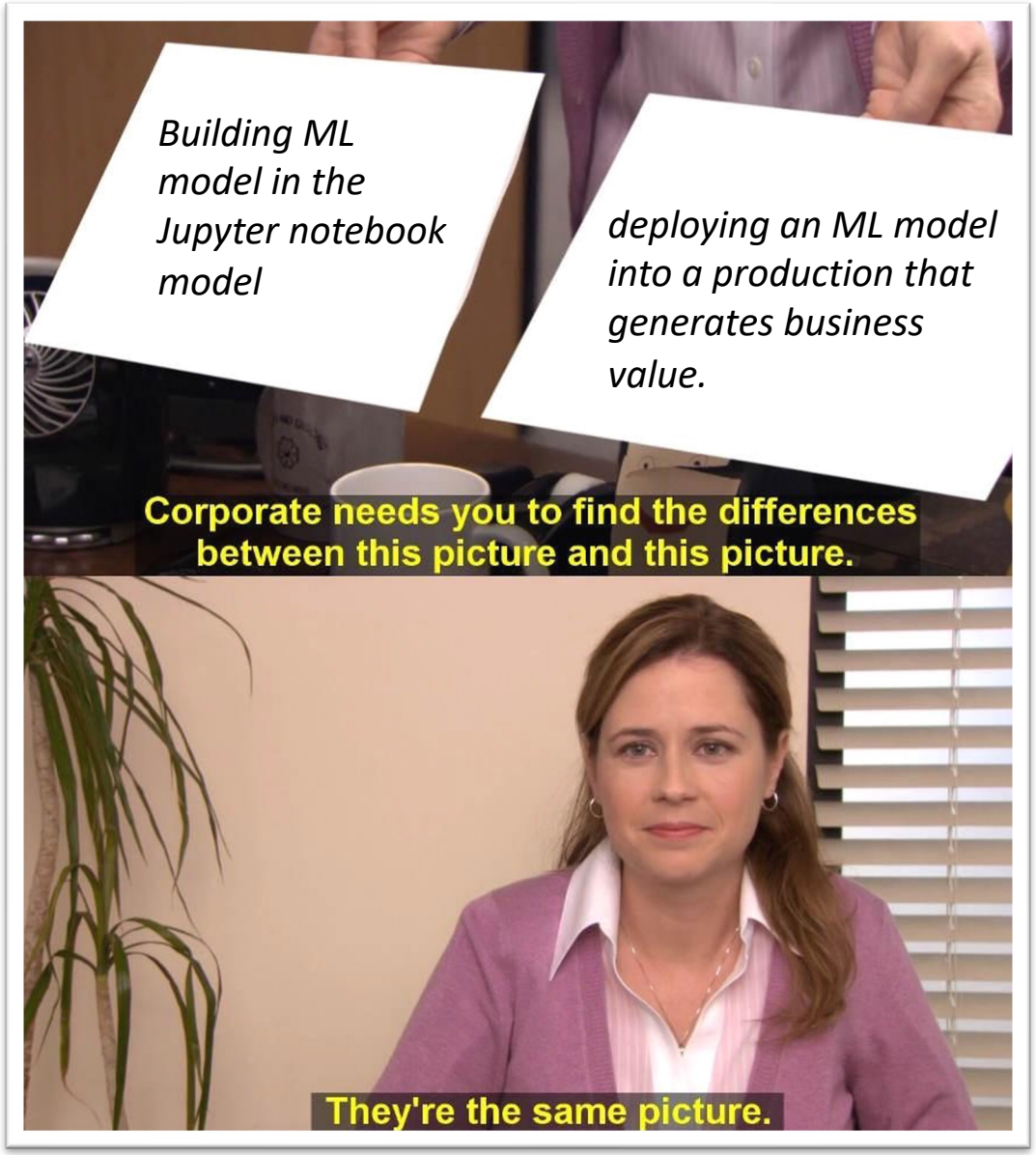
Motivation

What is MLOps?	
For all my analyst/ consultant friends	For all my PM friends
	

Motivation

Deployment Gap:

- Although AI budgets are on the rise, only 22 percent of companies that use machine learning have successfully deployed an ML model into production.



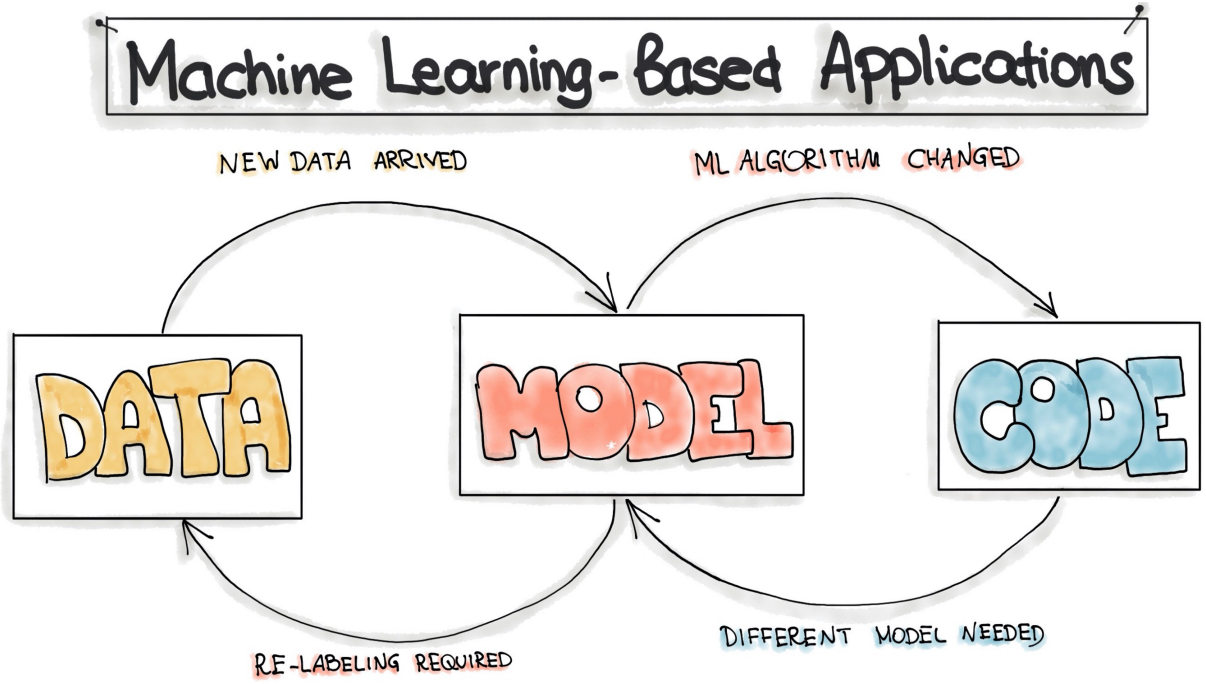
Motivation

Deployment Gap:

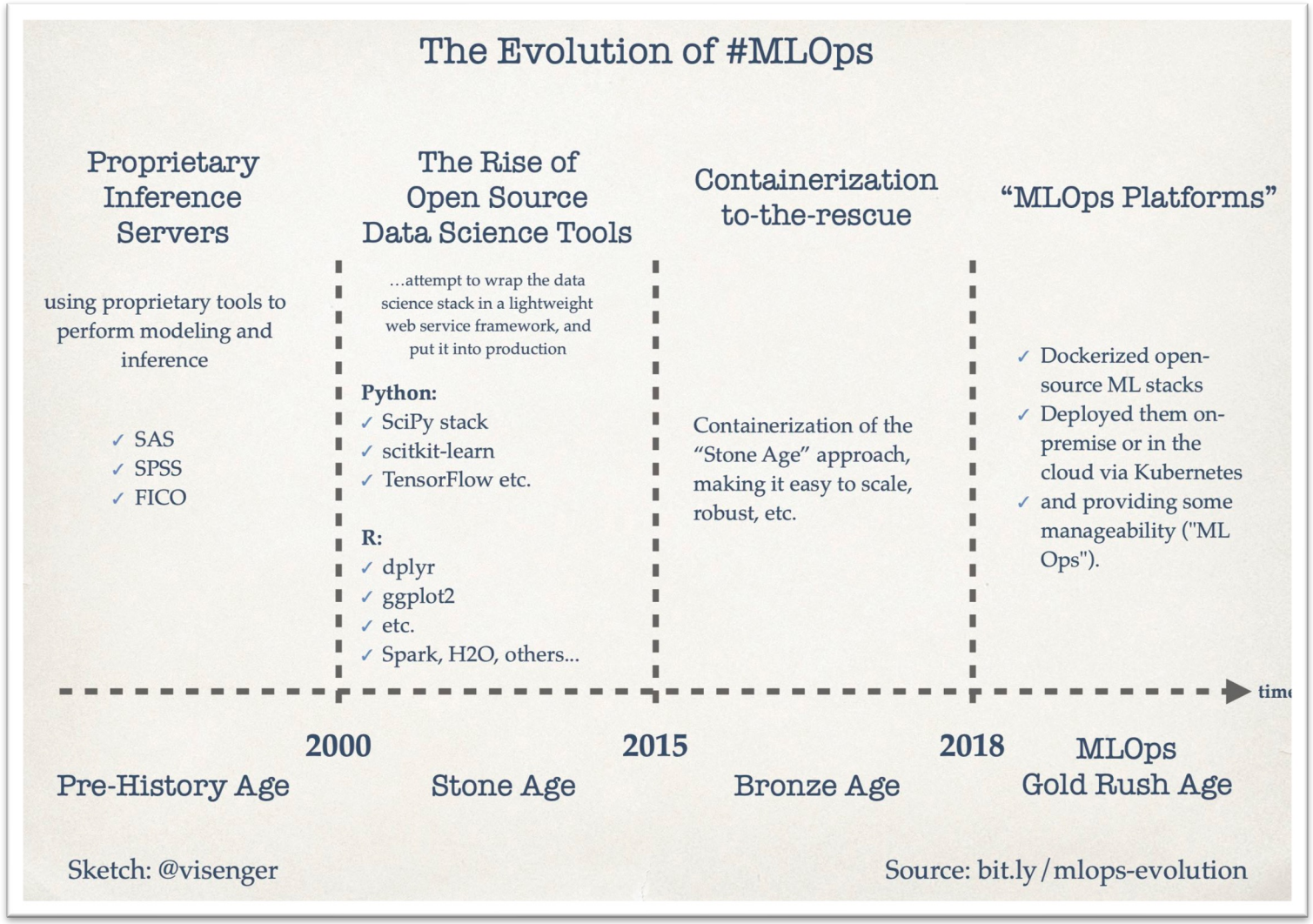
- Although AI budgets are on the rise, only 22 percent of companies that use machine learning have successfully deployed an ML model into production.

Scenarios that needs to be managed

- In machine learning-based systems, the trigger for a build might be the combination of a code change, data change, or model change.



Motivation



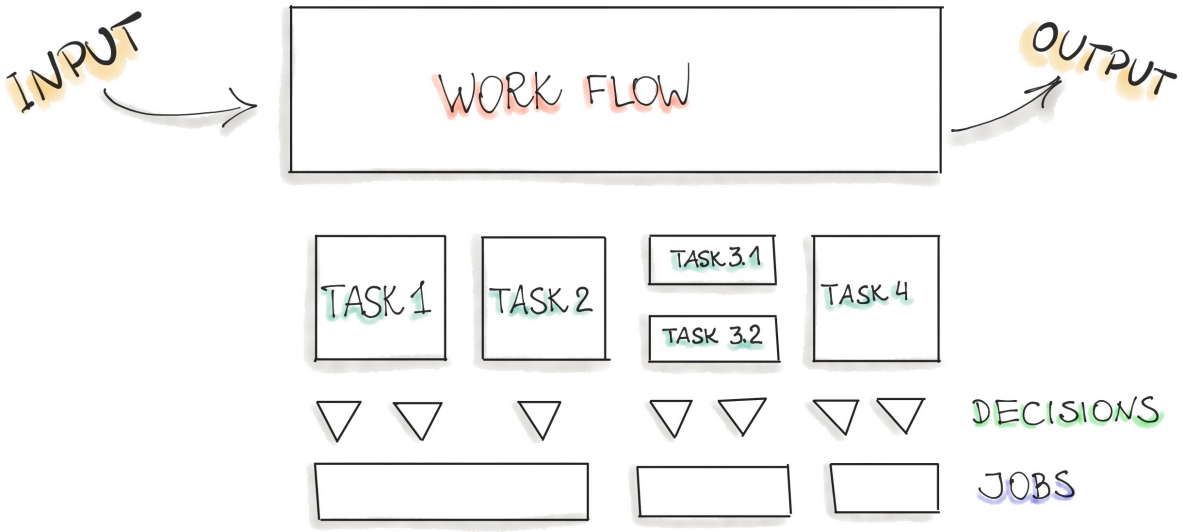
MLOPS

Design

Design a ML-powered software

To answer the question *"how to implement AI/ML"*, we follow the next steps:

1. Identify the concrete ***process*** that might be powered by AI/ML.
2. Decompose that process into a directed graph of ***tasks***.
3. Identify where humans can be removed from the task, meaning, what task can be replaced by a prediction element such as ML model?
4. Estimate the ROI for implementing an AI/ML tool to perform each task.
5. Rank-order the AI/ML implementation for each ***task*** in terms of ROI.
6. Start from the top of the list and structure the AI/ML implementation by *Machine Learning Canvas*.



MLOPS

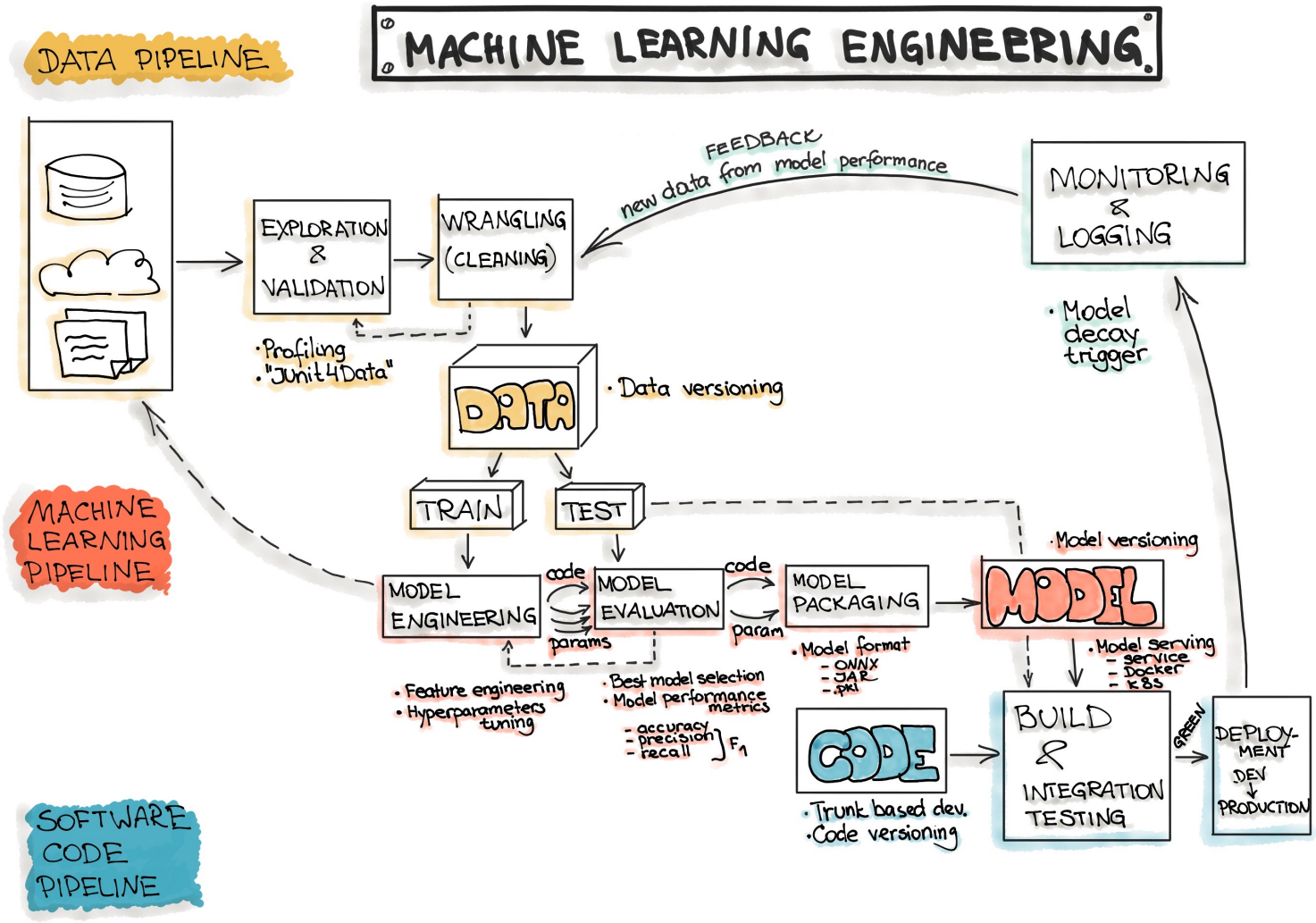
Life Cycle

Lifecycle

The Figure below shows the core steps involved in a typical ML workflow.

Based of three artifacts Data, ML Model, and Code, we have three ML workflows:

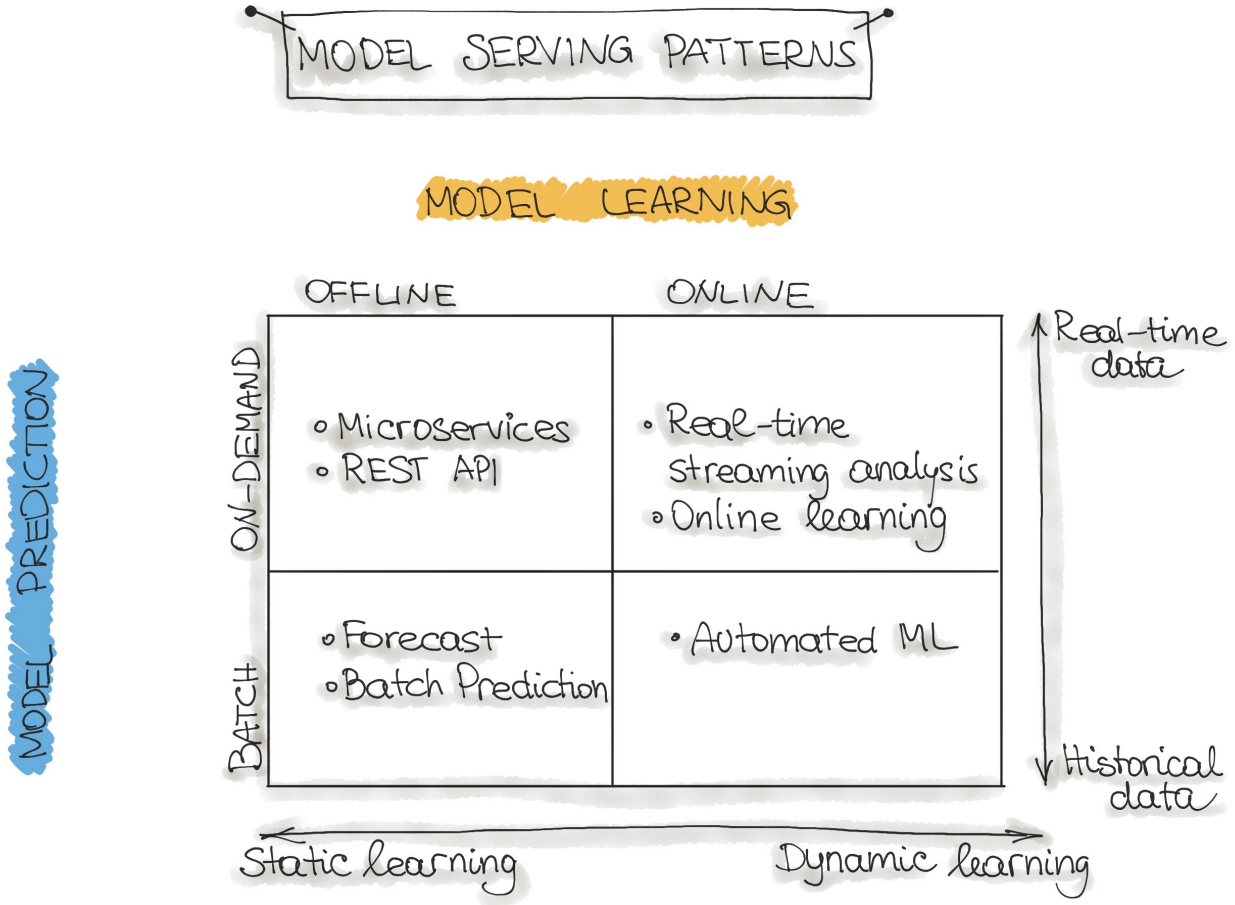
- **Data Engineering**: data acquisition & data preparation,
- **ML Model Engineering**: ML model training & serving, and
- **Code Engineering** : Integrating ML model into the final product.



MLOPS

Serving Patterns

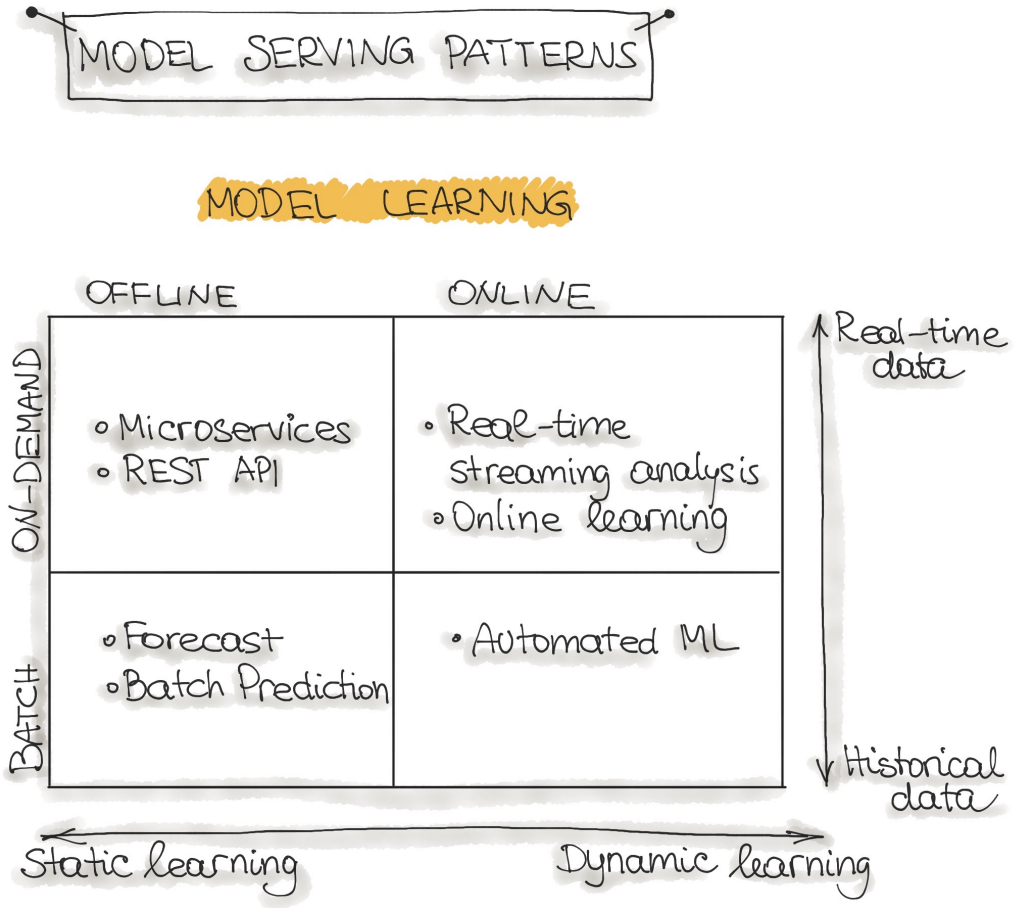
How to decide a serving pattern for your model?



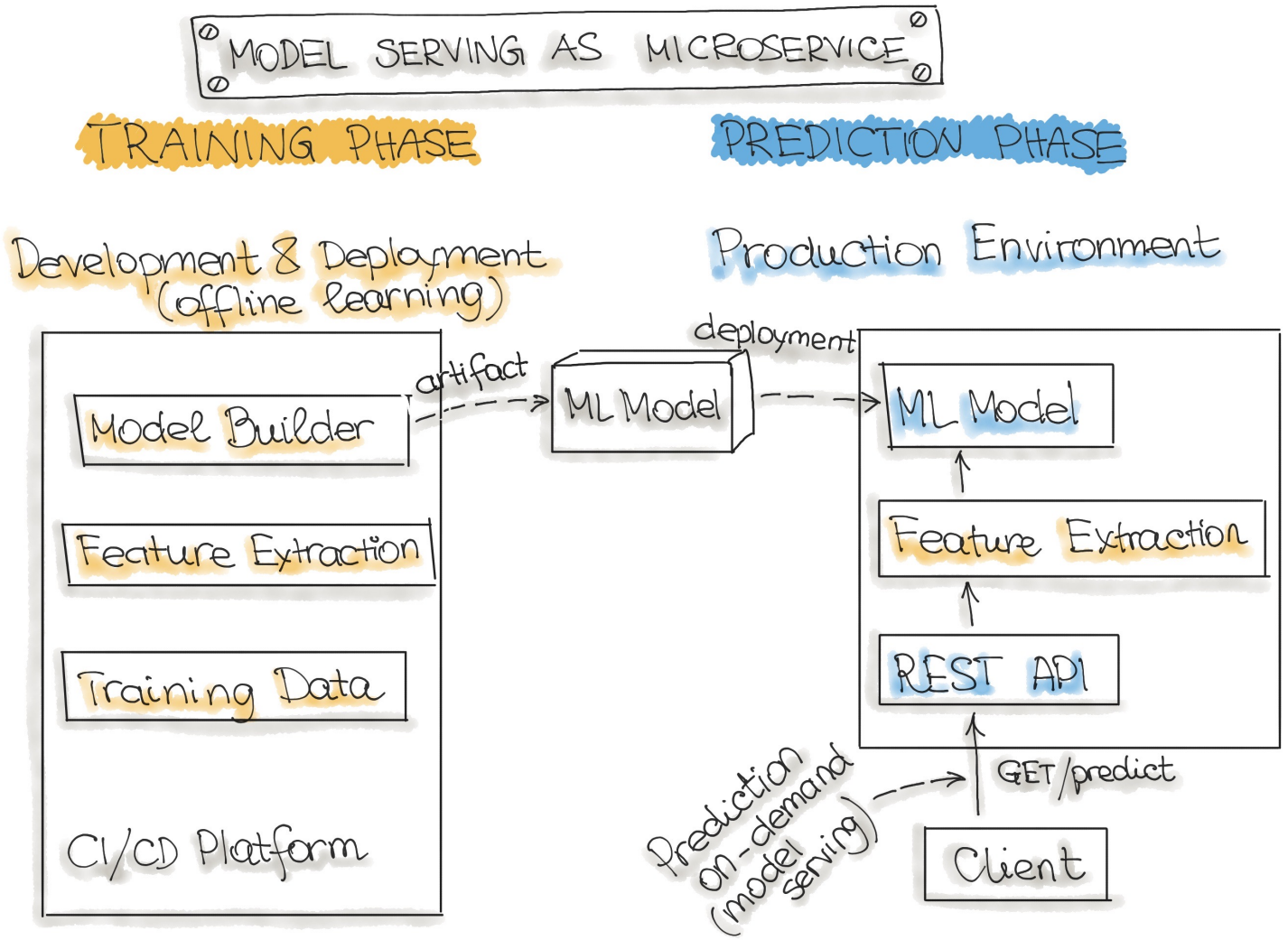
How to decide a serving pattern for your model?

Ask the following questions:

- Do I need to process requests in real-time?
- How many requests I need to handle at a time?
- How often I need to re-train my model?
- How the output of my model is consumed?
- What portion of app constitutes ML development?
- What's the volume of data for one request?



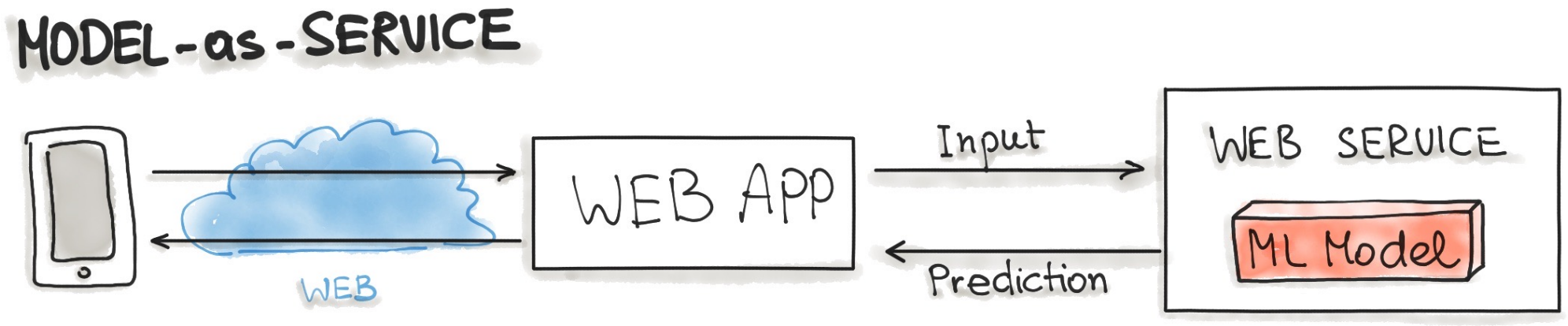
Model as Microservice serving



Model as Service

Model-as-Service is a common pattern for wrapping an ML model as an independent service. We can wrap the ML model and the interpreter within a dedicated web service that applications can request through a REST API or consume as a gRPC service.

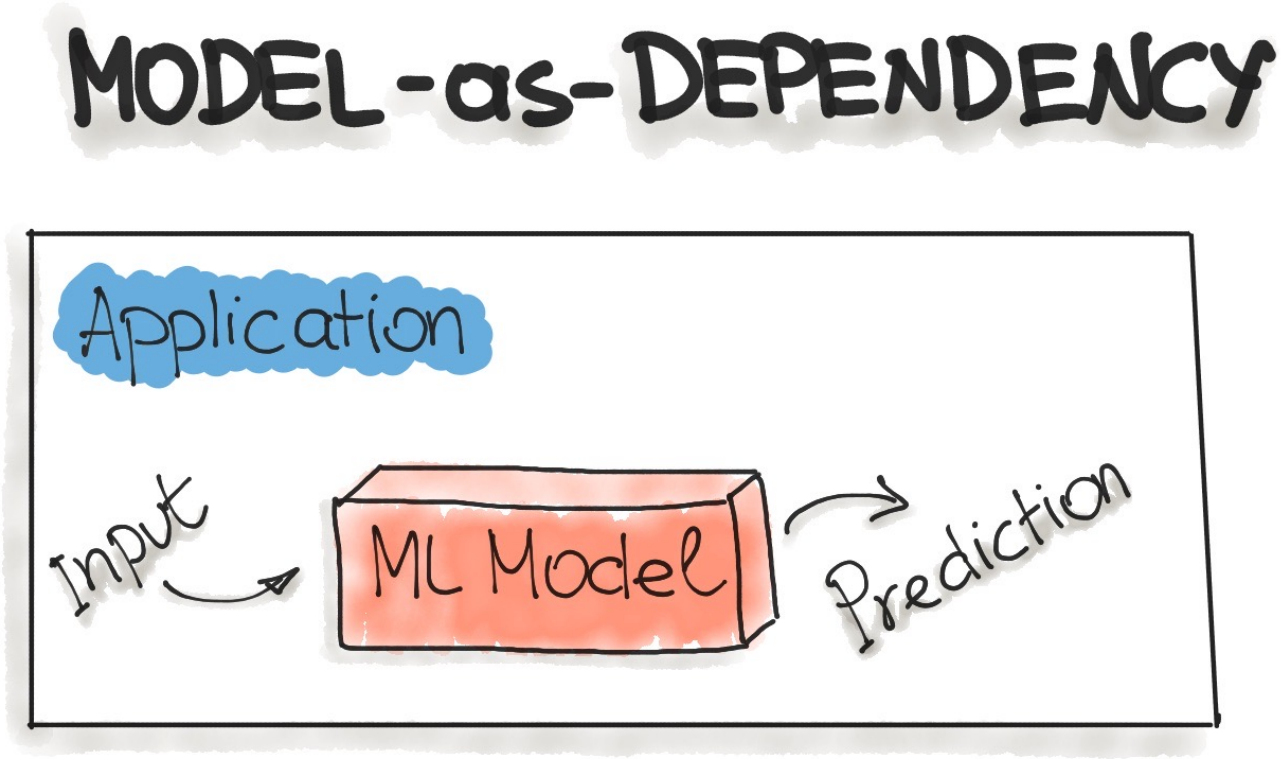
This pattern can be used for various ML workflows, such as Forecast, Web Service, Online Learning.



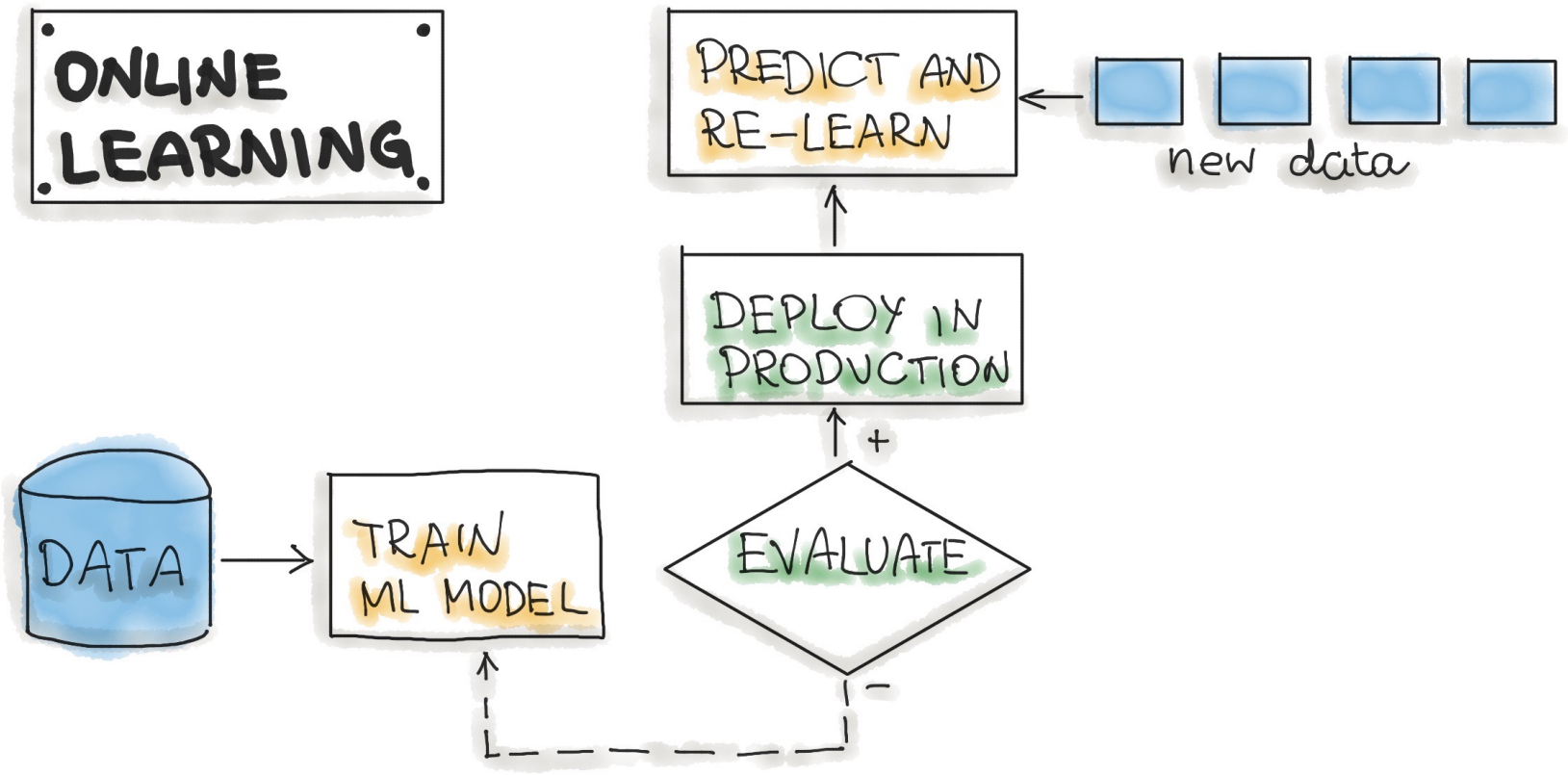
Model as Dependency Serving

Model-as-Dependency is probably the most straightforward way to package an ML model.
A packaged ML model is considered as a dependency within the software application.

For example, the application consumes the ML model like a conventional *jar* dependency by invoking the prediction method and passing the values. The return value of such method execution is some prediction that is performed by the previously trained ML model.

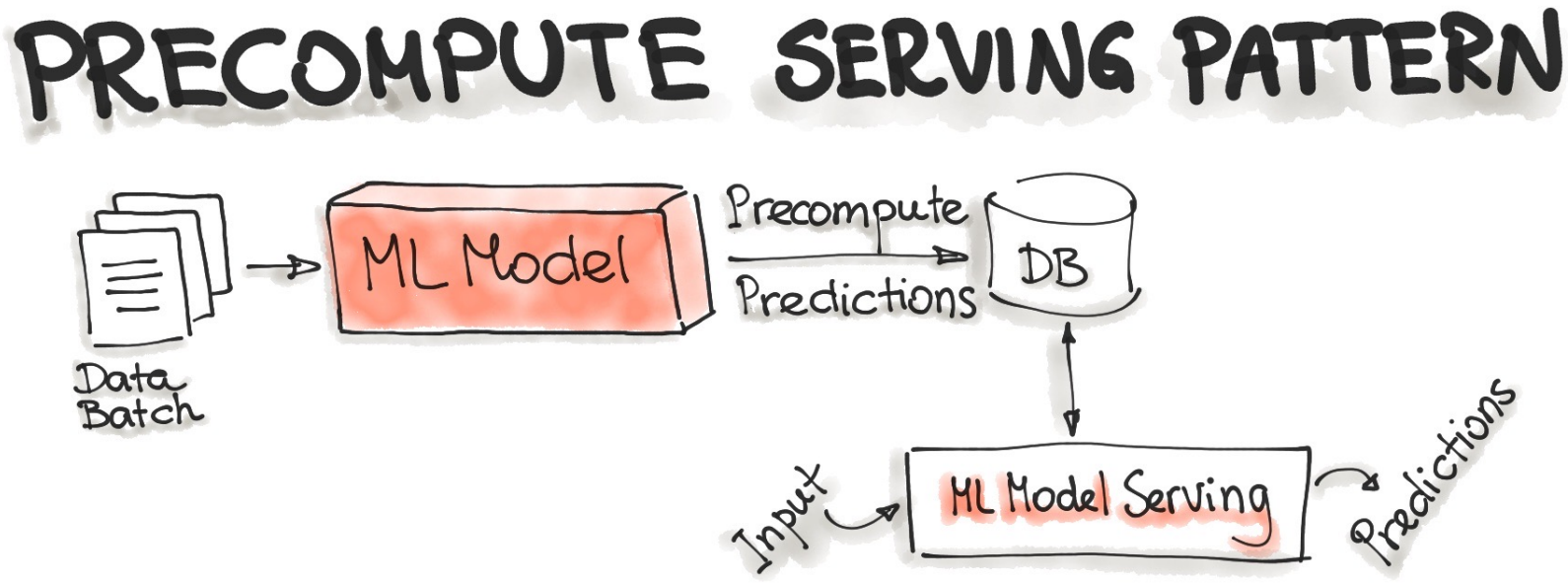


Online Learning



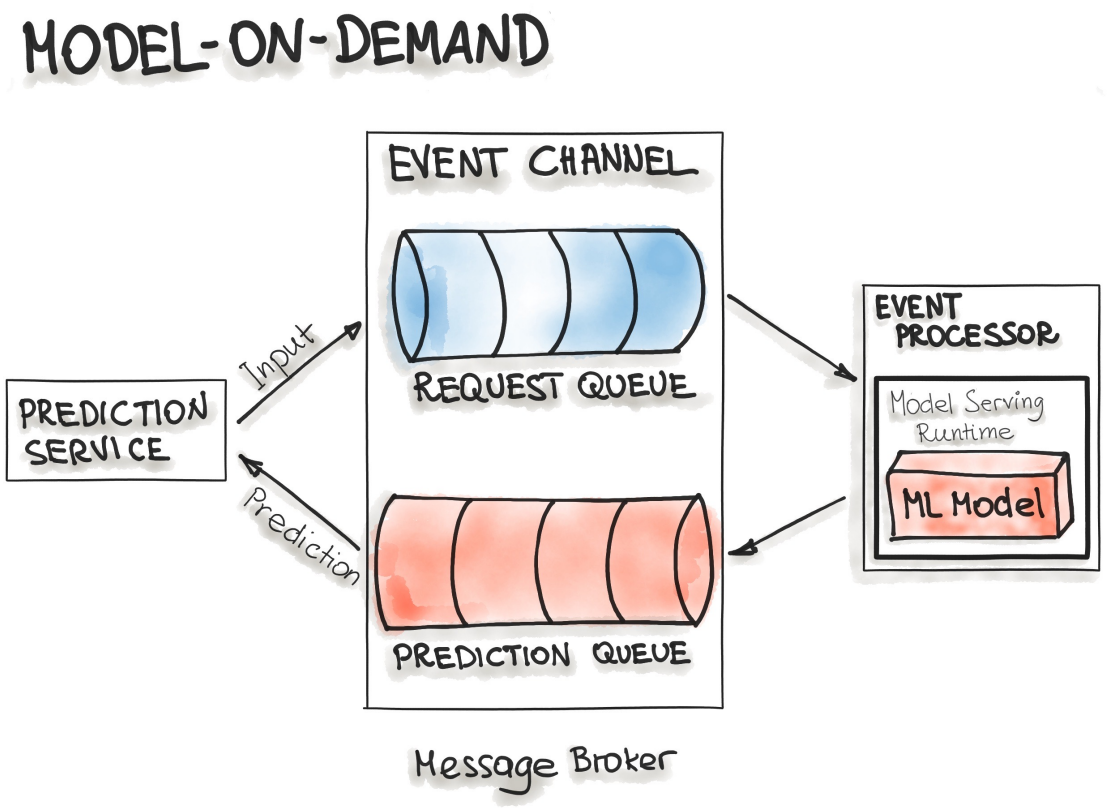
Precompute Serving

This type of ML model serving is tightly related to the *Forecast* ML workflow. With the *Precompute* serving pattern, we use an already trained ML model and precompute the predictions for the incoming batch of data. The resulting predictions are persisted in the database. Therefore, for any input request, we query the database to get the prediction result.



Model on Demand Serving

The Model-on-Demand pattern also treats the ML model as a dependency that is available at runtime. This ML model, contrary to the Model-as-Dependency pattern, has its own release cycle and is published independently.

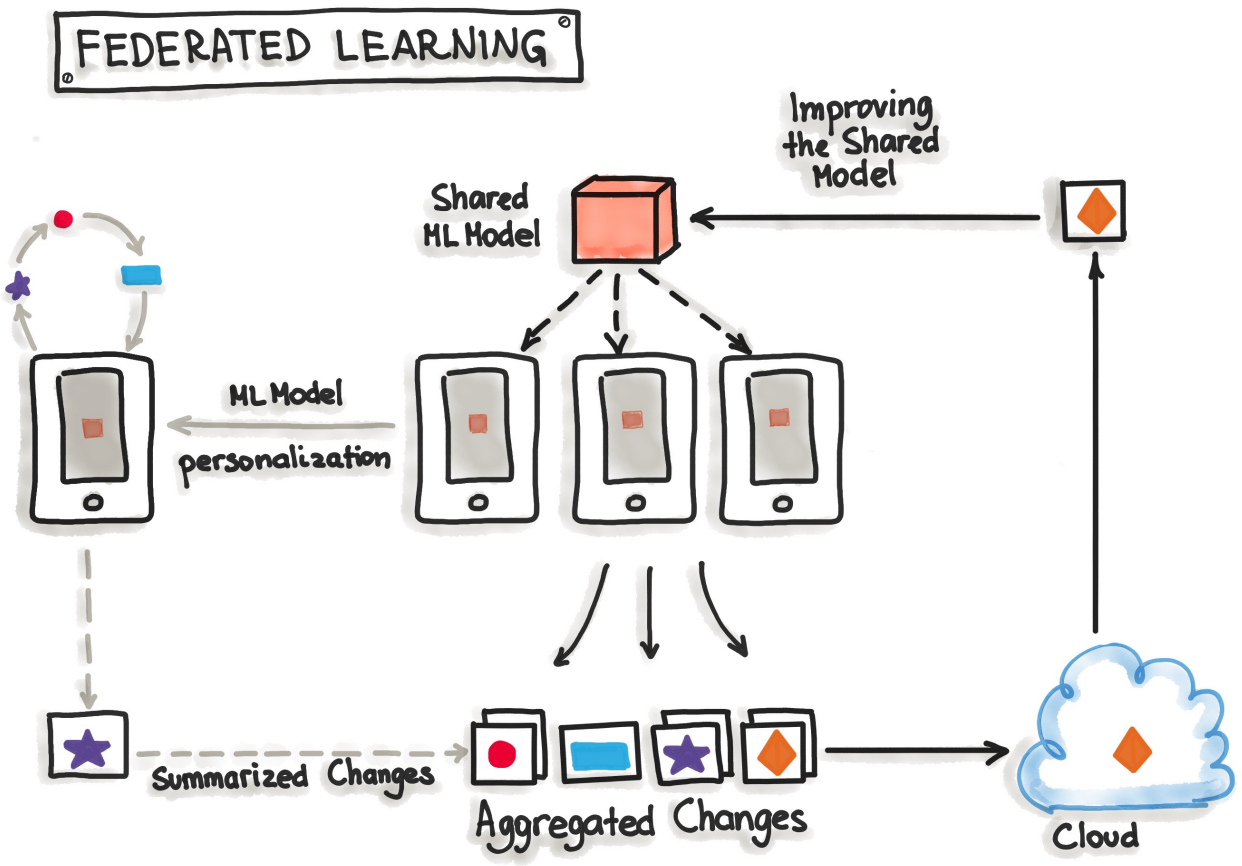


Hybrid Serving – Federated Learning

Federated Learning, also known as hybrid-serving, deploys multiple models, with one on the server and individual user-specific models.

The server's model is trained once on real-world data and serves as the initial model for users.

User-side models are trained on mobile devices, leveraging their increasing hardware capabilities. Periodically, these user models send data to update the server model, ensuring it captures current user trends.



MLOPS

Hands-on

Hands-on

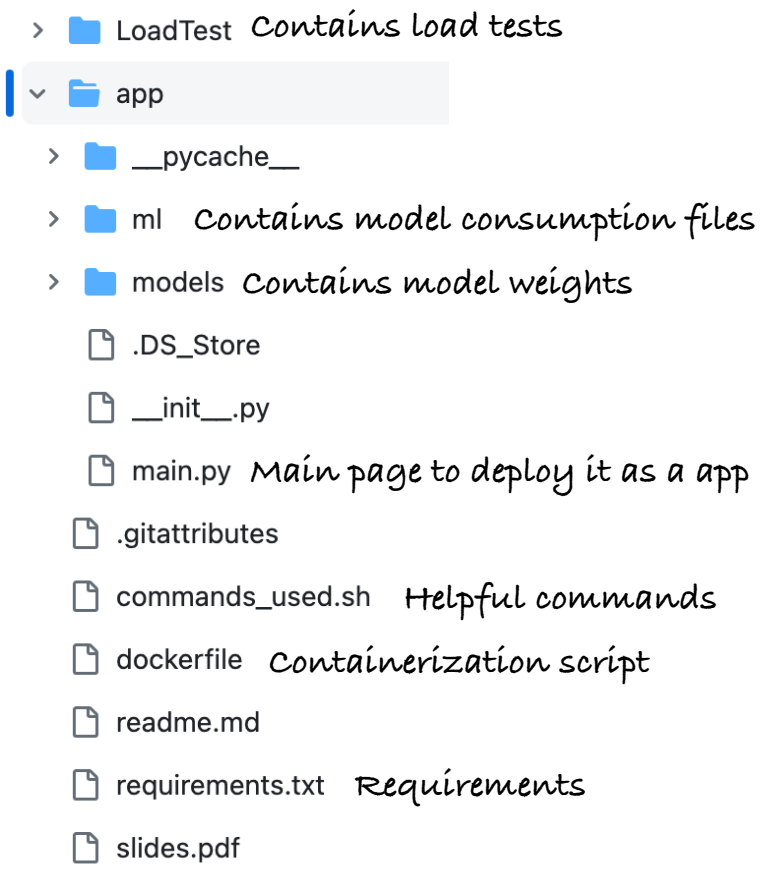
In hand-on session, we'll try to:

1. Serve a sentiment analysis model in "model-as-service" pattern using FastAPI & docker.
2. Load & Stress testing of the same model service, checking latency and TAT using JMeter.
3. Scale-up the same model to cater 10000+ users using Kubernetes and load-balancing.

https://github.com/tyagi-py/demystifying-mlops-and-kubernetes

Hands-on

Repo understanding



High Level Steps:

- Install `docker`
- `git clone` the repository
- Run `docker build .`
- Push image to docker-hub
- Deploy it using `Google Kubernetes Engine`
- Load test using `Jmeter`
- Scale-up settings and testing.

Hands-on