ESTIA
INSTITUTE OF TECHNOLOGY

MLOps / ML engineering

MSc BIHAR, Big Data & AI, 2024-2025

Dr Evgeniya ISHKINA

Laerning goals

- Understand the phases of the ML project lifecycle and their practical implications.
- Learn deployment strategies for ML applications.
- Explore and utilize MLOps tools to track experiments and manage ML training pipelines.
- Deploy an ML model and monitor its performance.

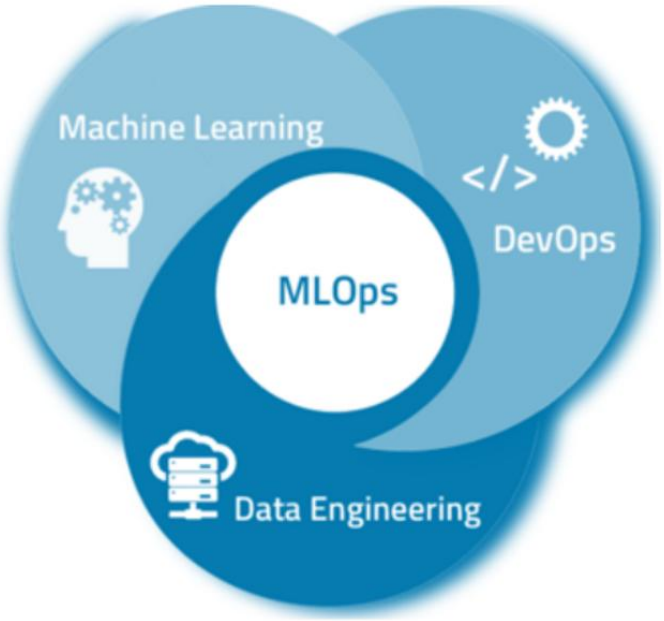
Syllabus

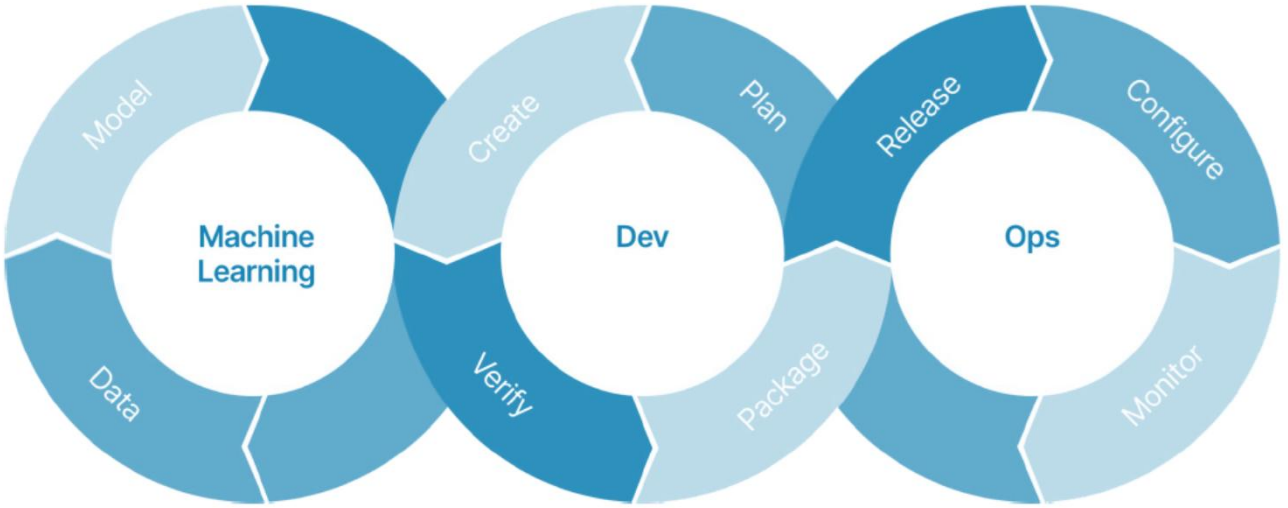
- ML Project Lifecycle – Key phases, common challenges, and an introduction to MLOps practices.
- CRISP-ML(Q) Methodology – Principles and application in ML projects.
- MLOps Tools – Introduction to MLflow, experiment tracking, model and metric traceability.
- API Development – Exposing ML models using FastAPI.
- Model Monitoring – Tracking performance in production, detecting drift, and managing updates.

MLOps

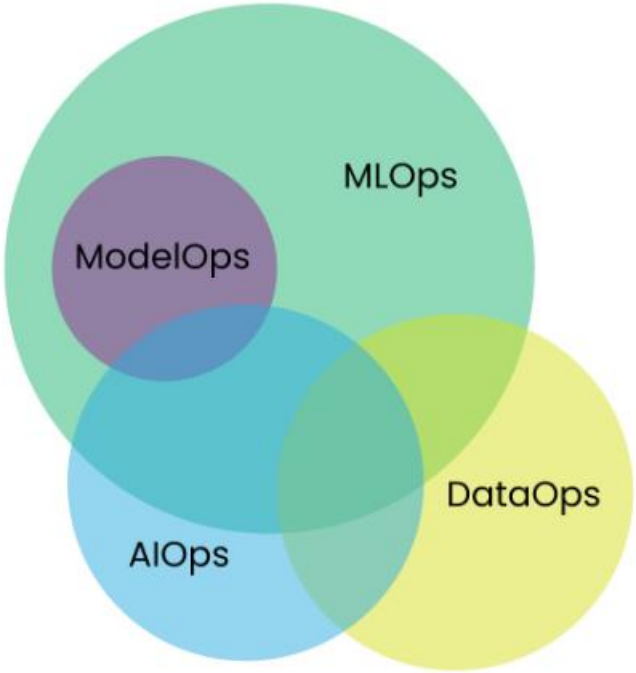
- MLOps = Machine Learning Operations
- … is a framework of practices to design, deploy and maintain machine learning models in production continuously, reliably, and efficiently.
- Machine Learning models in production should have four qualities: scalable, reproducible, testable, and evolvable.
- Machine learning engineering for production combines the foundational concepts of machine learning with the functional expertise of data engineering and software engineering roles, including DevOps.
- Started as a set of best practices, MLOps is evolving into an independent approach to ML lifecycle management.

MLOps


Machine Learning
DevOps
MLOps
Data Engineering


Model
Create
Plan
Release
Configure
Machine Learning
Dev
Ops
Data
Verify
Package
Monitor

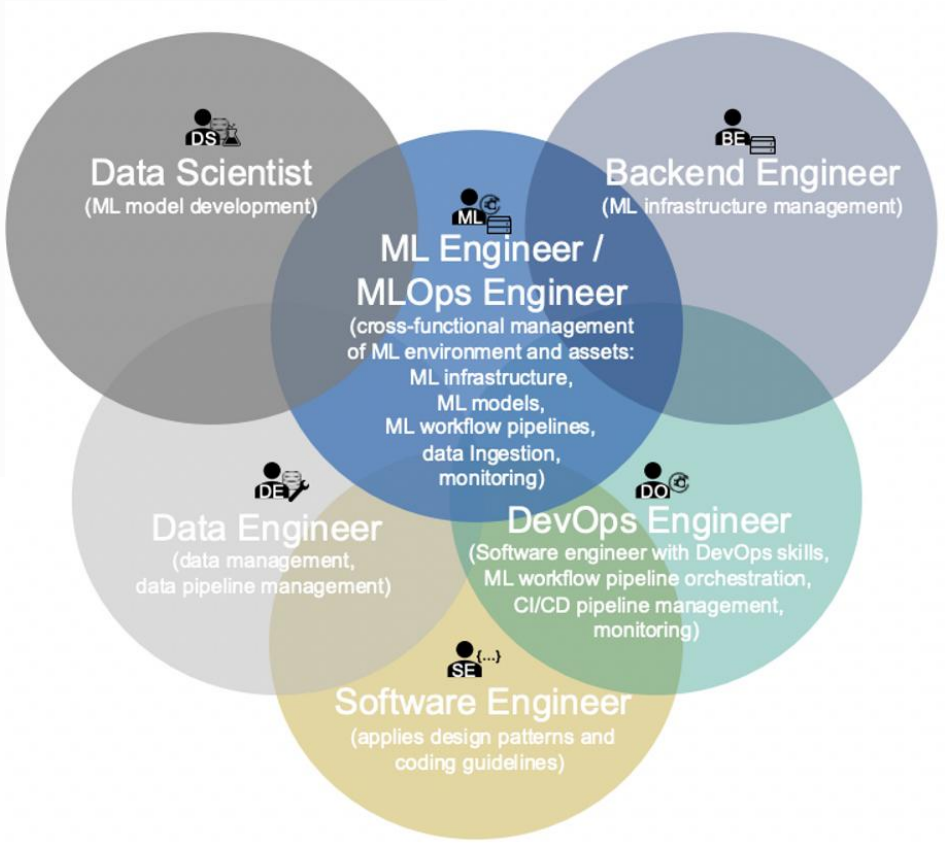
...Ops



Source: datacamp.com

- DataOps: Data Operations
  - Best practices in data serving and data quality
- ModelOps: Model Operations
  - Part/extension of MLOps, primarily focused on ML models
- AIOps: AI for IT operations
  - Using Big Data and Machine Learning to solve IT issues without human assistance

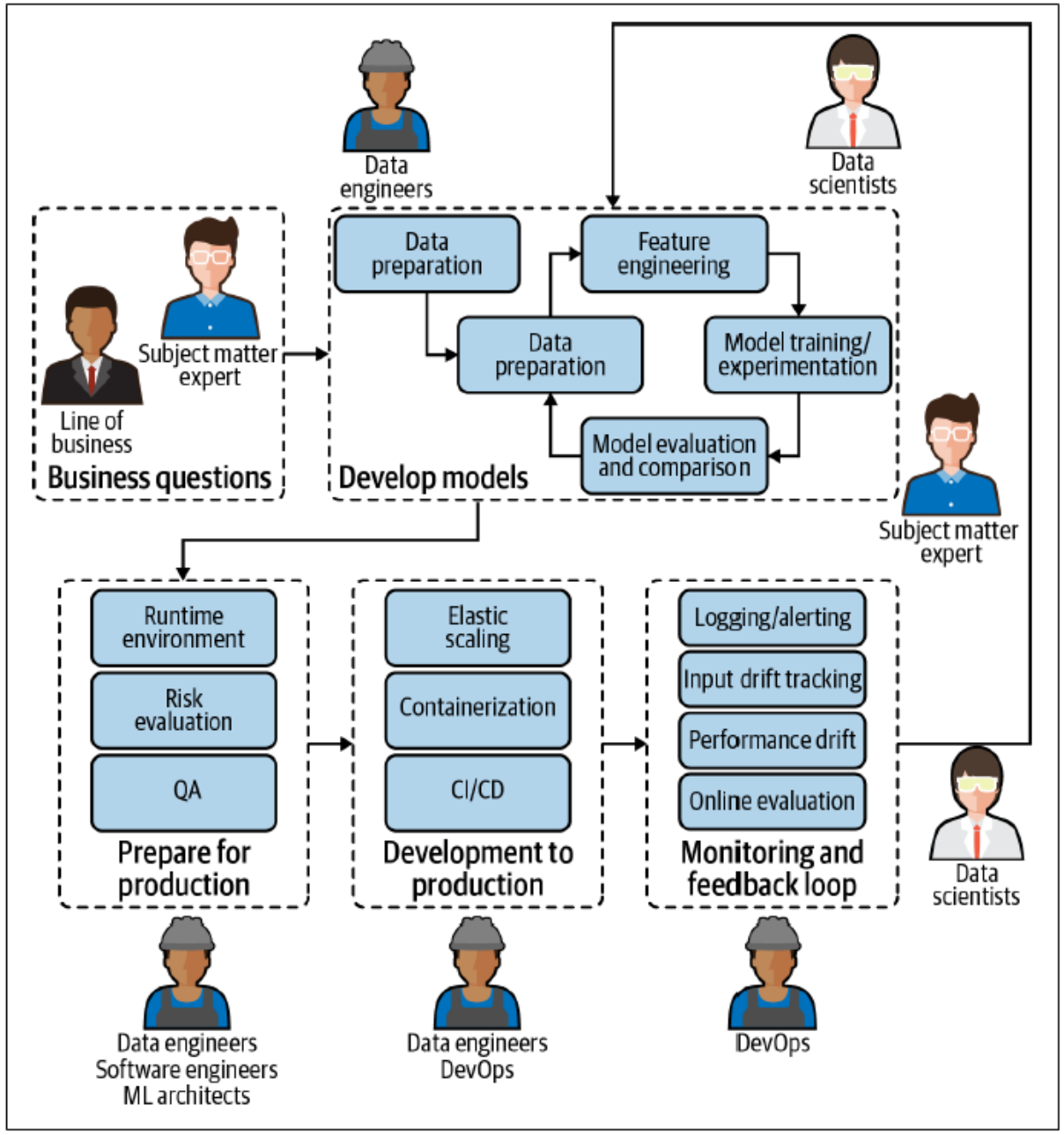
ML/MLOps engineer



Source: https://arxiv.org/ftp/arxiv/papers/2205/2205.02302.pdf

Develops a ML project and gets it into production:
cross-functional role (over the entire ML project lifecycle)

- Data engineering skills
  - Need to get data for analysis somewhere
  - ML engineer is **not** likely to write their own large-scale streaming ingestion ETL pipelines.
- Data science skills
  - With so many specialties existing in the field (NLP, CV, forecasting, deep learning, traditional linear and tree-based modeling, etc.), and so many existing algorithms to solve specific problems, it's very challenging to learn more than a tiny fraction of that.
- Software engineering skills
  - Serve models via a REST API /..
  - Write modular code and implement tests
  - ML engineer doesn't need to create applications and software frameworks for generic use cases, nor to create detailed and animated front-end visualizations.



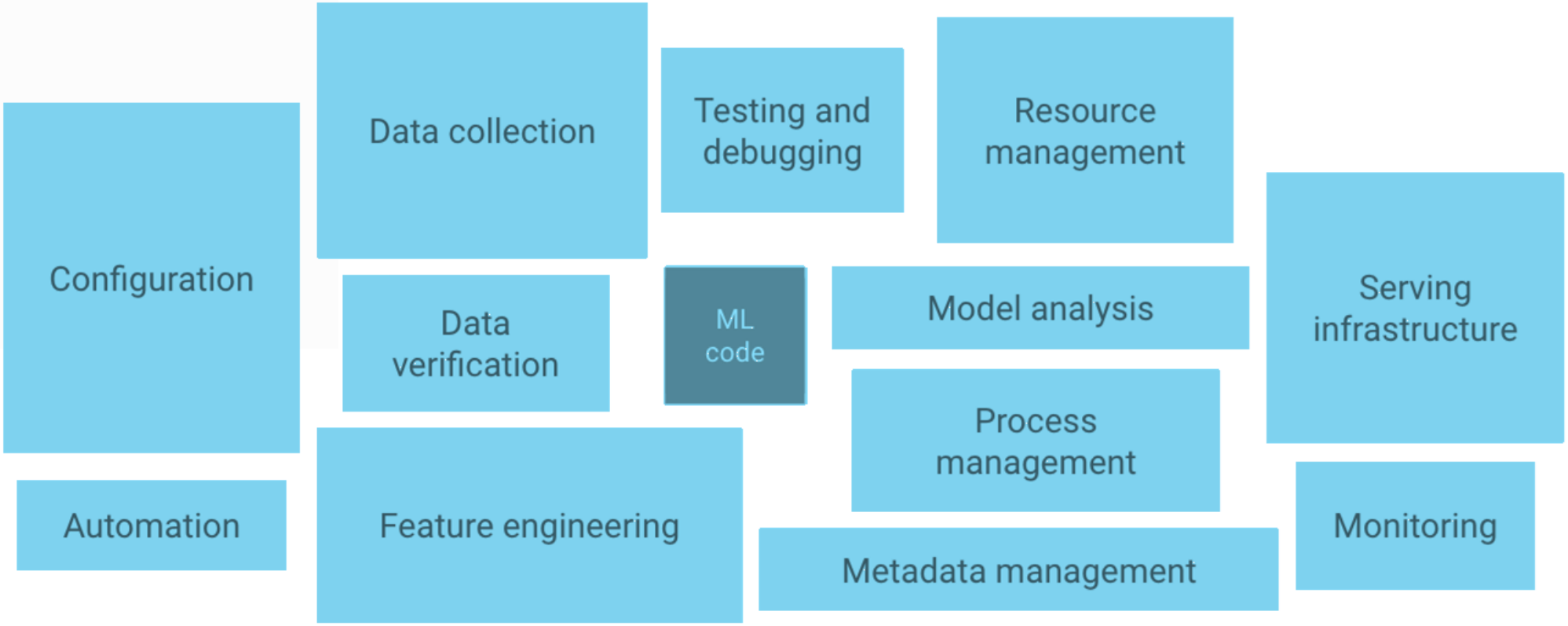
- The realistic picture of an ML model life cycle inside an average organization, which involves many different people with completely different skill sets and who are often using entirely different tools.
- Until recently, the number of models may have been manageable at a small scale, or there was simply less interest in understanding these models and their dependencies at a company-wide level. For most traditional organizations, working with multiple ML models is still relatively new.
- MLOps is the standardization of ML project life cycle management.

Real-world ML system

Data science and ML are becoming core capabilities for solving complex real-world problems, transforming industries, and delivering value across all domains. Currently, the following key ingredients for applying effective ML are available:

- Large datasets
- Inexpensive, on-demand compute resources
- Specialized accelerators for ML on various cloud platforms
- Rapid advances in different ML research fields (such as computer vision, natural language processing, and recommendation systems).

Real-world ML system



https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

ESTIA
INSTITUTE OF TECHNOLOGY

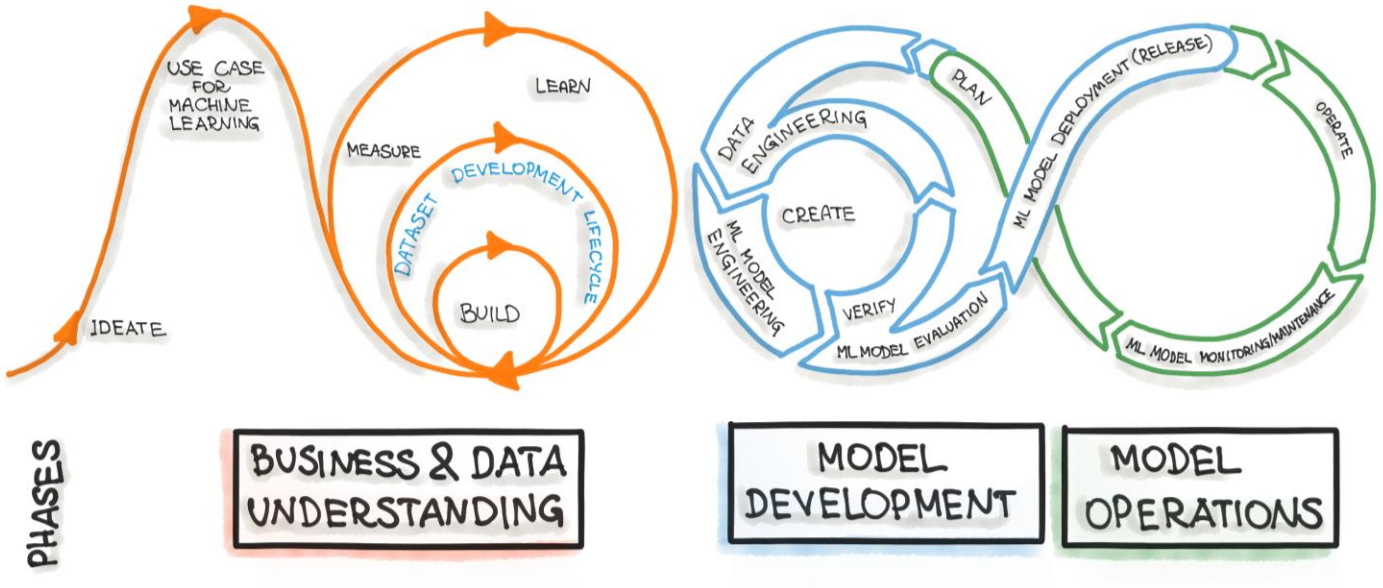
CRISP-ML(Q)

Cross-Industry Standard Process for Machine Learning development

CRISP-ML(Q)

The ML community is still trying to establish a standard process model for ML development. As a result, many ML projects are not well organized. In general, such projects are conducted in an ad-hoc manner.

CRISP-DM



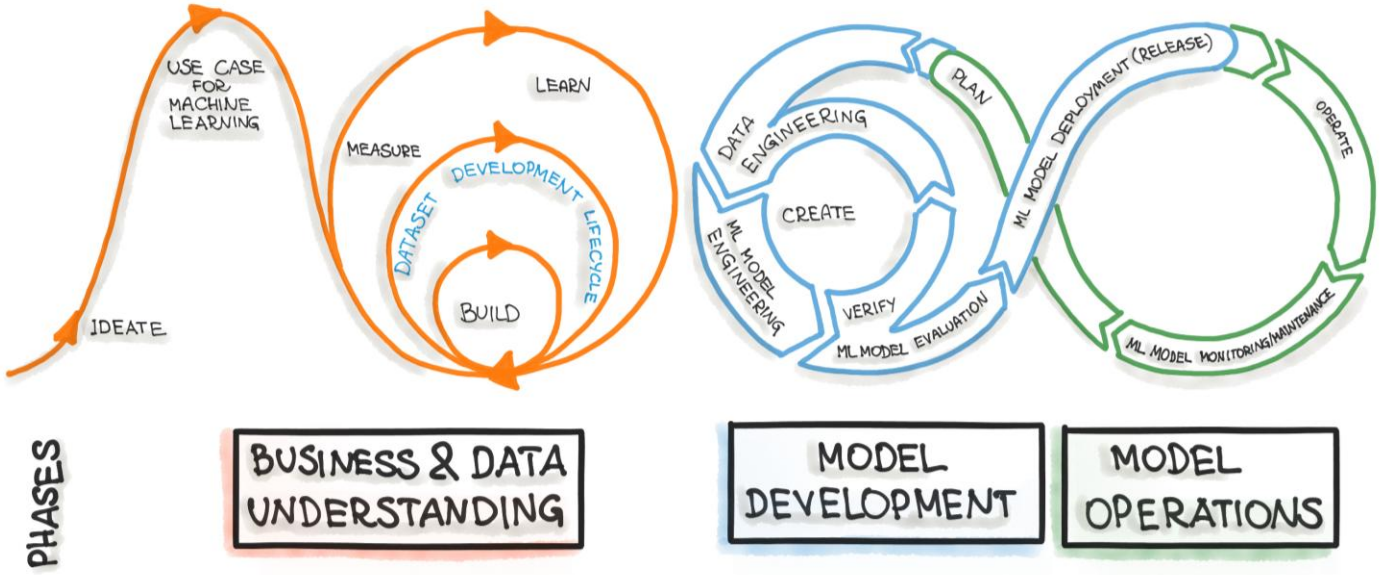
CRISP-ML

ML workflows are fundamentally iterative and exploratory, so that depending on the results from the later phases, earlier steps might be re-examined.

CRISP-ML(Q)

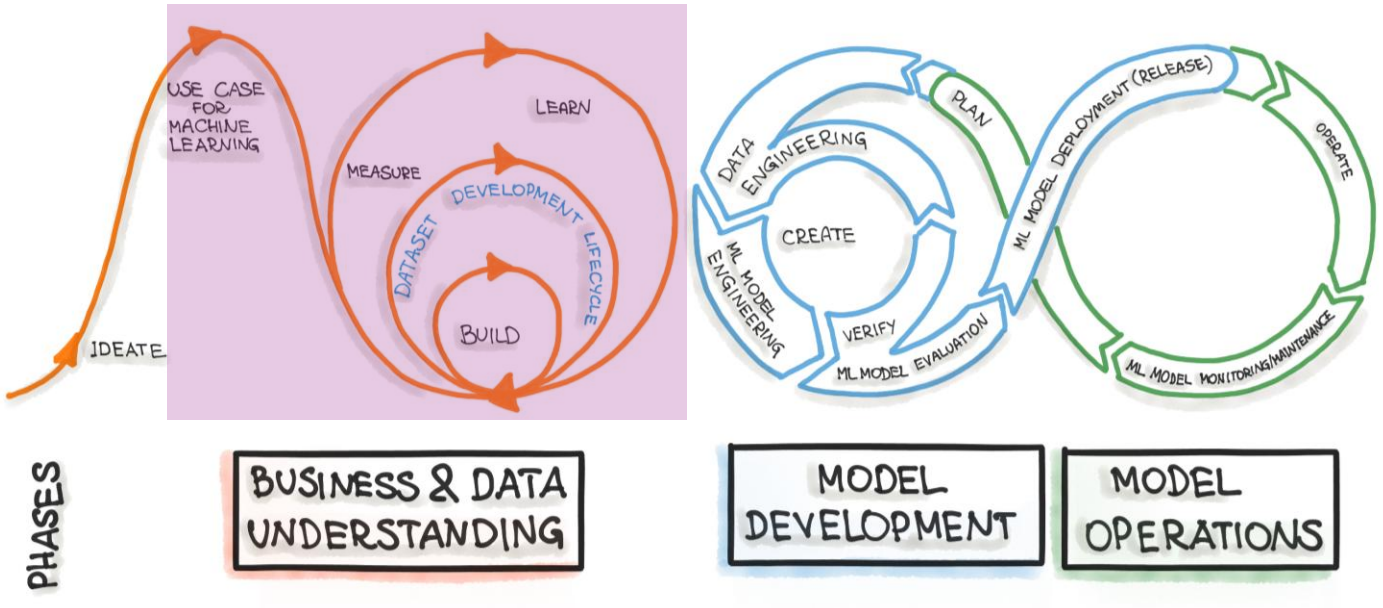
1. Business and Data Understanding
2. Data Engineering (Data Preparation)
3. Machine Learning Model Engineering
4. Quality Assurance
5. Deployment
6. Monitoring and Maintenance

Q : Quality assurance methodology



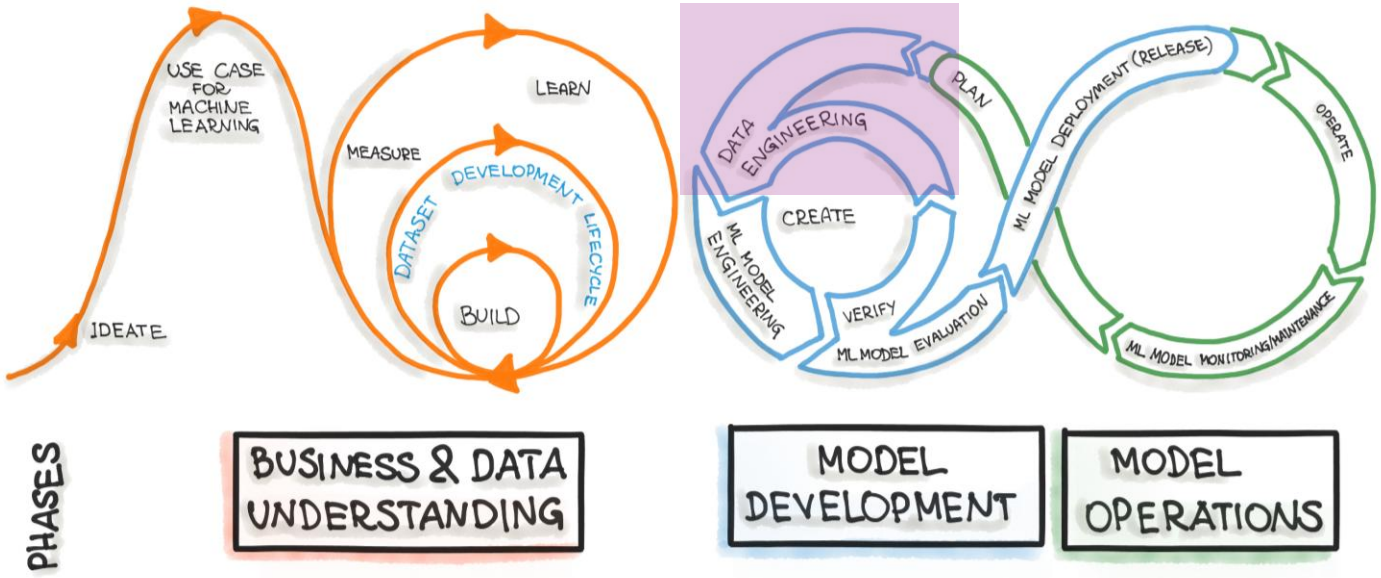
CRISP-ML(Q): Business and Data Understanding

- Define business objectives and key metrics
- Translate business objectives into ML objectives
- Collect and verify data
- Assess the project feasibility
- Create POC



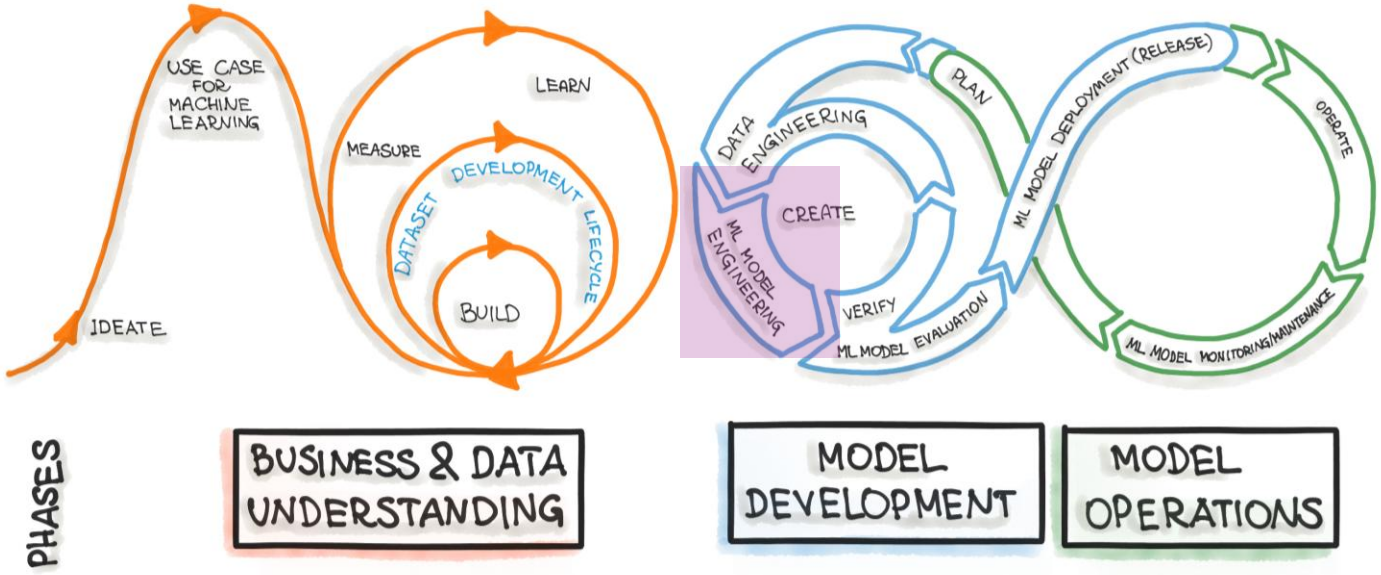
CRISP-ML(Q): Data Engineering

- Feature selection
- Data selection
- Class balancing
- Cleaning data (noise reduction, data imputation)
- Feature engineering
- Data augmentation
- Data standardization



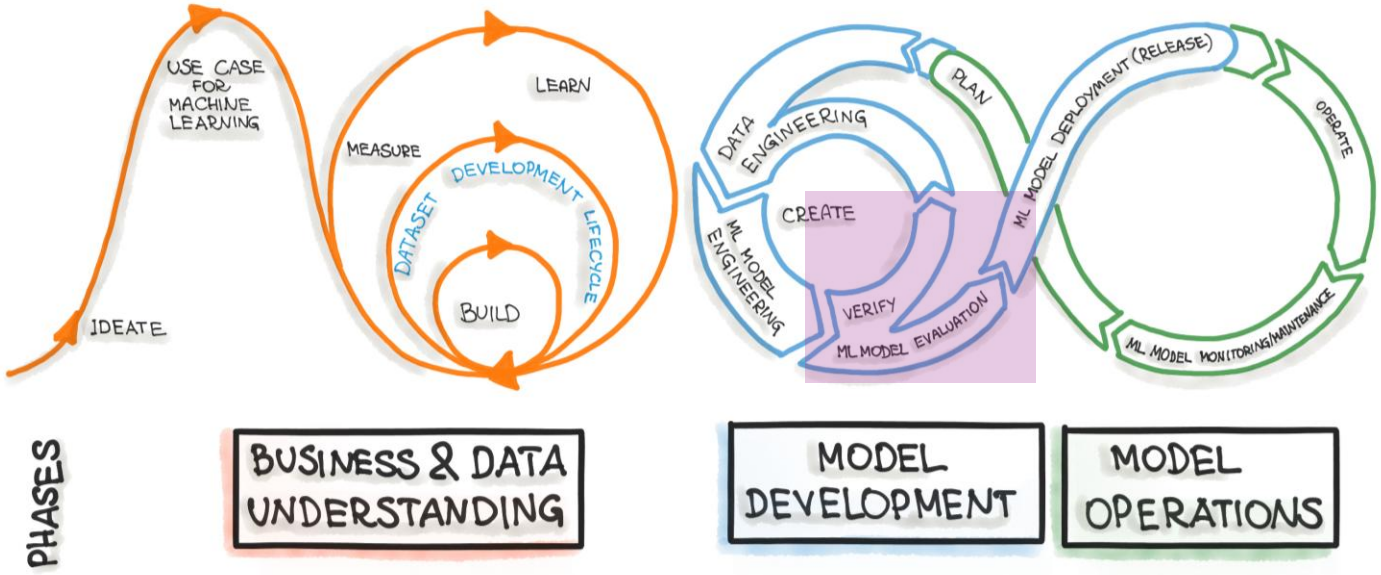
CRISP-ML(Q): ML Model Engineering

- Define quality measure of the model
- ML algorithm selection (baseline)
- Using domain knowledge to specialize the model
- Model training / tuning (transfer learning) / ensemble learning
- Model serialization
- Documenting the ML model and experiments



CRISP-ML(Q): ML Model Evaluation

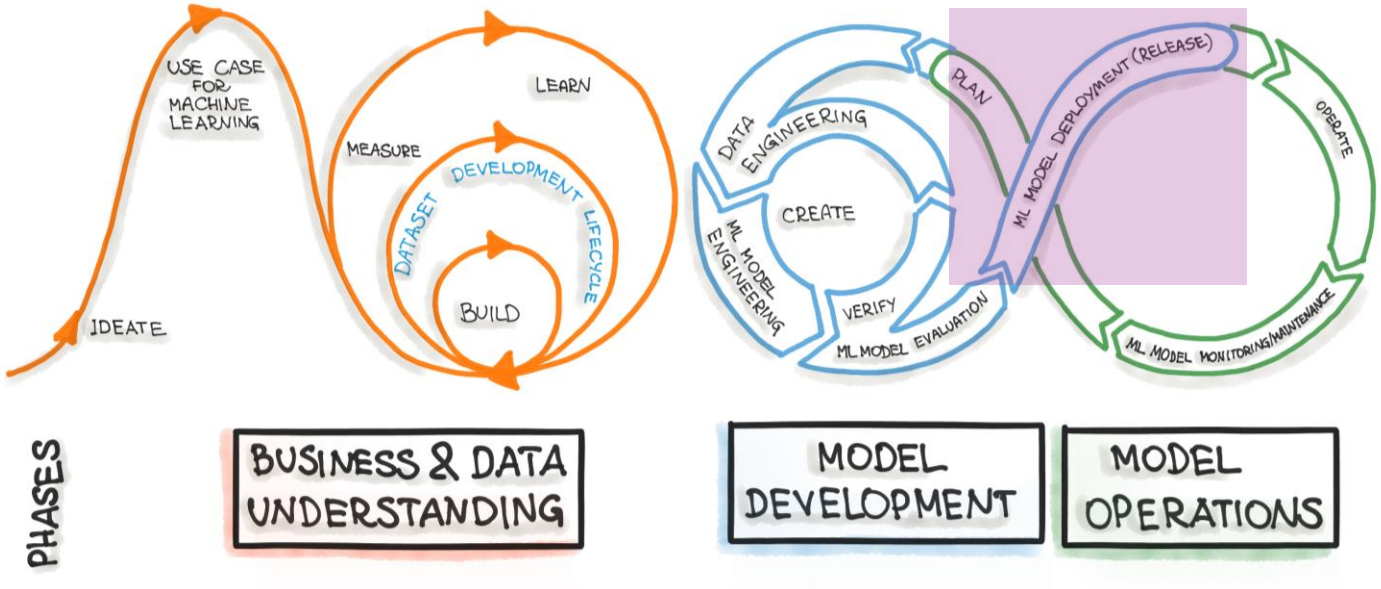
- Validate model's performance
- Determine robustness
- Increase model's explainability
- Make a decision whether to deploy the model
- Document the evaluation phase



CRISP-ML(Q): ML Model Deployment

- Build microservices from the ML model
- Evaluate model under production condition
- Assure user acceptance and usability
- Model governance
- Deploy according to the selected strategy

Model serving is executing a model with input data to make predictions (fetching the expected model, setting up the model's execution environment, executing the model to make a prediction with given data points, and returning the prediction result.

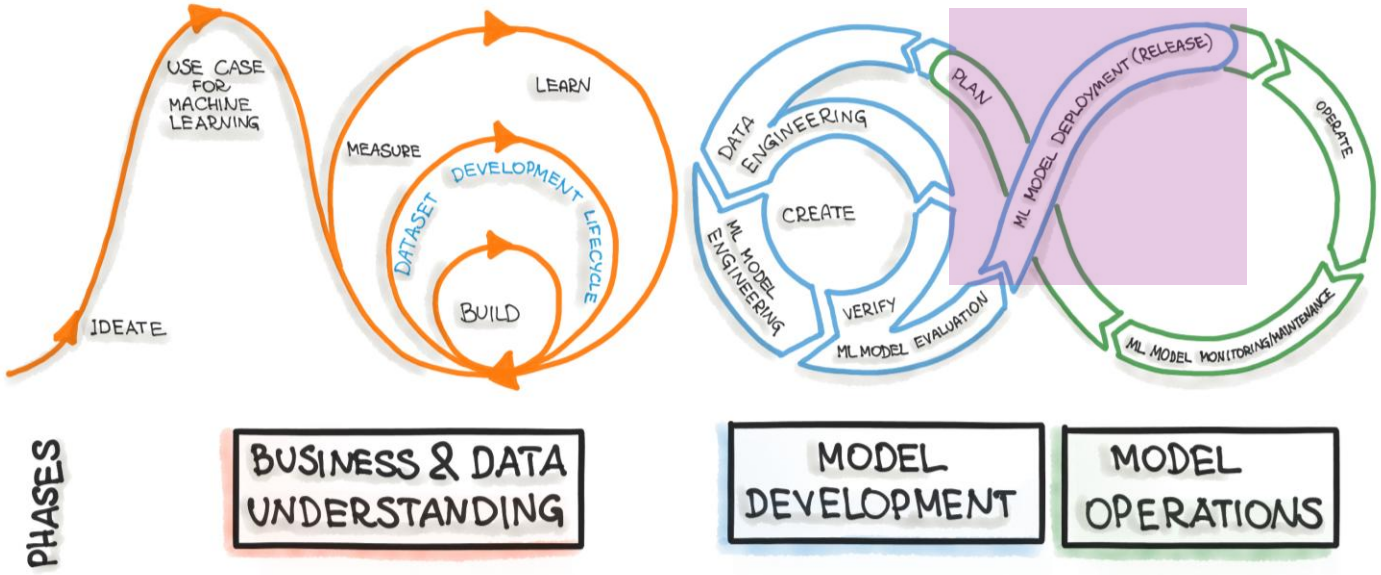


A microservice is a small application that includes the ML model such that we can easily integrate it into the business process

CRISP-ML(Q): ML Model Deployment

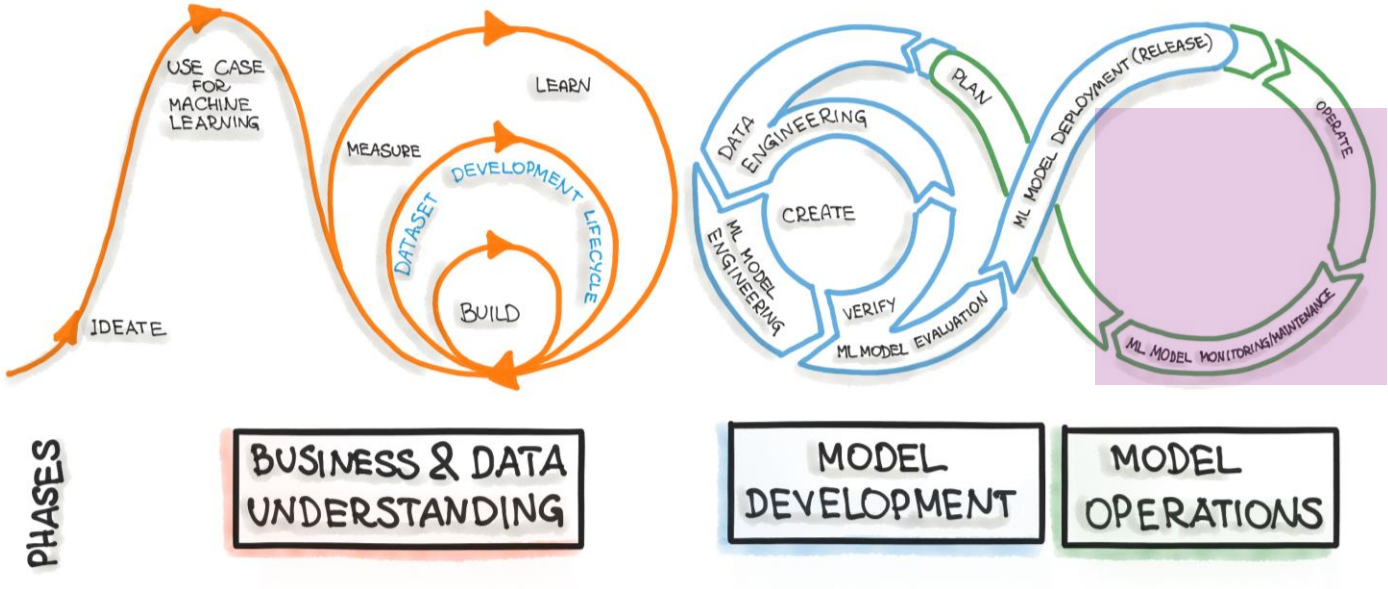
- Build microservices from the ML model
- Evaluate model under production condition
- Assure user acceptance and usability
- Model governance
- Deploy according to the selected strategy

Organizations often don't recognize the importance of Model Governance until models are supposed to be deployed. Many companies have automatized ML pipelines but fail to bring models into compliance with legal requirements.

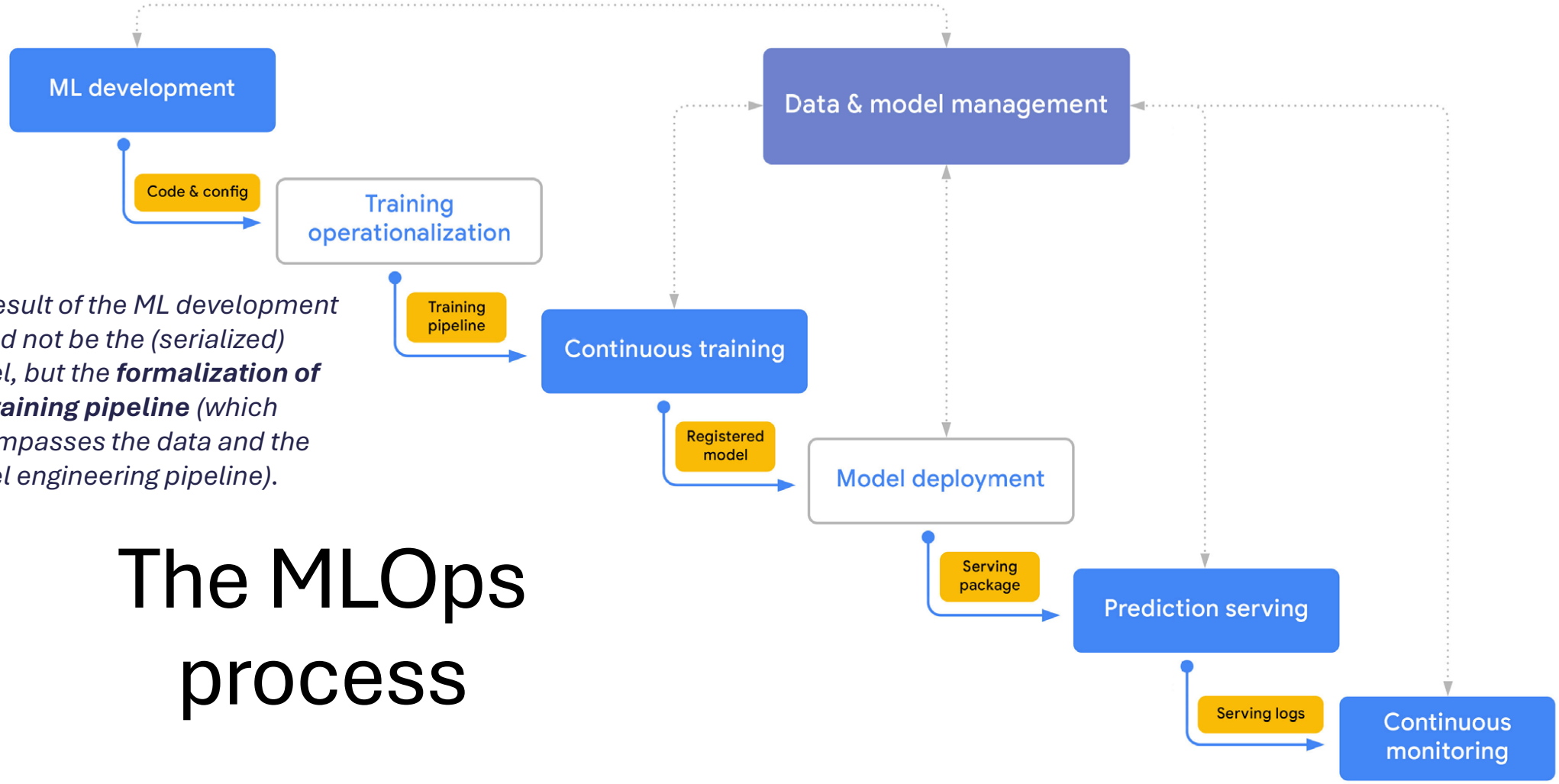


CRISP-ML(Q): Monitoring and Maintenance

- Monitor the efficiency and quality (accuracy) of the model prediction serving
- Compare to the previously specified success criteria
- Retrain model if required
  - Collect new data
  - Repeat tasks from the *Model Engineering* and *Model Evaluation* phases
  - Continuous, integration, training, and deployment of the retrained model



The MLOps process

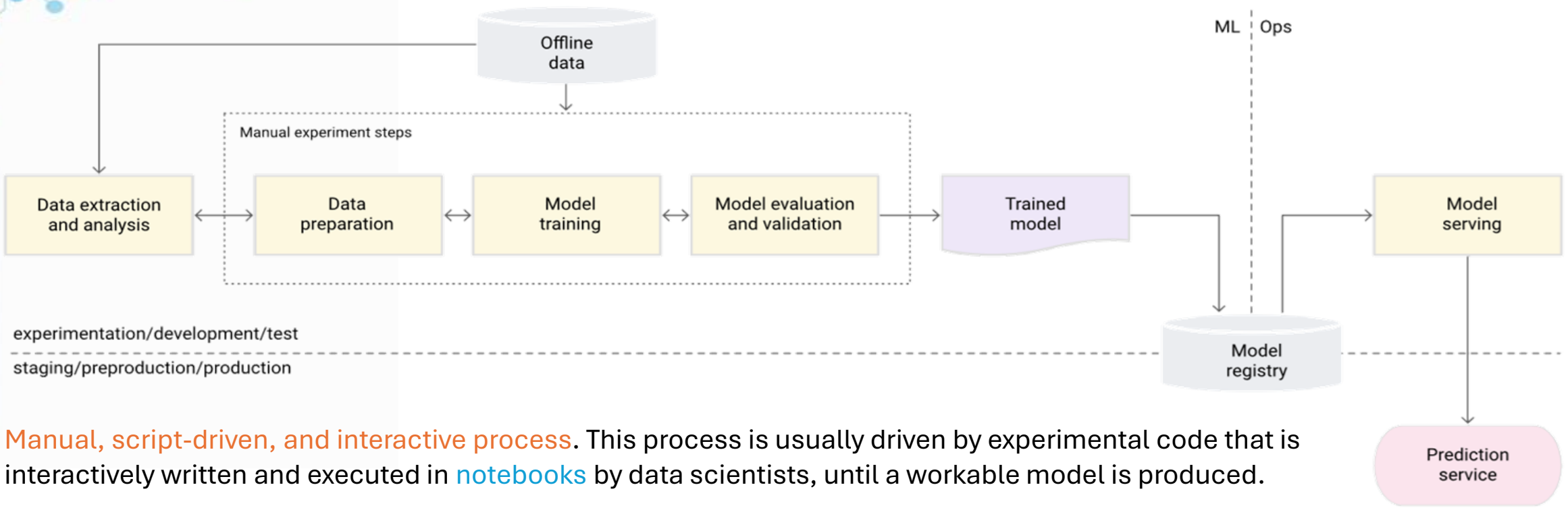


*The result of the ML development should not be the (serialized) model, but the **formalization of the training pipeline** (which encompasses the data and the model engineering pipeline).*

ESTIA
INSTITUTE OF TECHNOLOGY

MLOps automatization levels

MLOps level 0: manual process



Manual, script-driven, and interactive process. This process is usually driven by experimental code that is interactively written and executed in notebooks by data scientists, until a workable model is produced.

Disconnection between ML and operations (Ops): The process separates data scientists who create the model and engineers who serve the model as a prediction service. The data scientists hand over a trained model as an artifact to the engineering team to deploy on their API infrastructure.

No CI (few implementation changes are assumed). No CD (there aren't frequent model version deployments).

MLOps level 0 is common in many businesses that are beginning to apply ML to their use cases. This manual, data-scientist-driven process might be sufficient when models are rarely changed or retrained.

https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

MLOps level 1: ML pipeline automation

Rapid experiment: The steps of the ML experiment are orchestrated and the transition between them is automated.

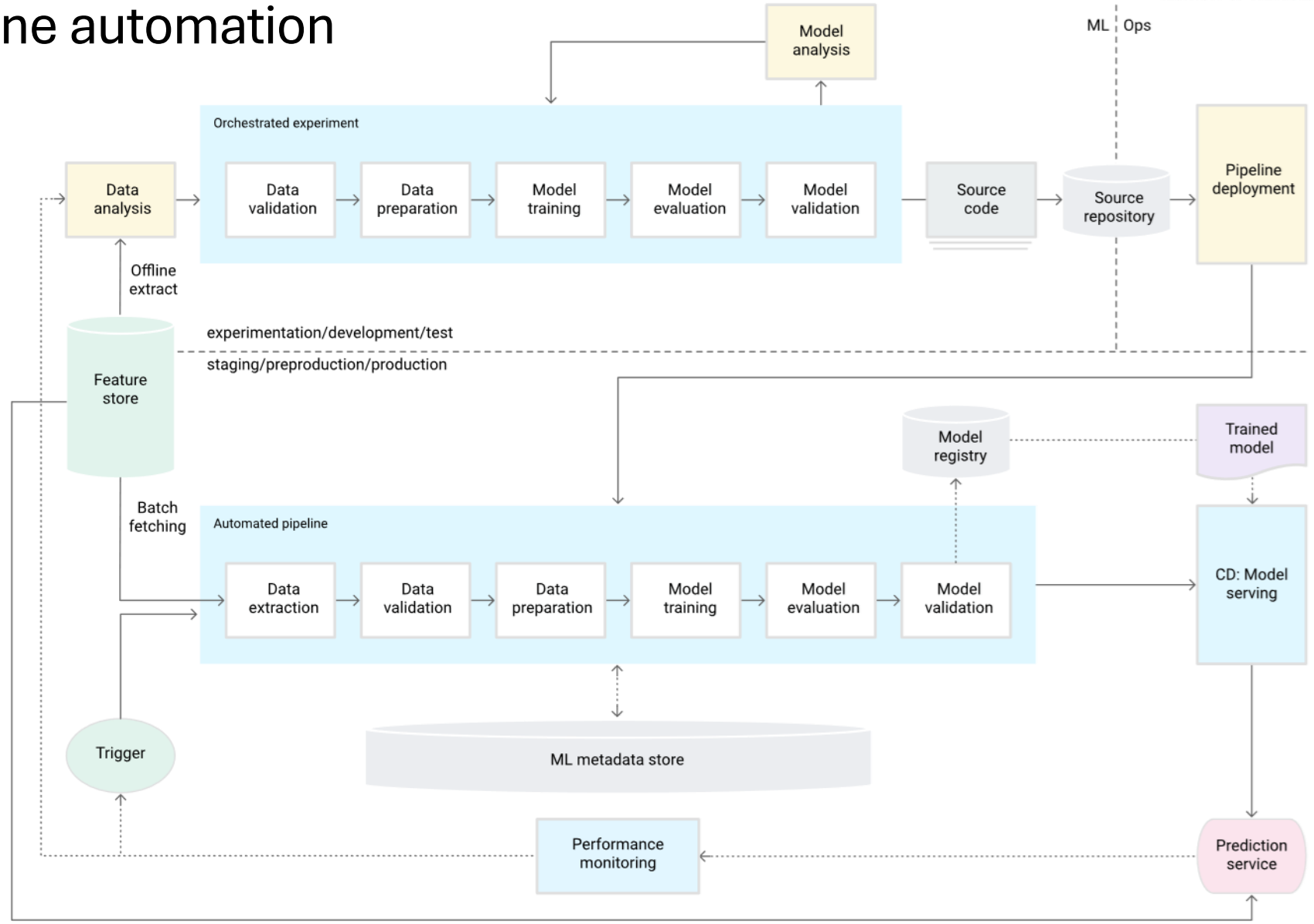
Continuous training (CT): The model is automatically trained in production.

Modularized code for components and pipelines (EDA code can still live in notebooks).

Continuous delivery of models: An ML pipeline in production continuously delivers prediction services to new models that are trained on new data.

Pipeline deployment:

- Level 0: a trained model is deployed
- Level 1: a whole training pipeline is deployed

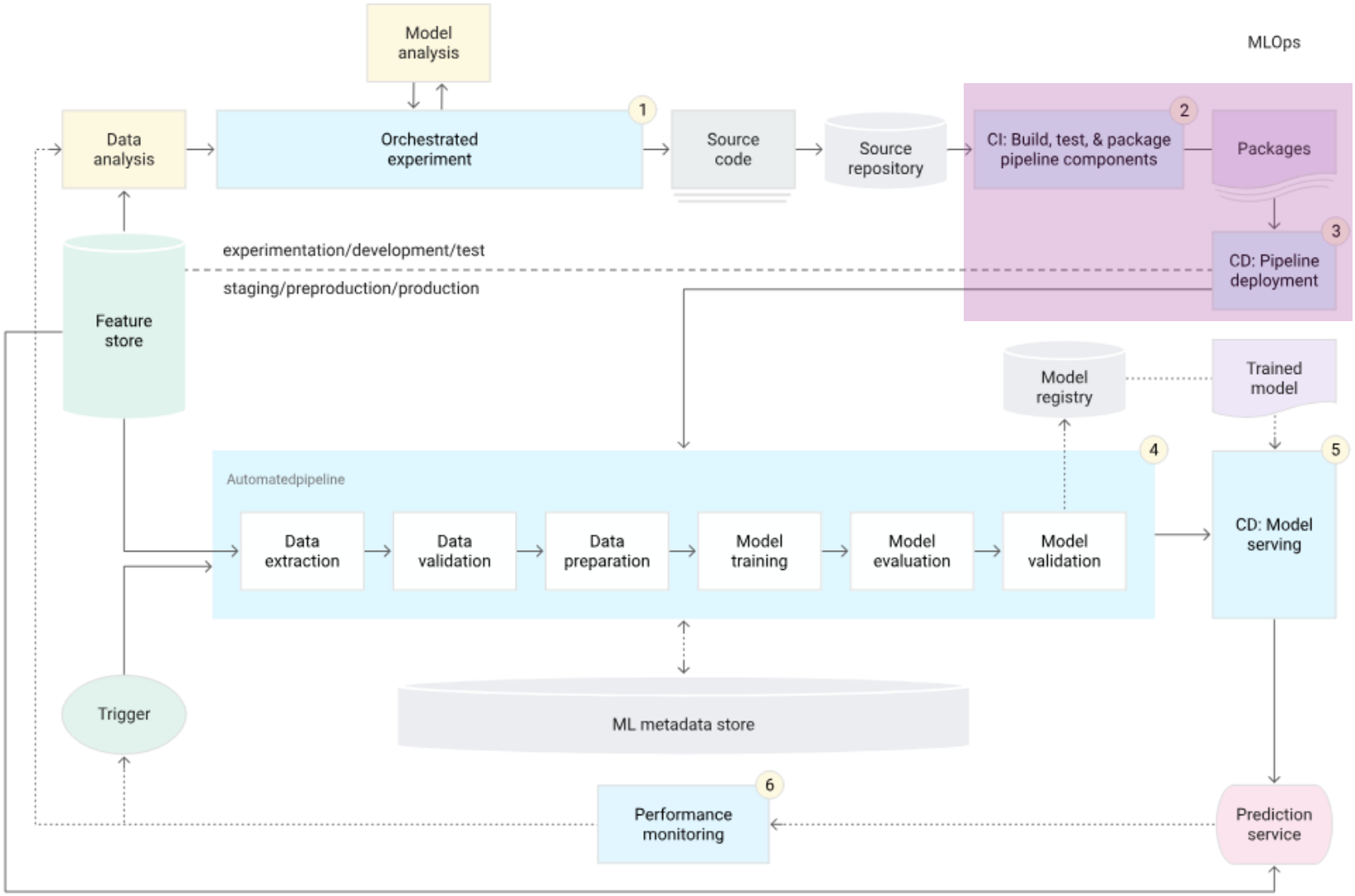


https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

MLOps level 2: CI/CD pipeline automation

Rapid and reliable update of the pipelines in production = robust automated CI/CD system

- Source control
- Test and build services
- Deployment services
- Model registry
- Feature store
- ML metadata store
- ML pipeline orchestrator



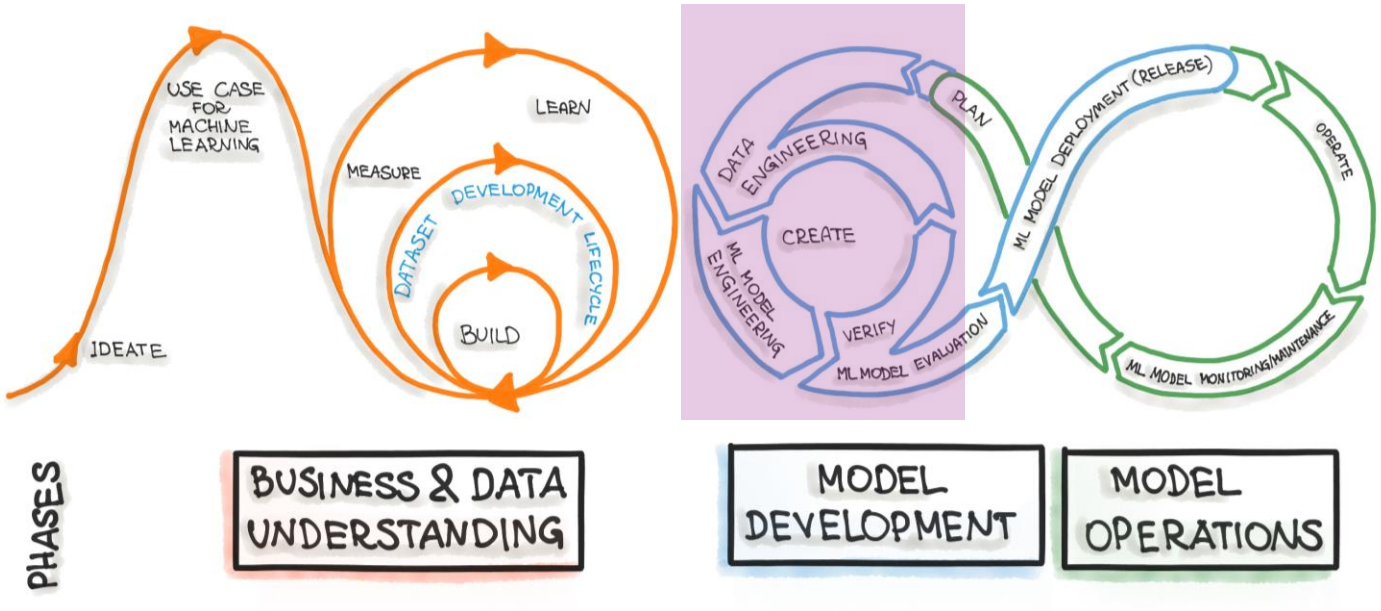
https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

ESTIA INSTITUTE OF TECHNOLOGY

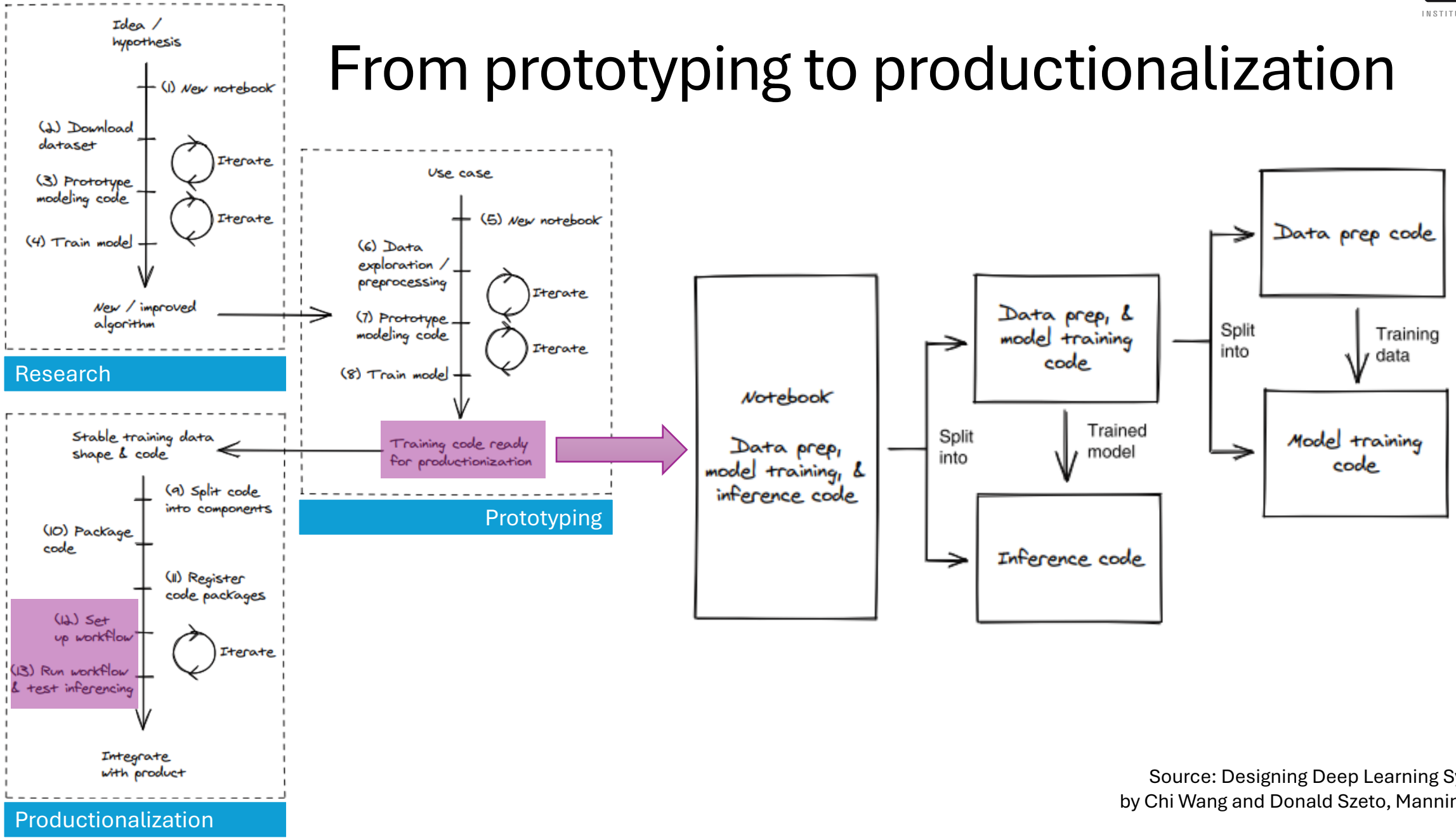
ML model training pipeline

CRISP-ML(Q): ML model training pipeline

- Data Engineering: data ingestion and cleaning, feature engineering
- Model Engineering: model training, hyperparameters tuning, model serialization
- Model Evaluation: model performance validation, deciding whether to deploy the model

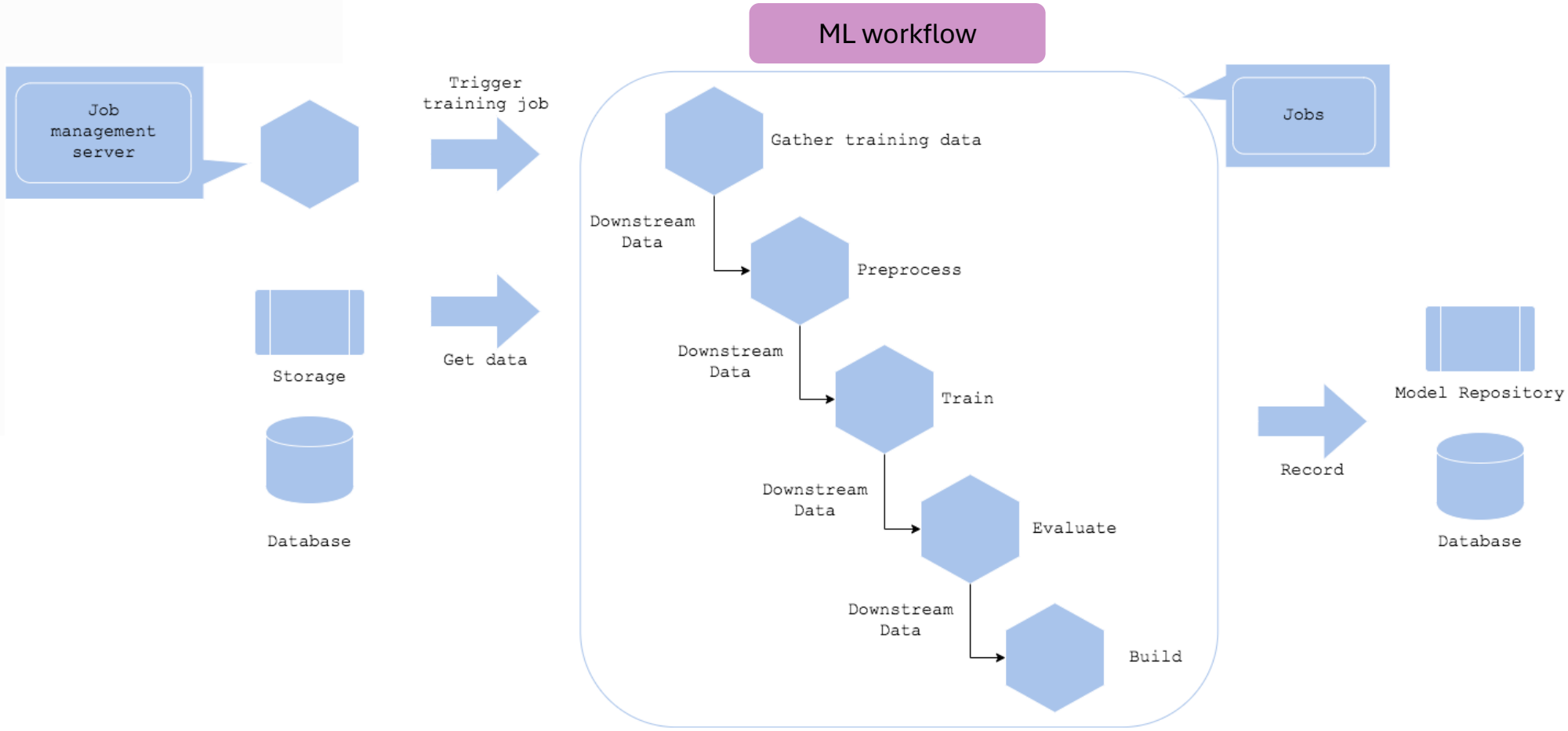


From prototyping to productionalization



Source: Designing Deep Learning Systems, by Chi Wang and Donald Szeto, Manning, 2023

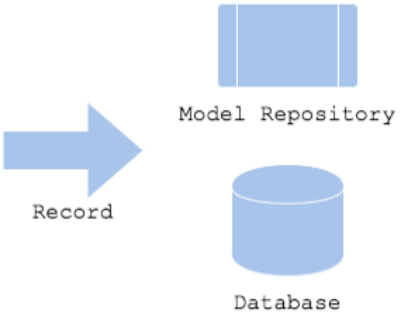
ML model training pipeline



https://github.com/mercari/ml-system-design-pattern/blob/master/Training-patterns/Pipeline-training-pattern/design_en.md

Persisting ML models: serialization

- Pickle: a Python library for object serialization. When pickling an object, you must unpickle it using the same Python and Pickle version. However, Pickle files can be a security risk, as they can execute arbitrary code upon unpickling, making them unsuitable for untrusted environments.
- Joblib: a Python library optimized for serializing large objects, particularly NumPy arrays and ML models. It extends Pickle with additional features like efficient compression, parallel processing, and caching, making it a great choice for ML pipelines.
- JSON: a standardized text-based serialization format. Typically used for storing hyperparameters, model metadata, or lightweight model representations. Many ML libraries, such as scikit-learn and TensorFlow, provide built-in methods for exporting models to JSON.
- Mleap: a serialization framework designed for the Apache Spark and Scala ecosystems. It enables ML models trained in Spark to be exported and served efficiently in production environments outside of Spark.
- ONNX (Open Neural Network Exchange): a cross-platform format for representing ML models, enabling interoperability between different ML frameworks (e.g., PyTorch, TensorFlow, and scikit-learn). While ONNX is highly portable, it is primarily optimized for deep learning models rather than traditional ML algorithms.



ESTIA
INSTITUTE OF TECHNOLOGY

Basic environment setup

Virtual environments, requirements, configuration

Virtual environments

- The main purpose of virtual environments is to manage dependencies (python libraries) of a particular project and isolate it from other projects.
- Good practice: one environment per project
- Why?
  - Installing packages globally usually requires elevated privileges (such as sudo, root, or administrator), which is a huge security risk.
  - Depending on how you installed Python, it can mess with the existing packages that are installed
  - It can break other projects. Many projects try their best to remain backward compatible, but every install could pull in new/updated dependencies that could break compatibility with other packages and projects.

Virtual environments

- Tools: *choose one depending on needs, team standards, etc.*
  - venv: **built-in tool**. It creates a folder with a copy of a specific interpreter. Packages installed in this folder are isolated from other workspaces.
  - conda: managed by conda package manager
  - and others (virtualenv, pipenv, poetry, ...)
- pip is the package installer for Python
- Python Virtual Environments: A Primer
- If the virtual environment folder is inside the project directory, that folder should be added to .gitignore file.
- Creating virtual environment in PyCharm

https://docs.python.org/3/library/venv.html

Create a virtual environment

```
python -m venv ENV_DIR
PS> python -m venv venv\
```

Activate the environment

```
PS> venv\Scripts\activate
(venv) PS>
```

Install Packages Into It

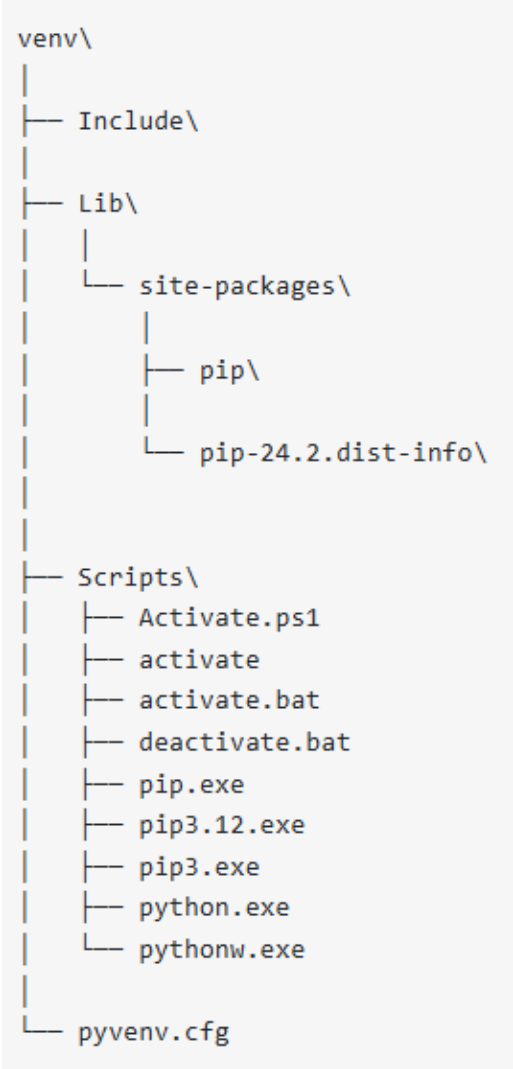
```
(venv) PS> python -m pip install
<package-name>
```

Deactivate It

```
(venv) PS> deactivate
PS>
```

Virtual environments

- Tools: *choose one depending on needs, team standards, etc.*
  - venv: **built-in tool**. It creates a folder with a copy of a specific interpreter. Packages installed in this folder are isolated from other workspaces.
  - conda: managed by conda package manager
  - and others (virtualenv, pipenv, poetry, ...)
- pip is the package installer for Python
- Python Virtual Environments: A Primer
- If the virtual environment folder is inside the project directory, that folder should be added to .gitignore file.
- Creating virtual environment in PyCharm



requirements.txt

Requirements files (documentation) serve as a list of items to be installed by pip, when using pip install.

Create requirements file from the packages installed in venv

```
(venv) pip freeze -l > requirements.txt
```

Install dependencies to venv

```
(venv) pip install -r requirements.txt
```

```
# This is a comment, to show how #-prefixed lines are ignored.
# It is possible to specify requirements as plain names.
pytest
pytest-cov
beautifulsoup4

# The syntax supported here is the same as that of requirement specifiers.
docopt == 0.6.1
requests [security] >= 2.8.1, == 2.8.* ; python_version < "2.7"
urllib3 @ https://github.com/urllib3/urllib3/archive/refs/tags/1.26.8.zip

# It is possible to refer to other requirement files or constraints files.
-r other-requirements.txt
-c constraints.txt

# It is possible to refer to specific local distribution paths.
./downloads/numpy-1.9.2-cp34-none-win32.whl

# It is possible to refer to URLs.
http://wxpython.org/Phoenix/snapshot-builds/wxPython_Phoenix-
3.0.3.dev1820+49a8884-cp34-none-win_amd64.whl
```

Configuration files

(Microsoft Windows) INI files: consist of sections, each of which contains a set of key-value pairs

```
[PATHS]
DB_PATH=C:/sqlite/databases/diabetes.db
MODEL_PATH=C:/dev/models/diabetes.model
[ML]
RANDOM_STATE=42
```

Using

```
from configparser import ConfigParser
config = ConfigParser()
config.read("../config.ini")
DB_PATH = config.get("PATHS", "DB_PATH")
```

`[PATHS]` is a section.
`RANDOM_STATE=42` is a key-value pair.

No data hierarchy can be represented.
Good for simple configurations.

YAML: YAML Ain't Markup Language (*data-oriented, rather than document markup*; originally *Yet Another Markup Language*) is a human-readable data serialization format.

Set of indentation-based rules is used to define the structure of data.

```
paths:
  db_path: C:/sqlite/databases/diabetes.db
  model_path: C:/dev/models/diabetes.model
ml:
  random_state: 42
```

Using

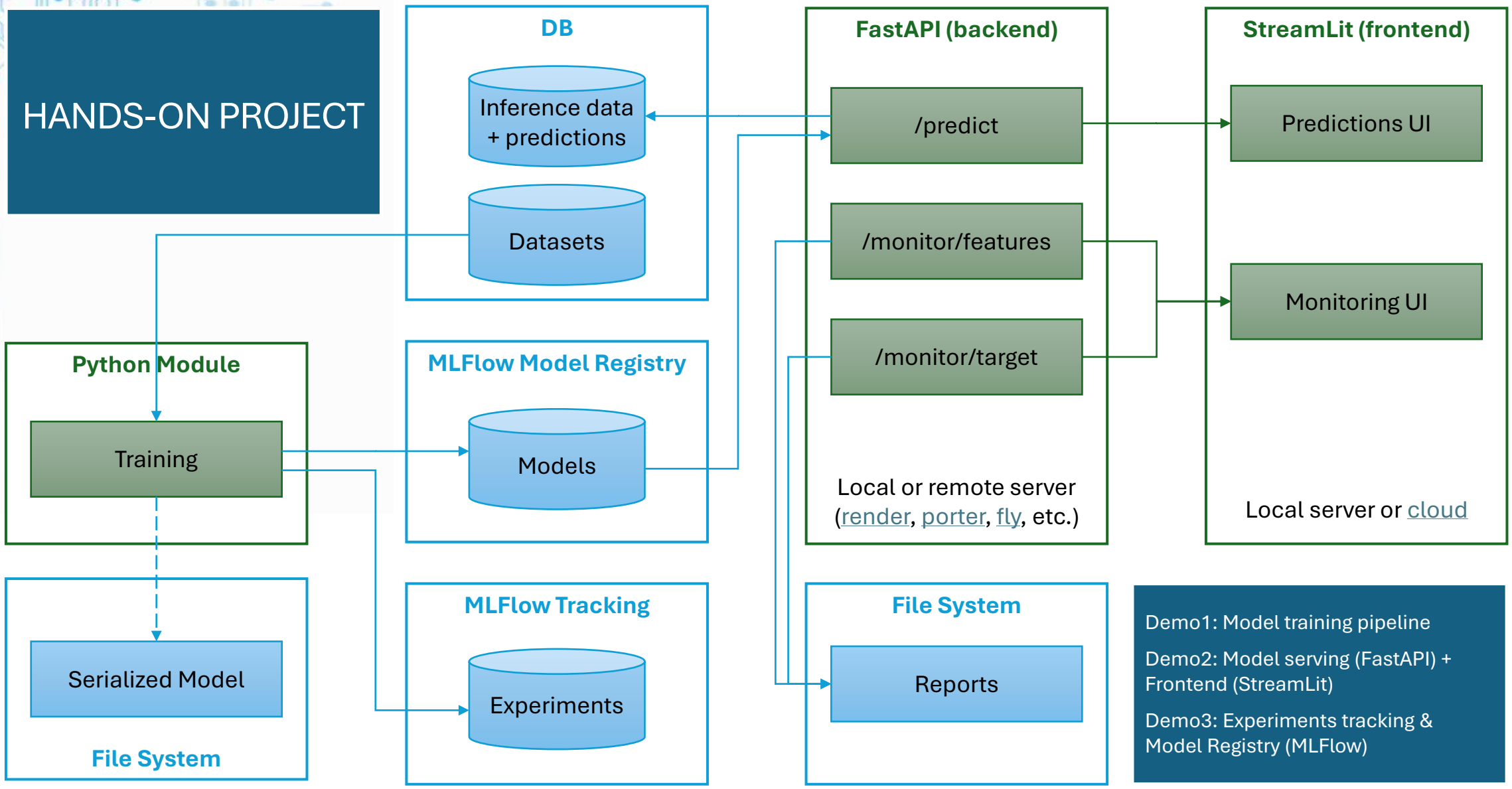
```
import yaml
with open("config.yml", "r") as f:
    config_yaml = yaml.load(f, Loader=yaml.SafeLoader)
    DB_PATH = str(config_yaml['paths']['db_path'])
    MODEL_PATH = str(config_yaml['paths']["model_path"])
    RANDOM_STATE =
int(config_yaml["ml"]["random_state"])
```

Widely used in DevOps and Cloud.

ESTIA
INSTITUTE OF TECHNOLOGY

Hands-on project: MLOps level 0

HANDS-ON PROJECT



INSTITUTE OF TECHNOLOGY

Demo #1
ML model training pipeline

https://github.com/eishkina-estia/bihar-diabetes/

HANDS-ON PROJECT

Environment setup

- IDE of your choice (PyCharm, VSCode, ...) + Python interpreter
  - You can get PyCharm Professional License using your student email
  - Configure Python interpreter: instructions for PyCharm
- Anaconda or Miniconda distribution (for conda environments)
  - Installation instructions: Anaconda or Miniconda
  - The Miniconda installers are on the same page as the Anaconda Distribution installers, past registration: anaconda.com/download

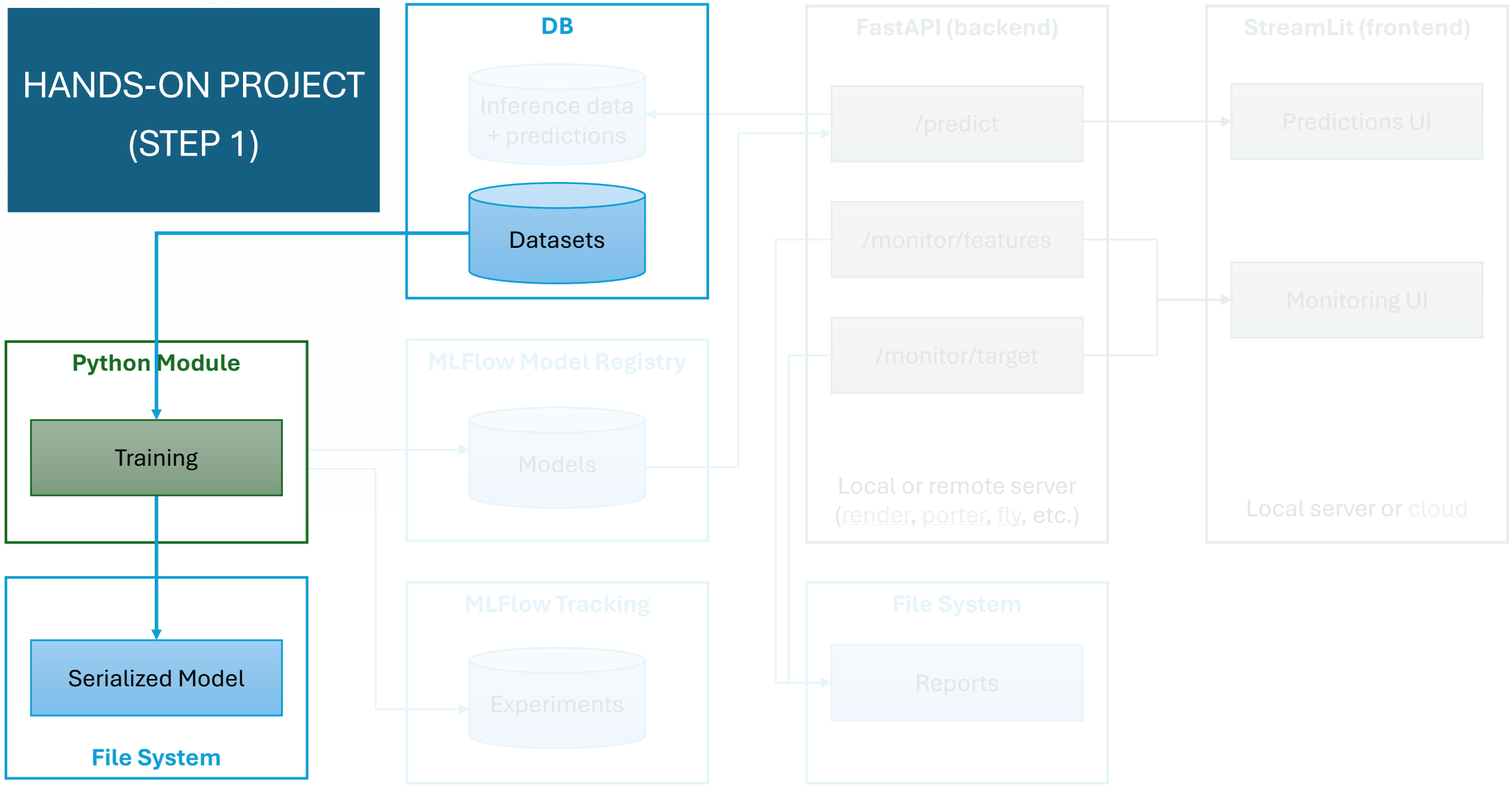
Exercise

- Clone repository bihar-diabetes
- Create a virtual environment and install all the dependencies (from requirements.txt)
- Run notebooks/EDA.ipynb (Jupyter Notebook interface)
- Load data into a db
  `python -m data.download_data`
  - data/diabetes.db file is created. It is a lightweight disk-based database with 2 tables: train and test.
- Train and persist model
  `python -m model.train`
  - models/diabetes.model file is created. It a serialized regression model.
- Test inference
  `python -m model.evaluate`
  - It uses the previously serialized model.

HANDS-ON PROJECT
Step #1: training pipeline

- Create a project using IDE of your choice for the NYC taxi trip duration prediction use case (model trained in step 4.6 of the notebook)
- Create a virtual environment, requirements.txt and config file
- Create a github repo for the project. Add readme and .gitignore file
- Implement python modules for preprocessing and training logic.
- Check the inference (predictions) using the model trained in the previous step.

HANDS-ON PROJECT (STEP 1)



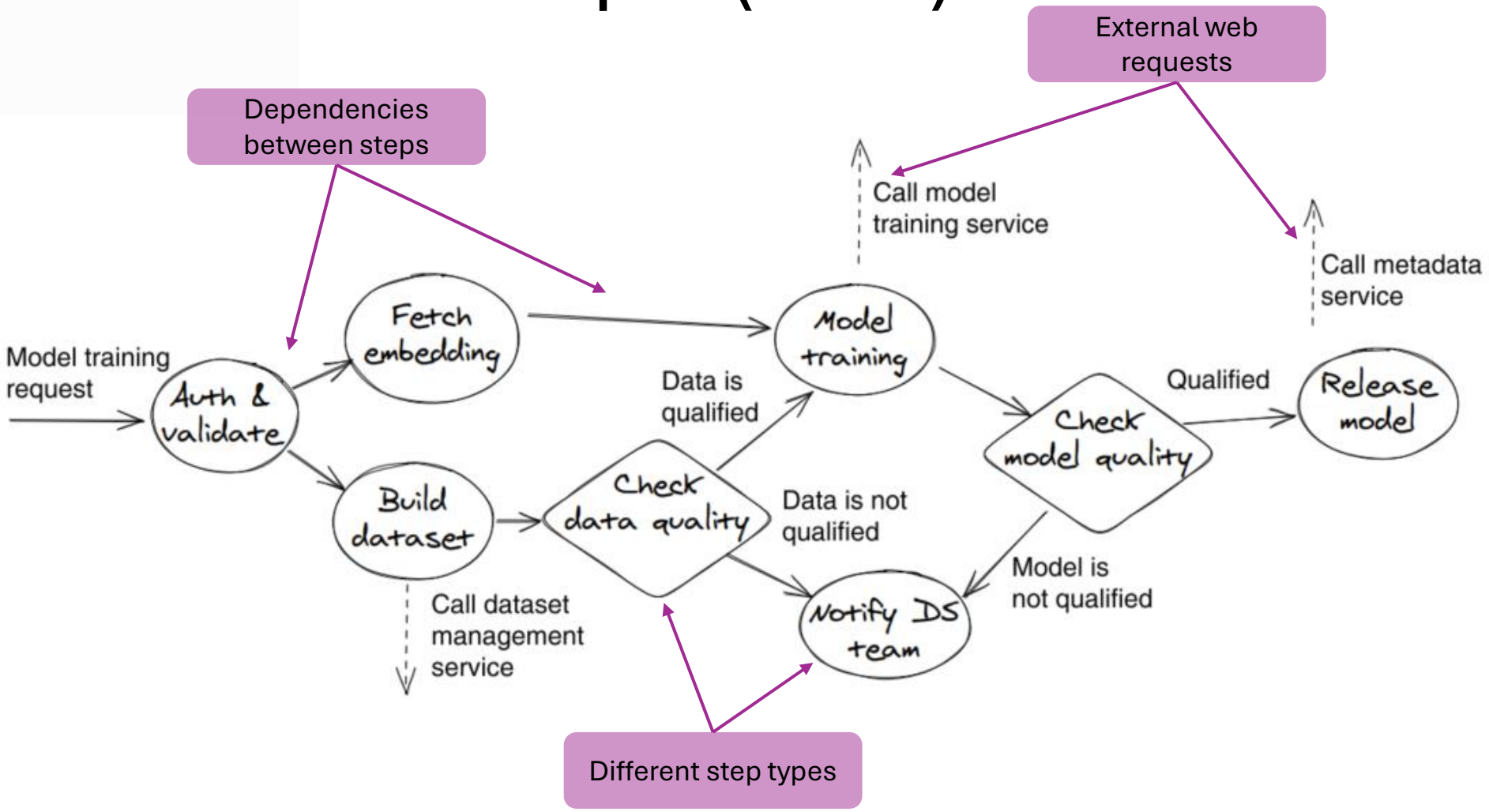
ESTIA
INSTITUTE OF TECHNOLOGY

Workflow orchestration

ML workflow orchestration

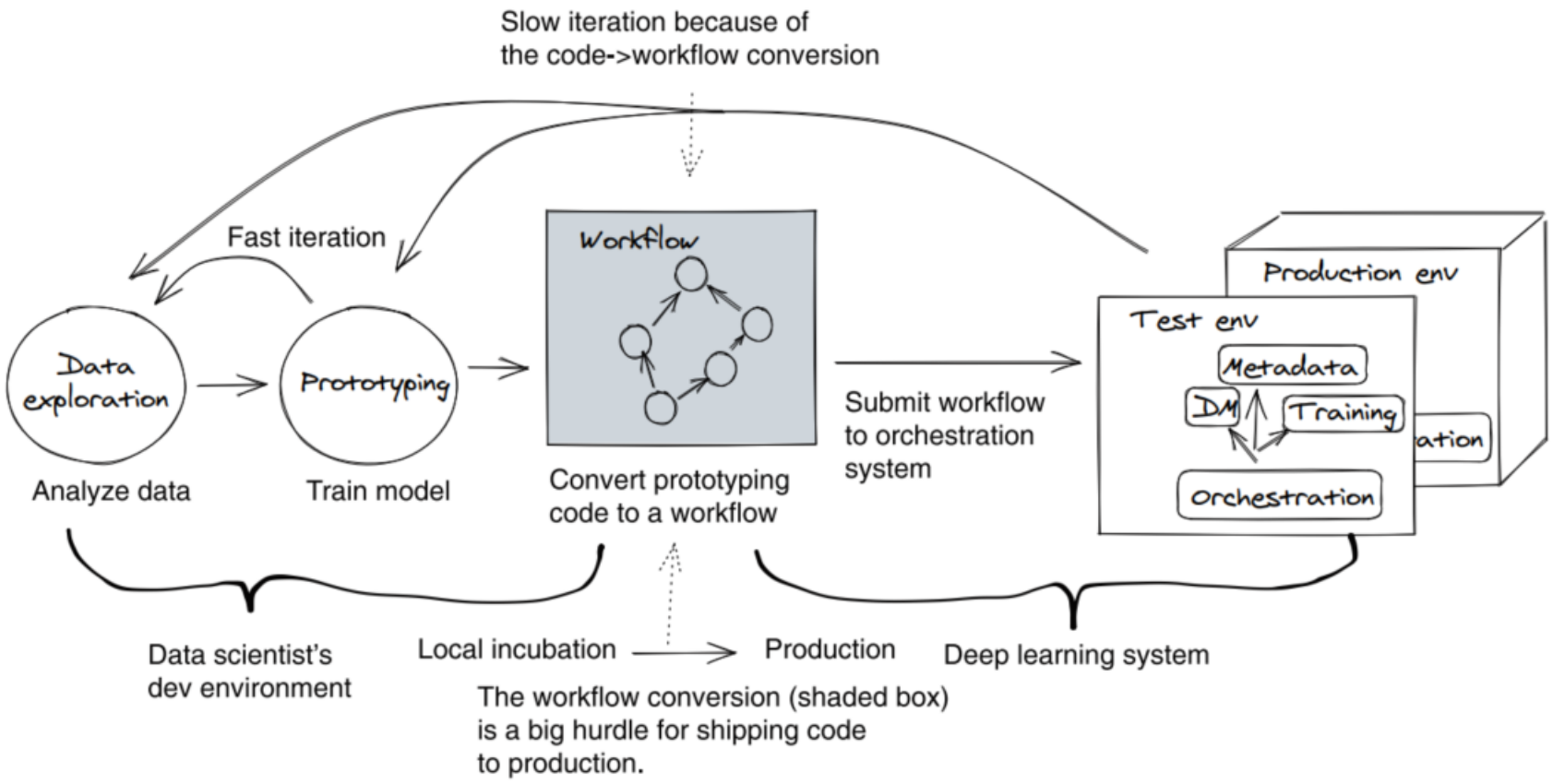
- Workflow is a sequence of operations that are part of some larger task. It can be performed manually or can be automated.
- A workflow for training a machine learning model (ML training pipeline) typically consists of the following tasks: *fetching raw data, rebuilding the training dataset, training the model, evaluating the model performance, and deploying the model*.
- A workflow orchestration system is built to manage workflow life cycles, including workflow creation, execution, and troubleshooting, ensuring reliable and scalable automation.
- A workflow can be viewed as a directed acyclic graph (DAG) with steps (tasks) as nodes. A step is the smallest resumable computational unit. DAG edges specify dependencies between steps and execution order.
- To guarantee a workflow can be completed under any condition, its execution graph needs to be a DAG, which prevents the workflow execution from falling into a dead loop.

ML workflow example (DAG)



Source: Designing Deep Learning Systems, by Chi Wang and Donald Szeto, Manning, 2023

ML development process

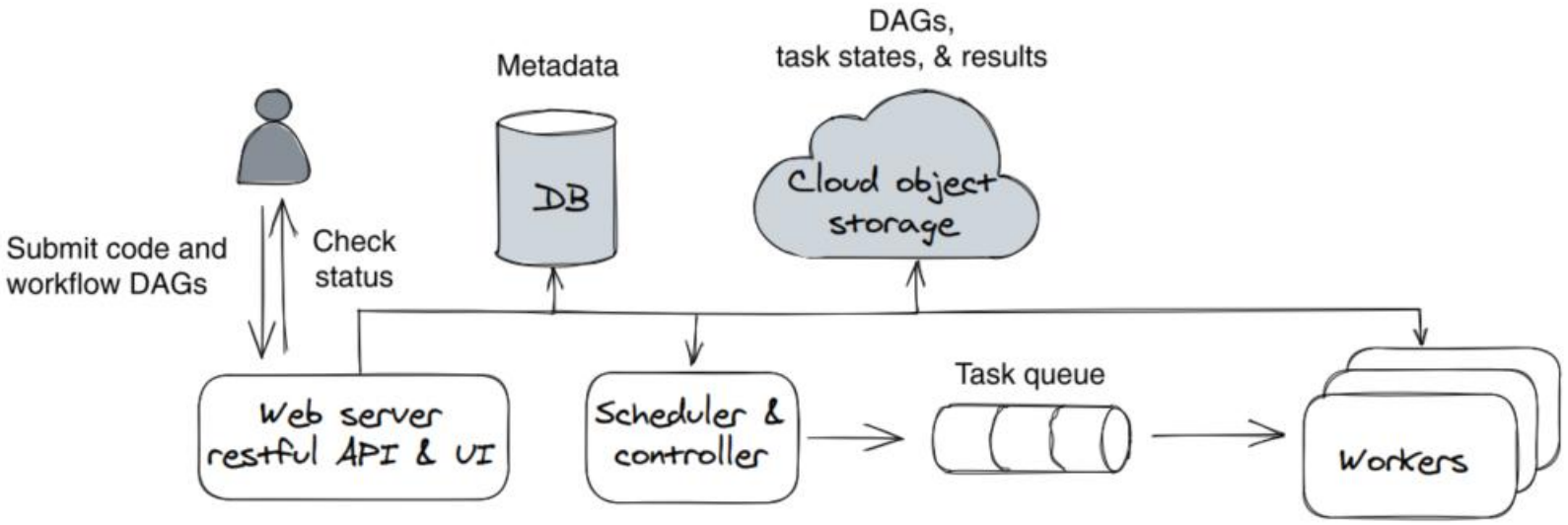


Source: Designing Deep Learning Systems,
by Chi Wang and Donald Szeto, Manning, 2023

A general orchestration system design

Metadata database stores the workflows' configuration, DAG, editing and execution history, and the tasks' execution state.

Object store is shared file storage, typically built on top of cloud object storage, such as Amazon S3. One usage of an object store is task output sharing. When a worker runs a task, it reads the output value of the previous task from the object store; the worker also saves the task output to the object store for its successor tasks.



Workers provide the compute resource to run workflow tasks. They abstract the infrastructure and are agnostic to the task that's running.

Web server presents a web user interface and a set of web APIs for users to create, inspect, trigger, and debug the behavior of a workflow.

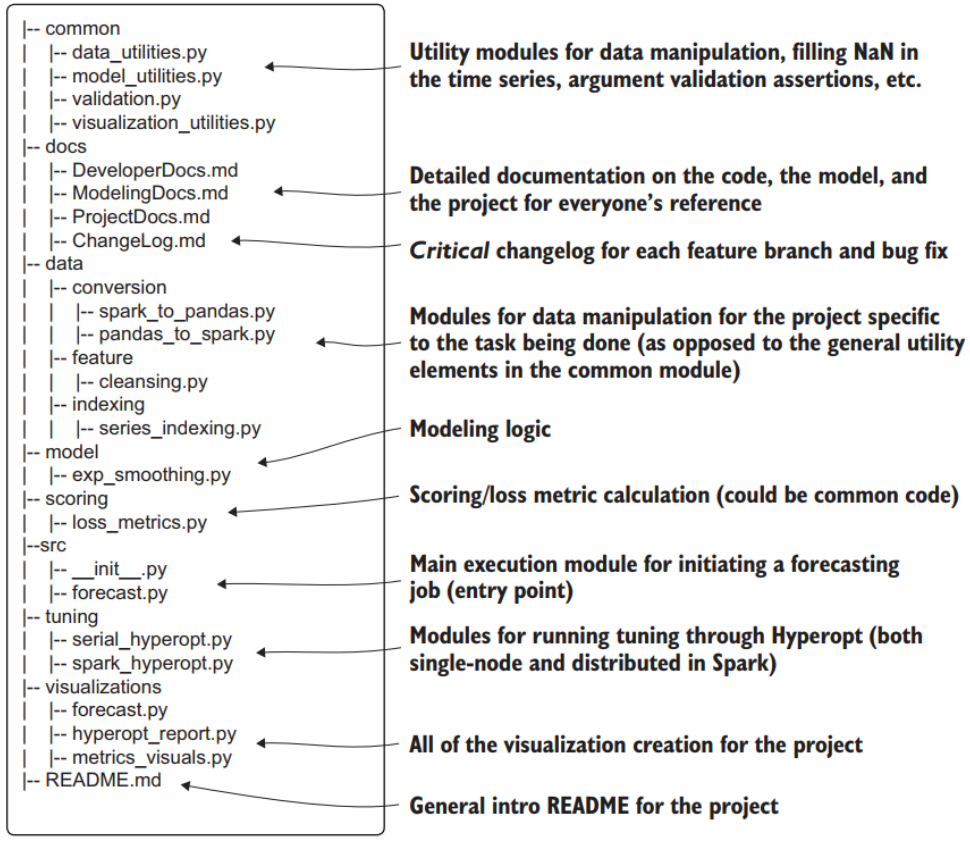
Scheduler and controller are usually implemented together because they are all related to workflow execution. First, the scheduler watches every active workflow in the system, and it schedules the workflow to run when the time is right. Second, the controller dispatches the workflow tasks to workers.

Workflow orchestration systems

- Airflow (widely used tool)
- Argo Workflows (Kubernetes-native tool)
- Metaflow (“designed for data scientists”)
- Prefect (modern alternative to Airflow that simplifies workflow orchestration)
- Kubeflow Pipelines (Kubernetes-native tool)
- Mage (great especially for ETL tasks)

Machine learning system design

- Design patterns: https://github.com/mercari/ml-system-design-pattern/
- Cookiecutter Data Science: A logical, reasonably standardized, but flexible project structure for doing and sharing data science work.
- Kedro is a toolbox for production-ready data science (Kedro starters).



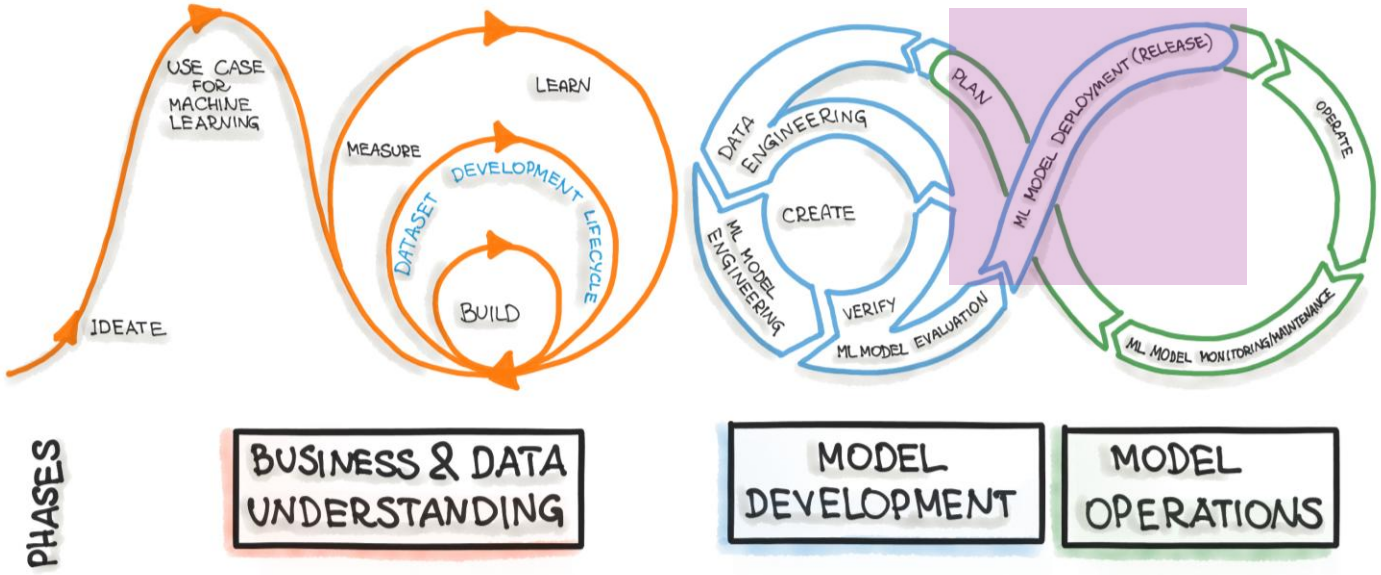
Source: Machine Learning Engineering in Action, by Ben Wilson, Manning, 2022

ESTIA
INSTITUTE OF TECHNOLOGY

Model deployment and serving

CRISP-ML(Q): ML Model Deployment and Serving

- Build microservices from the ML model
- Evaluate model under production condition
- Assure user acceptance and usability
- Model governance
- Deploy according to the selected strategy



Model serving patterns

- **Batch**
- Asynchronous
- Synchronous

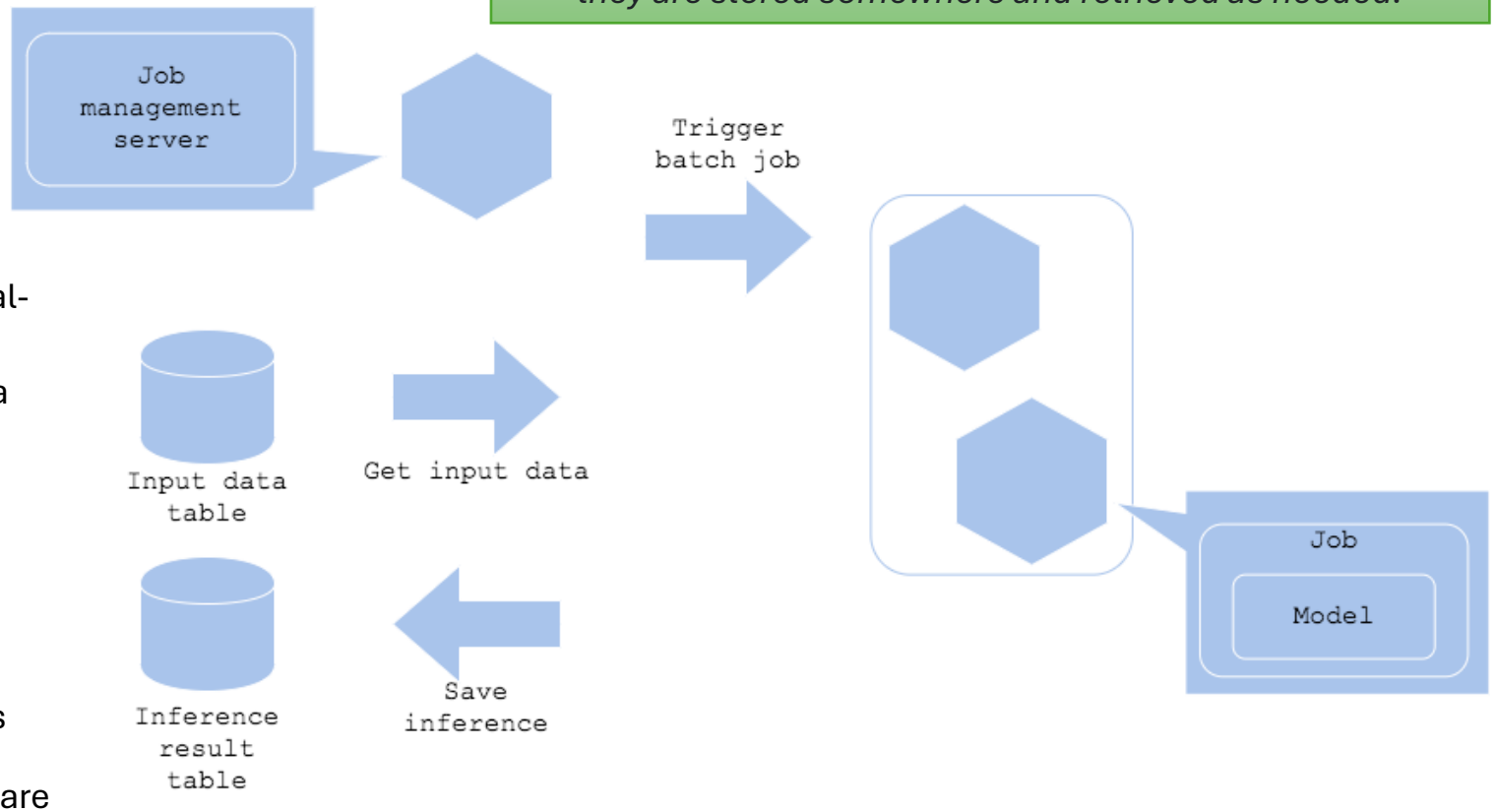
Usecase

- If there is no need for (near-)real-time prediction.
- If predictions are expected on a bulk of data.
- If running predictions is schedulable; hourly, daily, weekly,...

Example

Netflix might generate movie recommendations for all its users every four hours, and the precomputed recommendations are fetched and shown to users when they log on to Netflix.

Predictions are generated periodically or whenever triggered, they are stored somewhere and retrieved as needed.

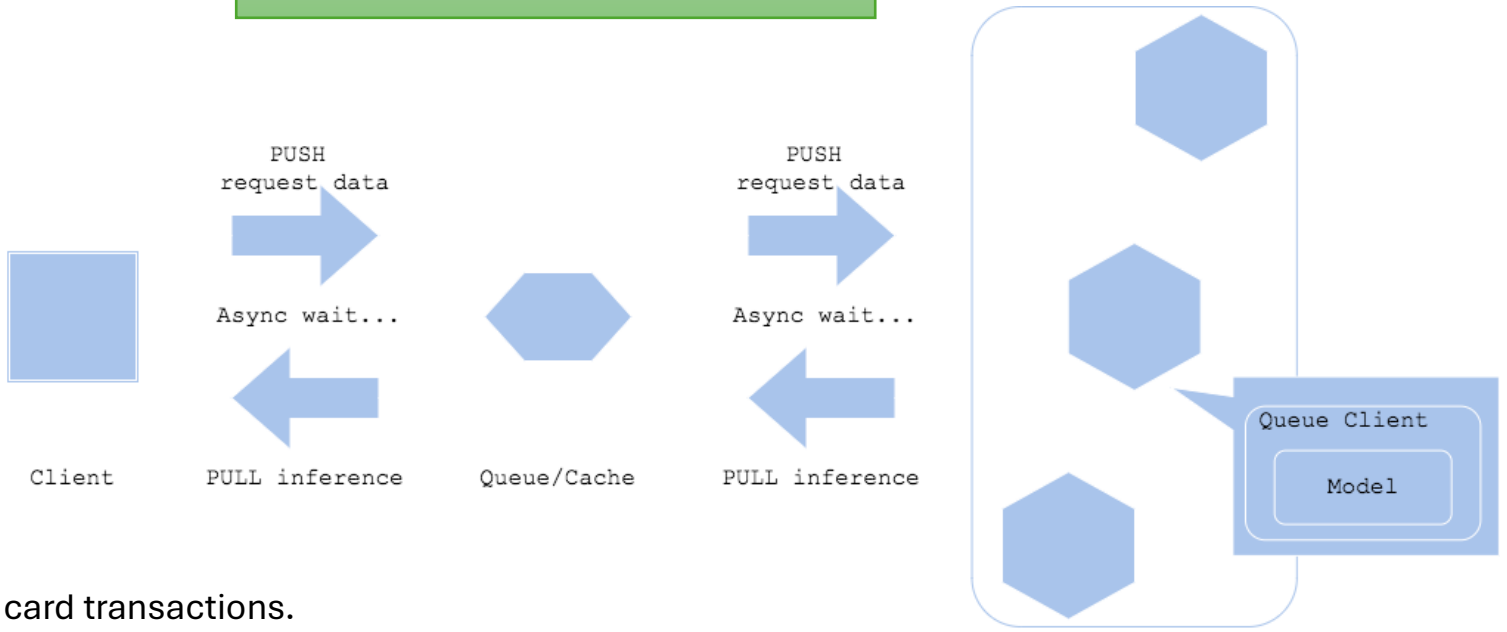


https://github.com/mercari/ml-system-design-pattern/tree/master/Serving-patterns

Model serving patterns

- Batch
- **Asynchronous**
- Synchronous

Separation of a prediction request and prediction retrieval



Usecase

- When the immediate process does not depend on the prediction.
- To separate the client which is making the prediction request and the destination where response is expected.

Example

A fraud detection system for credit card transactions. When a transaction occurs, a request is sent to a fraud detection model, which processes it in the background. The transaction continues, and once the model finishes its analysis, the result is logged or triggers an alert if fraud is suspected.

https://github.com/mercari/ml-system-design-pattern/tree/master/Serving-patterns

Model serving patterns

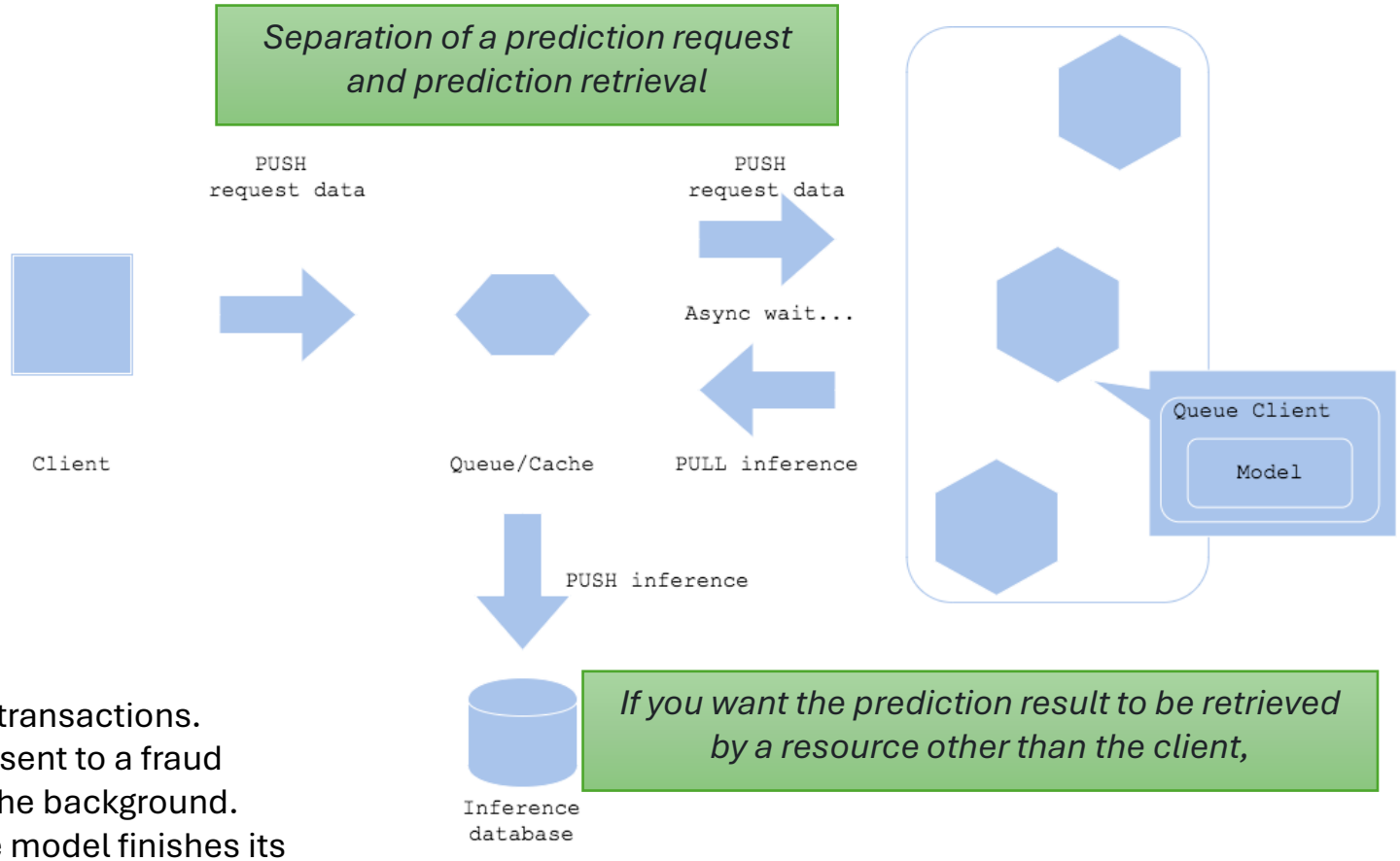
- Batch
- **Asynchronous**
- Synchronous

Usecase

- When the immediate process does not depend on the prediction.
- To separate the client which is making the prediction request and the destination where response is expected.

Example

A fraud detection system for credit card transactions. When a transaction occurs, a request is sent to a fraud detection model, which processes it in the background. The transaction continues, and once the model finishes its analysis, the result is logged or triggers an alert if fraud is suspected.

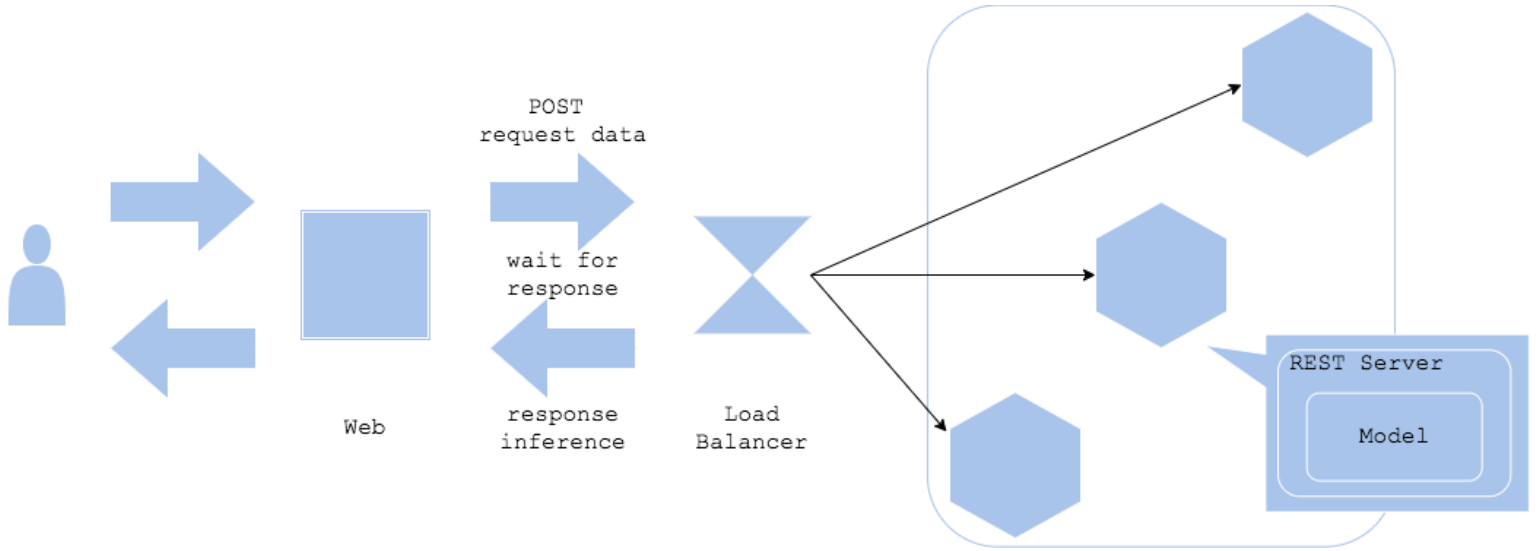


https://github.com/mercari/ml-system-design-pattern/tree/master/Serving-patterns

Model serving patterns

- Batch
- Asynchronous
- **Synchronous**

Run prediction synchronously: block the workflow until the prediction finishes.



Usecase

- When in your business logic, the model inference is a blocker to proceed to the next step.
- When your workflow depends on the inference result.

Example

A virtual assistant like Siri or Google Assistant. When a user asks a question, the request is sent to an ML model, which processes it in real time and immediately returns a response (e.g., an answer, action, or search result).

https://github.com/mercari/ml-system-design-pattern/tree/master/Serving-patterns

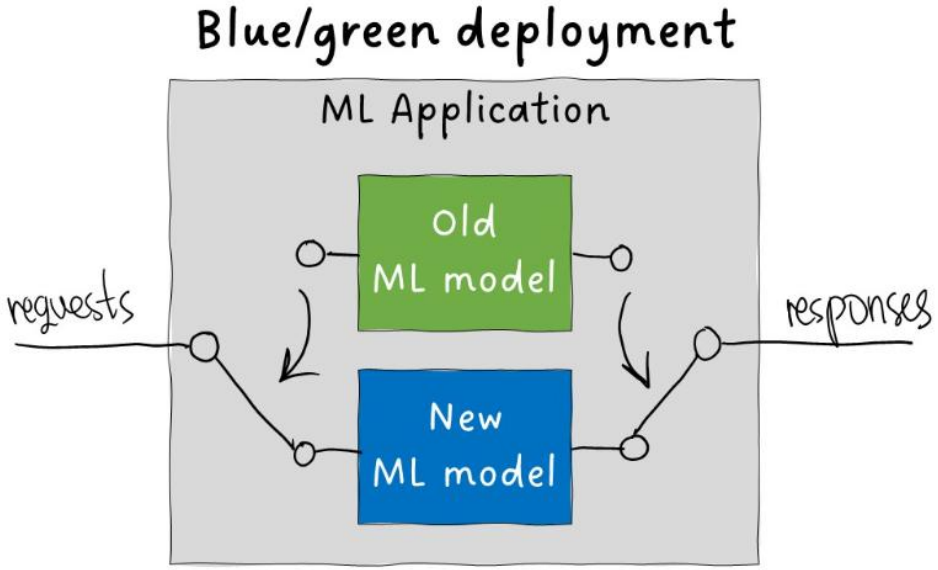
Model Deployment: strategies

Blue-Green Deployment

- Two identical production environments (only one active) + load balancer.
- Updates are deployed to inactive version, and once it is successfully upgraded, traffic is switched to it.

- Pros
  - Simplicity and quick rollbacks
- Cons:
  - High risk if the new model finally does not work as expected

https://earthly.dev/blog/deployment-strategies/

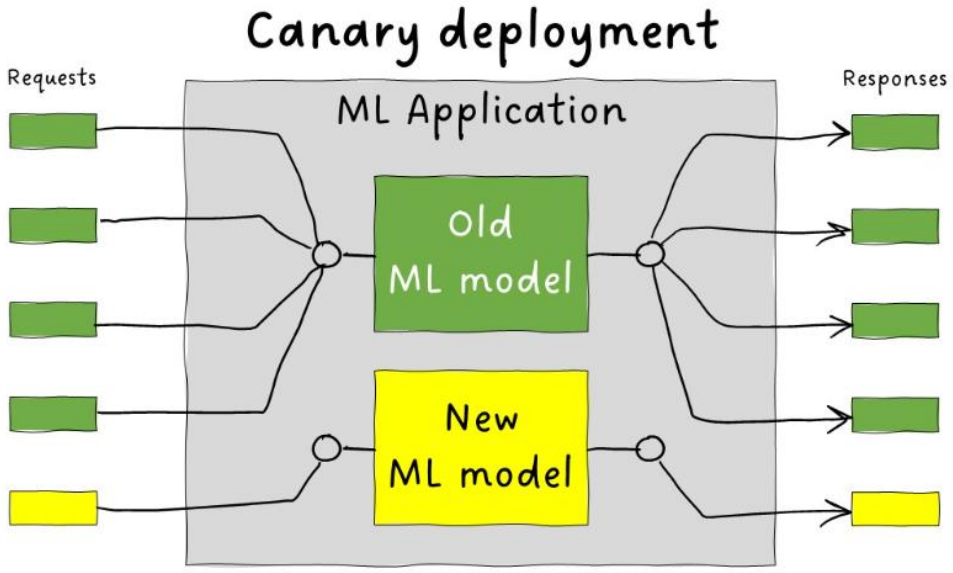


https://campus.datacamp.com/courses/mlops-deployment-and-life-cycling/

Model Deployment: strategies

Canary Deployment

- Similar to blue-green, but instead of switching all traffic over to the new version, only a part of it is sent: like a canary in the coal mine.
- If everything looks OK, request volume is slowly ramped up until its being entirely served by the new version.
- Pros
  - Catch problems early
- Cons
  - Canary users bear the brunt of production issues



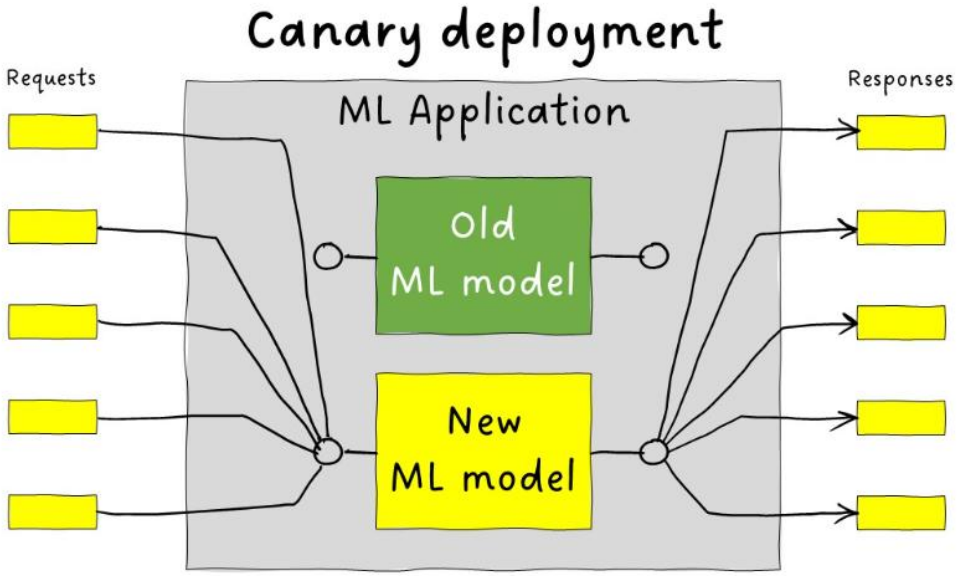
https://earthly.dev/blog/deployment-strategies/

https://campus.datacamp.com/courses/mlops-deployment-and-life-cycling/

Model Deployment: strategies

Canary Deployment

- Similar to blue-green, but instead of switching all traffic over to the new version, only a part of it is sent: like a canary in the coal mine.
- If everything looks OK, request volume is slowly ramped up until its being entirely served by the new version.
- Pros
  - Catch problems early
- Cons
  - Canary users bear the brunt of production issues



https://earthly.dev/blog/deployment-strategies/

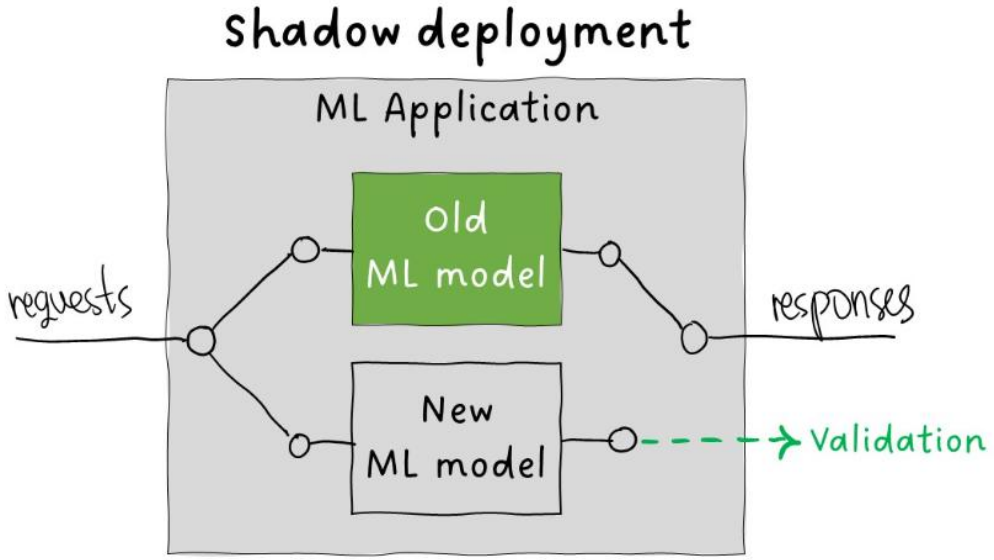
https://campus.datacamp.com/courses/mlops-deployment-and-life-cycling/

Model Deployment: strategies

Shadow Deployment

- The new version of the service is started, and all traffic is mirrored by the load balancer. The requests are sent to the current version and the new version, but all responses come only from the existing stable version.
- Pros
  - Safest strategy: catch production problems without custom impact
- Cons
  - Performance burden: two executions for each requests

https://earthly.dev/blog/deployment-strategies/

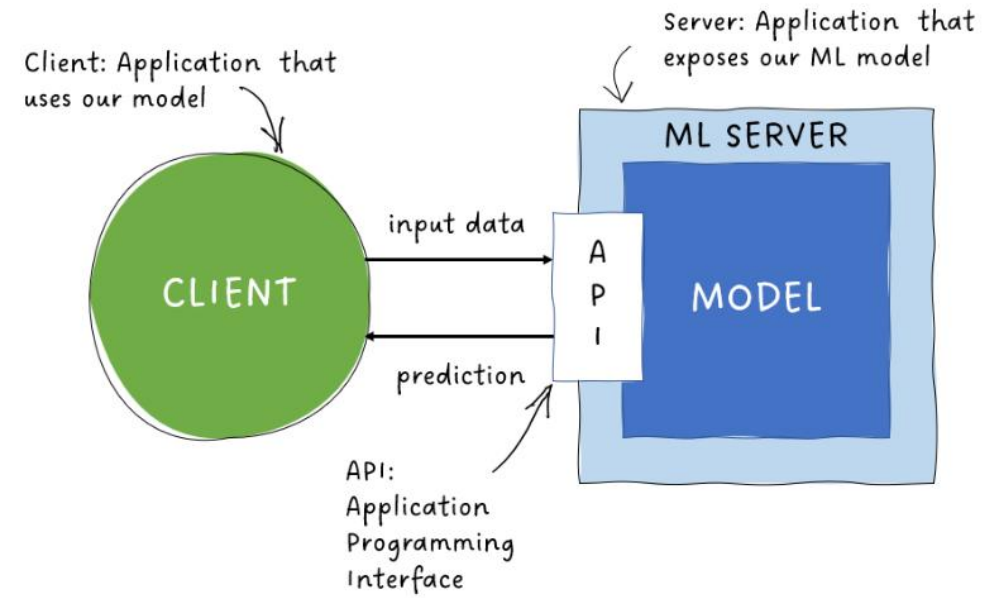


Potential hybrid solutions:

- Use shadow model on a fraction of requests (« canary shadow »)
- Run shadow model offline, outside of peak hours.

https://campus.datacamp.com/courses/mlops-deployment-and-life-cycling/

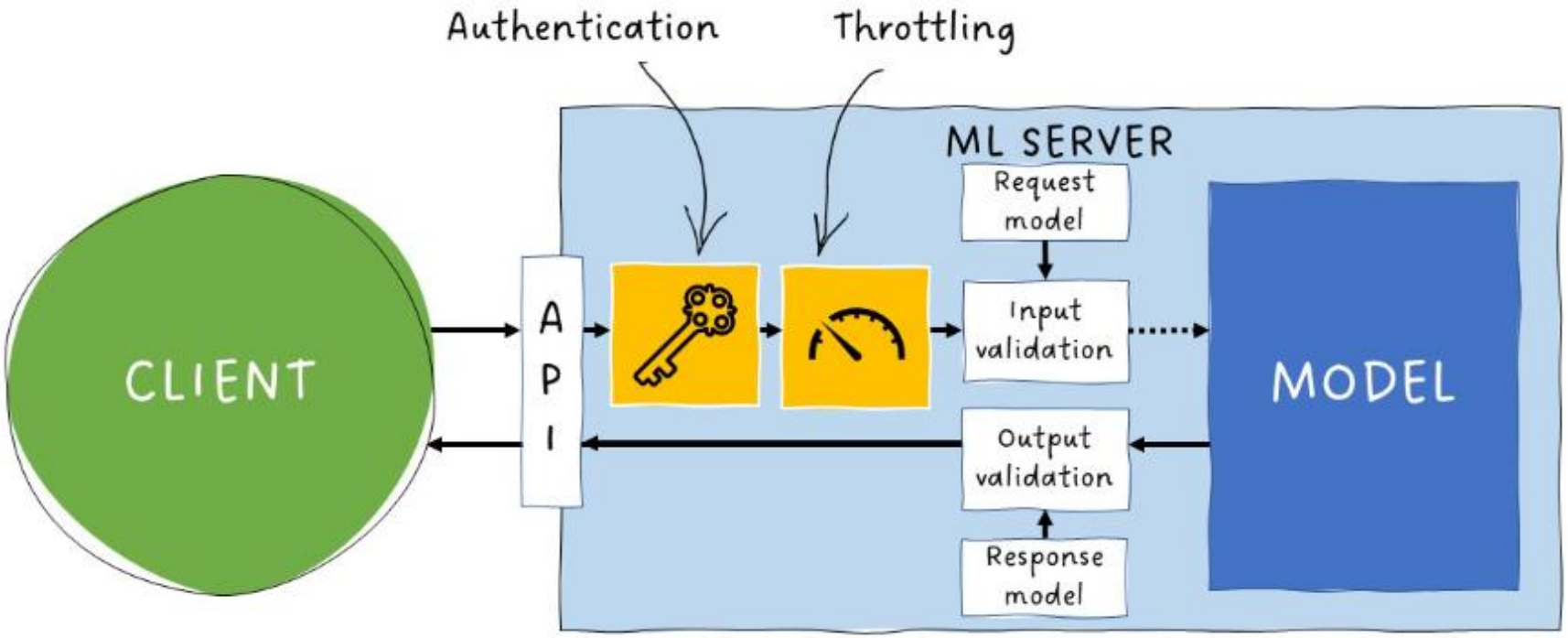
Model Serving: API



- Intermediary software that allows two applications to talk to each other.
- Contract between the given application and other applications saying if the user software provides input in a pre-defined format, the API will provide the output to the user.
- End-point where you host the trained machine learning models (pipelines) for use.
- Example of an API architecture: REST (REpresentational State Transfer)

https://campus.datacamp.com/courses/mlops-deployment-and-life-cycling/

Model Serving: API

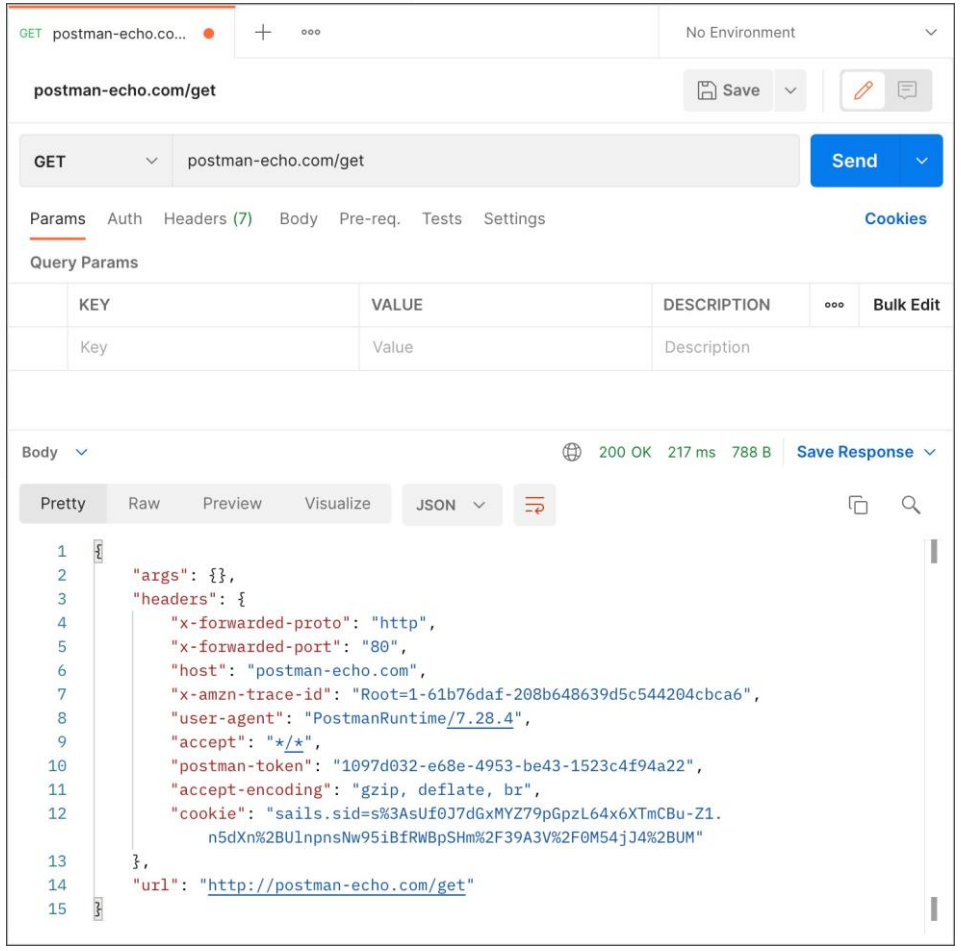


https://campus.datacamp.com/courses/mlops-deployment-and-life-cycling/

Model Serving: API

	Django	Flask	FastAPI
Performance	Django is a full-featured framework with many built-in features.	Flask is a lightweight micro-framework that performs better than Django, but lacks built-in async support.	FastAPI is one of the fastest web frameworks, thanks to its native async support and efficient request handling.
Async support	Limited async support via ASGI but not fully optimized for async operations.	No native support: deployed on WSGI (Python Web Server Gateway Interface)	Native async support: deployed on ASGI (Asynchronous Server Gateway Interface)
Ease of use	Django is massive and hence a bit complicated to learn.	Flask is easy to learn and pretty straightforward in use.	FastAPI is the simplest of all three.
Data Validation	No built-in validation	No built-in validation	Yes (based on pydantic)
		pip install flask	pip install fastapi[all]

Model Serving: API



Existing API platforms for building and using APIs:

- Postman: https://www.postman.com/
- Insomnia: https://insomnia.rest/

GET vs. POST when serving ML model predictions:

- A payload within a GET request message has no defined semantics; sending a payload body on a GET request might cause some existing implementations to reject the request.
- The response to a GET request is cacheable; a cache MAY use it to satisfy subsequent GET and HEAD requests unless otherwise indicated by the Cache-Control header field

Source: HTTP 1.1 2014 Spec

Model Serving: FastAPI

- FastAPI is a modern, high-performance web framework for building APIs with Python. It is very flexible and easy to get started.
- FastAPI is built on another Python library called pydantic that makes it easy to validate the correctness of inputs into the API.
- uvicorn is a Python-based web server that allows FastAPI to handle requests

```
pip install fastapi
pip install uvicorn[standard]
```

- Example

main.py

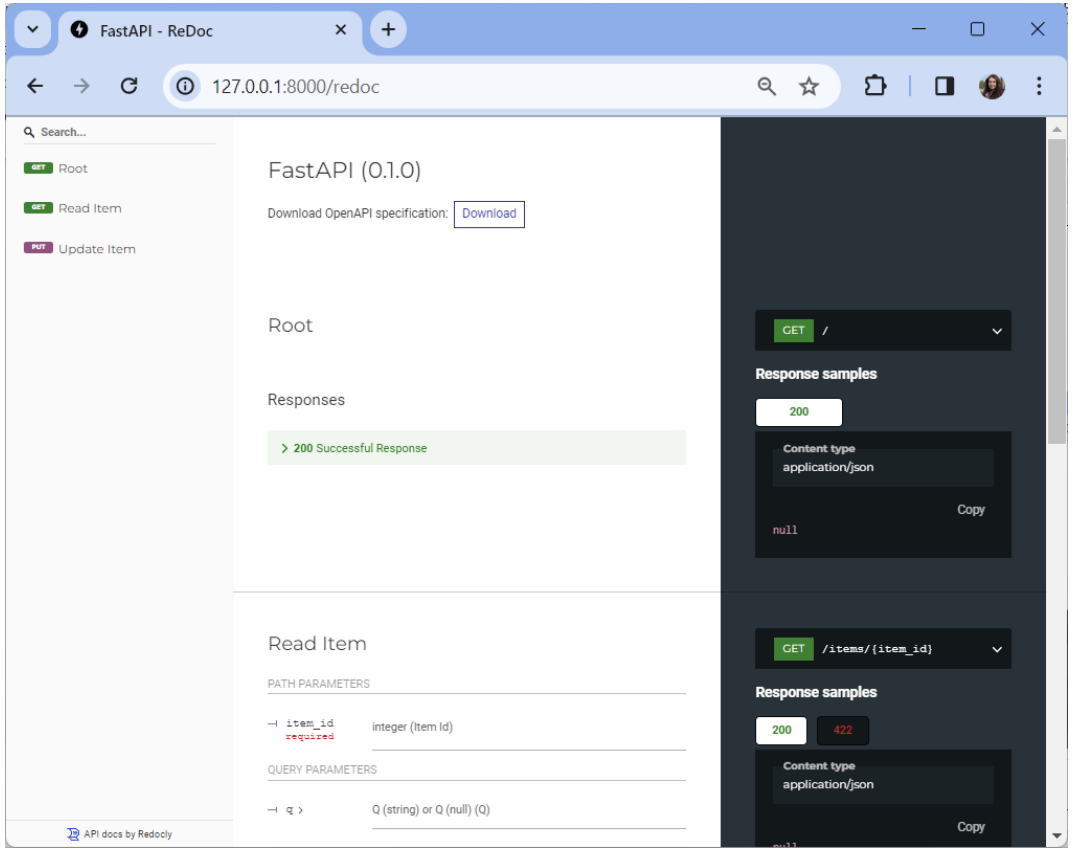
```python
from fastapi import FastAPI

app = FastAPI()

# Create a decorator specifying the URL endpoint
# for this API and indicating that GET requests
# should be used to hit the API
@app.get("/")
# Define a function that handles these GET requests
def root():
  # Return a message when a request hits the API
  return {"message": "Hello!"}

@app.get("/items/{item_id}")
def read_item(item_id: int):
  return {"item_id": item_id}

if(__name__) == '__main__':
    uvicorn.run("main:app", host = "0.0.0.0",
                port = 5000, reload = True)
```

Application will be available on http://127.0.0.1:8000

Model Serving: FastAPI

Swagger UI: 127.0.0.1/docs

ReDoc: 127.0.0.1/redoc

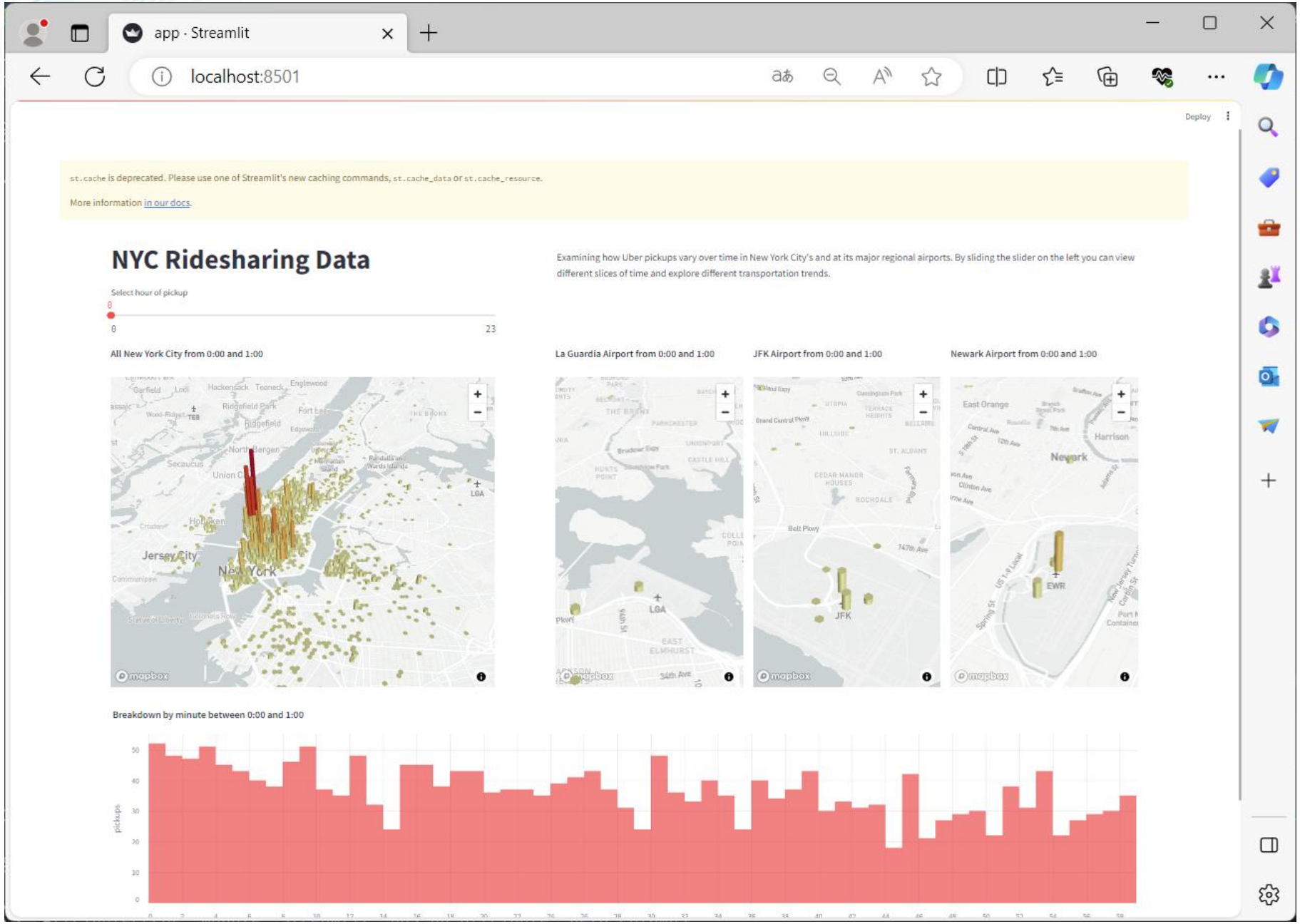


Streamlit

- https://streamlit.io/, a faster way to build and share data app
- Open-source Python library enabling data scientists to build attractive user interfaces in no time.
- Pros
  - No front-end (html, js, css) knowledge is required.
  - It is compatible with most Python libraries (e.g. pandas, matplotlib, seaborn, etc.).
  - Low code is needed to create amazing web apps.

- Get started
- App example: dashboards
- App gallery
- API reference
- Cheat sheet
- Installation and use (documentation)

```
pip install streamlit
streamlit hello
```


app · Streamlit
localhost:8501
Deploy
st.cache is deprecated. Please use one of Streamlit's new caching commands, st.cache_data or st.cache_resource.
More information in our docs.
NYC Ridesharing Data
Examining how Uber pickups vary over time in New York City's and at its major regional airports. By sliding the slider on the left you can view different slices of time and explore different transportation trends.
Select hour of pickup
0
0
23
All New York City from 0:00 and 1:00
La Guardia Airport from 0:00 and 1:00
JFK Airport from 0:00 and 1:00
Newark Airport from 0:00 and 1:00
Jersey City
New York
LGA
JFK
EWR
Newark
East Orange
Harrison
Breakdown by minute between 0:00 and 1:00
pickups

ESTIA INSTITUTE OF TECHNOLOGY

Demo #2
Model serving (FastAPI) + Frontend (StreamLit)

https://github.com/eishkina-estia/fastapi_example
https://github.com/eishkina-estia/streamlit_example
https://github.com/streamlit/demo-uber-nyc-pickups/ (the app deployed on Streamlit Cloud)

HANDS-ON PROJECT
Step #2: Model Serving (backend and frontend)

Exercises

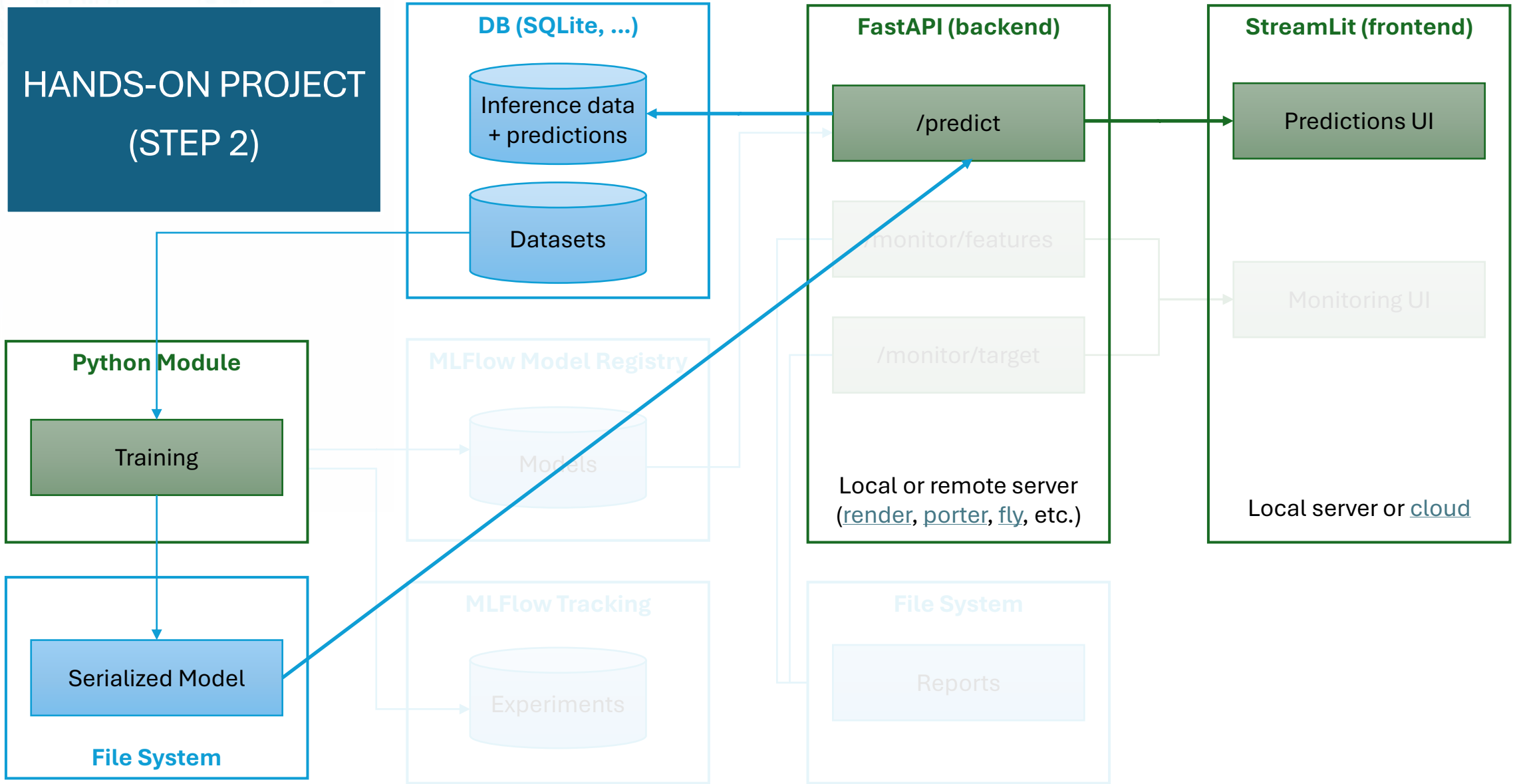
- **Clone** https://github.com/eishkina-estia/fastapi_example and open it in your IDE
- **Activate the venv** that you created for your project
- **Install fastapi** package in your venv
- **Launch the application**: python main.py
- **Test the API using SwaggerUI**: localhost:8000/docs
- **Clone** streamlit_example and test it (see readme)

Design, implement, and test API to serve the model for NYC taxi trip duration prediction (hands-on #1).

- **Add a subfolder** in your hands-on project folder for API
- **Define the schema (model) for your input** – a class inherited from pydantic BaseModel)
- **Implement the endpoint for making predictions** using your model (stored in models folder)
- **Test the endpoint** using SwaggerUI
- **Persist predictions** in an SQLite database (optional)

Implement frontend for predictions using StreamLit

- **Add a subfolder** in your hands-on project for UI
- **Implement the UI** (app.py) using the demo as template

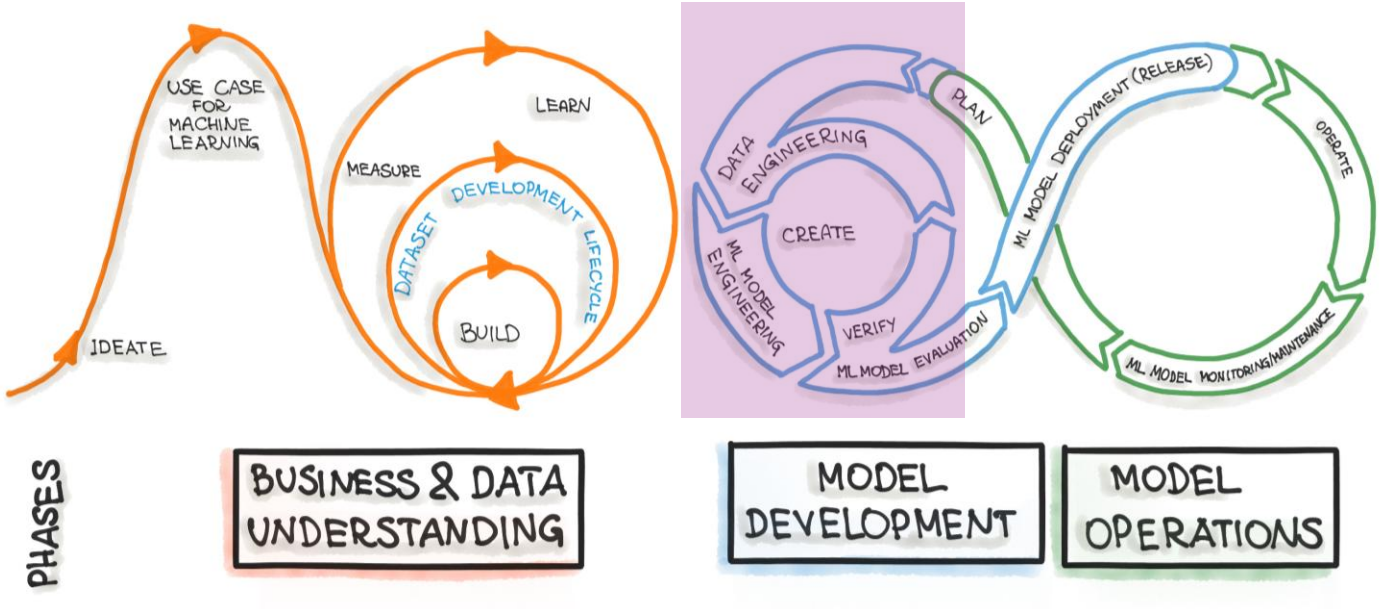

HANDS-ON PROJECT
(STEP 2)
DB (SQLite, ...)
Inference data
+ predictions
Datasets
FastAPI (backend)
/predict
/monitor/features
/monitor/target
Local or remote server
(render, porter, fly, etc.)
StreamLit (frontend)
Predictions UI
Monitoring UI
Local server or cloud
Python Module
Training
Serialized Model
File System
MLFlow Model Registry
Models
MLFlow Tracking
Experiments
File System
Reports

ESTIA
INSTITUTE OF TECHNOLOGY

Experiments tracking and model registry

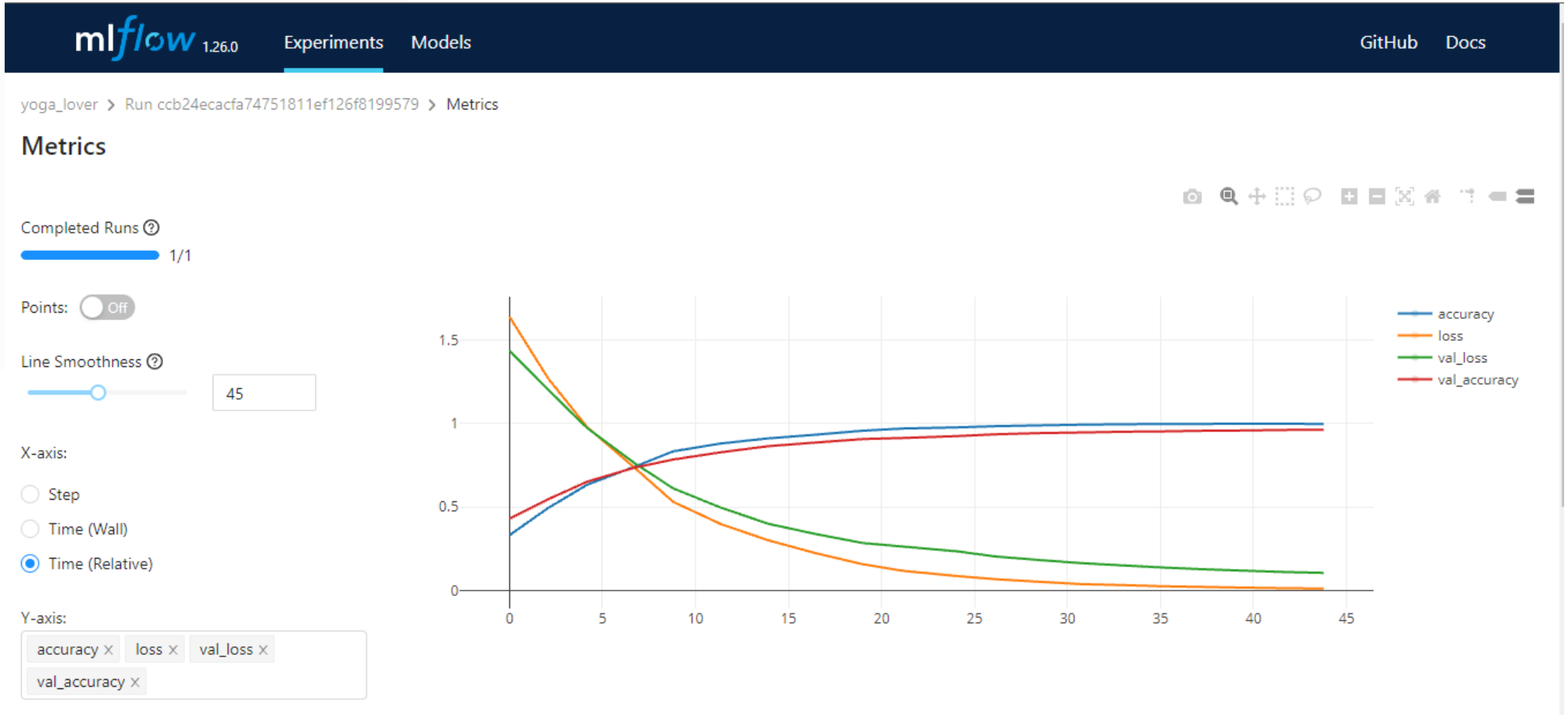
CRISP-ML(Q): Experiments tracking

- Data Engineering: data ingestion and cleaning, feature engineering
- Model Engineering: model training, hyperparameters tuning, model serialization, documenting and tracking experiments
- Model Evaluation: model performance validation, deciding whether to deploy the model, documenting the evaluation phase



MLflow

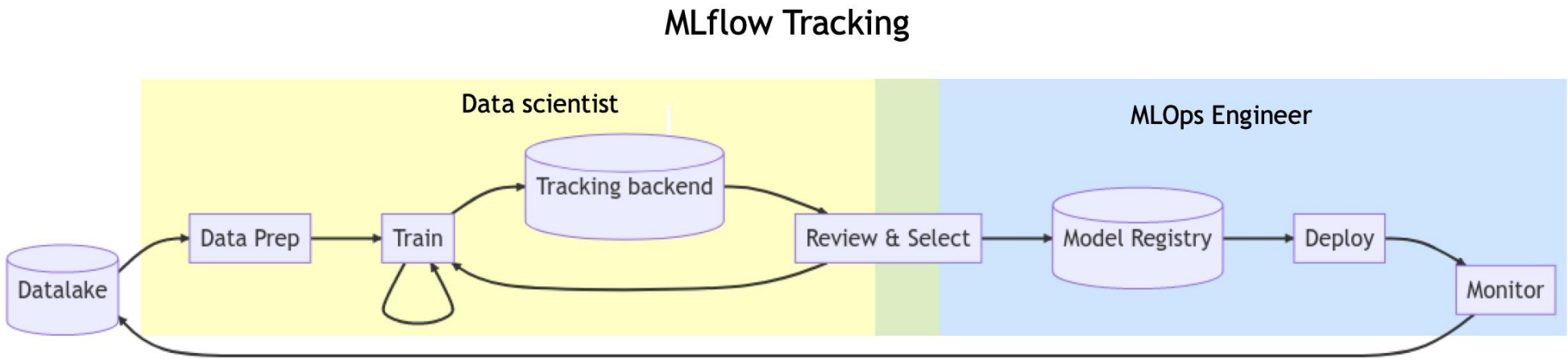
MLflow is an open-source platform for the ML life cycle, with a focus on reproducibility, training, scalability, and deployment.



Use Cases of MLflow

- Experiment Tracking: A data science team leverages MLflow Tracking to log parameters and metrics for experiments within a particular domain. Using the MLflow UI, they can compare results and fine-tune their solution approach. The outcomes of these experiments are preserved as MLflow models.
- Model Selection and Deployment: MLOps engineers employ the MLflow UI to assess and pick the top-performing models. The chosen model is registered in the MLflow Registry, allowing for monitoring its real-world performance.
- Model Performance Monitoring: Post deployment, MLOps engineers utilize the MLflow Registry to gauge the model's efficacy, juxtaposing it against other models in a live environment.
- Collaborative Projects: Data scientists embarking on new ventures organize their work as an MLflow Project. This structure facilitates easy sharing and parameter modifications, promoting collaboration.

Mlflow Tracking



As a data scientist, your explorations involve running your evolving training code many times. MLflow Tracking allows you to record important information about your run, review and compare it with other runs, and share results with others.

As an ML / MLOps Engineer, it allows you to compare, share, and deploy the best models produced by the team.

https://mlflow.org/docs/latest/getting-started/quickstart-1/index.html

Core Components of MLflow

- Tracking: Record and query experiments: code, data, config, and results
- Models: Deploy machine learning models in diverse serving environments.
- Model Registry: store, annotate, discover, and manage models in a central repository.
- Projects: package data science code in a format to reproduce runs on any platform

MLflow Tracking: Concepts

- **Run** is the main concept. It is an execution of some piece of data science code.
- **Code version**: git commit hash used for the run, if it was run from an MLflow Project.
- **Start & End Time**: start and end time of the run
- **Source**: name of the file to launch the run, or the project name and entry point for the run if run from an MLflow Project.
- **Parameters**: key-value input parameters of your choice. Both keys and values are strings. For example, you can log hyperparameter values.
- **Metrics**: key-value metrics, where the value is numeric. Each metric can be updated throughout the course of the run (for example, to track how your model's loss function is converging), and MLflow records and lets you visualize the metric's full history.
- **Artifacts**: output files in any format. For example, you can record images (for example, PNGs), models (for example, a pickled scikit-learn model), and data files (for example, Parquet files) as artifacts.

MLflow Tracking: Storing data

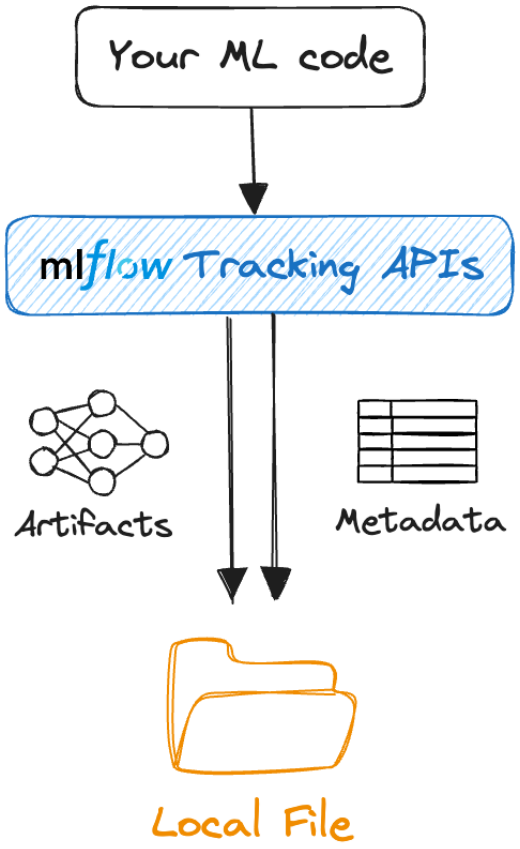
- **Backend Store**

  Persists various metadata for each run, such as run ID, start and end times, parameters, metrics, etc. MLflow supports two types of storage for the backend: file-system-based like local files and database-based like PostgreSQL.

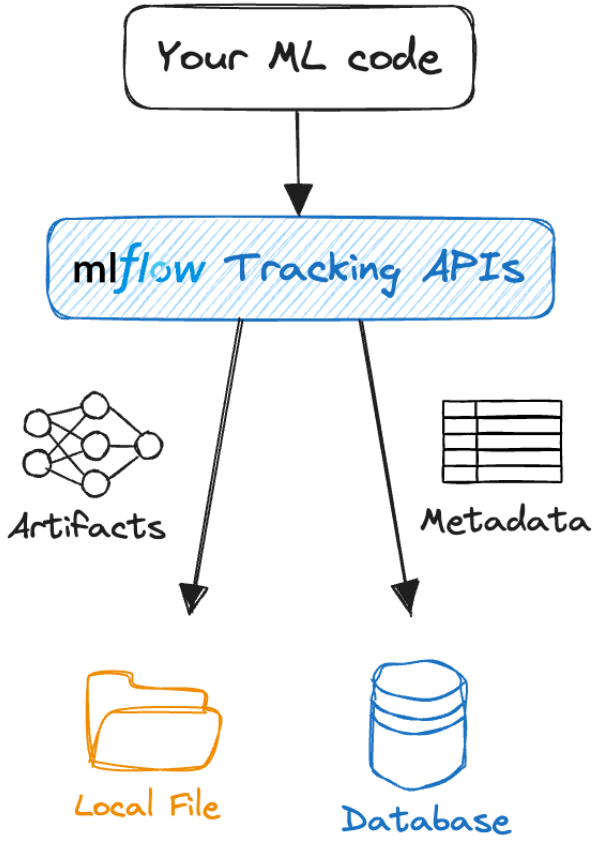
- **Artifact Store**

  Persists (typicaly large) artifacts for each run, such as model weights (e.g. a pickled scikit-learn model), images, model and data files. MLflow stores artifacts in local files (*mlruns*) by default, but also supports different storage options such as Amazon S3 and Azure Blob Storage.

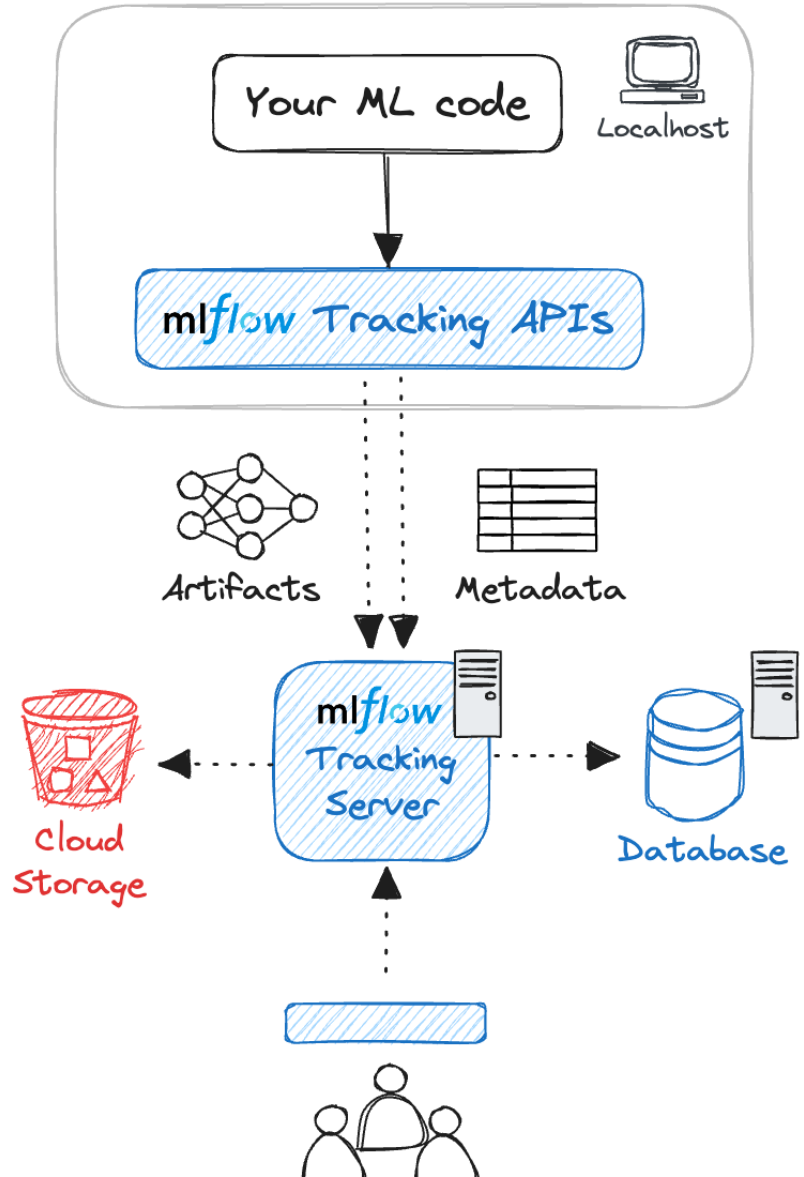
1. Localhost (default)



2. Localhost w/ various data stores



3. Remote Tracking w/ Tracking Server



MLflow Projects

- Format for packaging data science code in a reusable and reproducible way, based primarily on conventions.
- Name: a human-readable name for the project.
- Entry Points: commands that can be run within the project (for example, .py or .sh files).
- Environment: the software environment that should be used to execute project entry points (library dependencies). Supported environments: virtualenv (preferred), docker, conda.

File: MLproject (yaml format)

```yaml
name: My Project

python_env: python_env.yaml


entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization} {data_file}“
  validate:
    parameters:
      data_file: path
    command: "python validate.py {data_file}"
```

File: python_env.yaml

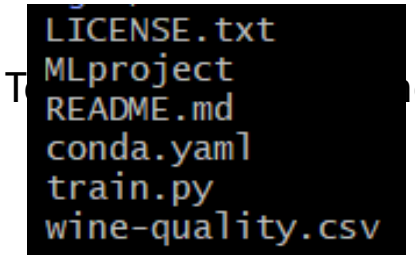
```yaml
# Python version required to run the project.
python: "3.8.15“
# Dependencies required to build packages. This field is optional.
build_dependencies:
  - pip
  - setuptools
  - wheel==0.37.1
# Dependencies required to run the project.
dependencies:
  - mlflow==2.3
  - scikit-learn==1.0.2
```

MLflow Projects

- Format for packaging data science code in a reusable and reproducible way, based primarily on conventions.
- Name: a human-readable name for the project.
- Entry Points: commands that can be run within the project (for example, .py or .sh files).
- Environment: the software environment that should be used to execute project entry points (library dependencies). Supported environments: virtualenv (preferred), docker, conda.

- Example of project https://github.com/mlflow/mlflow-example

- T ore details here):




```
mlflow run
git@github.com:mlflow/mlflow-
example.git -P alpha=0.5
```

MLflow Models

- MLflow Model is a directory containing arbitrary files, together with an MLmodel file in the root of the directory that can define multiple flavors (conventions that deployment tools) that the model can be viewed in.
- MLflow defines multiple built-in flavors:
  - python_function (default): running the model as a Python function (mlflow.sklearn in the example: loading models as a scikit-learn Pipeline object)
  - sklearn: using either Python's pickle module or CloudPickle for model serialization.
  - Built-In Model Flavors

```
# Directory written by mlflow.sklearn.save_model(model, "my_model")
my_model/
├── MLmodel
├── model.pkl
├── conda.yaml
├── python_env.yaml
└── requirements.txt
```

File: MLmodel (yaml format)

```
time_created: 2018-05-25T17:28:53.35

flavors:
  sklearn:
    sklearn_version: 0.19.1
    pickled_model: model.pkl
  python_function:
    loader_module: mlflow.sklearn
```

MLflow Models

- Apart from flavors, the MLmodel YAML format can contain the following fields:
  - **time_created**: when the model was created.
  - **run_id**: ID of the run that created the model, if the model was saved using MLflow Tracking.
  - **signature**: model signature in JSON format.
  - **input_example**: reference to an artifact with input example.
  - **mlflow_version**: the version of MLflow that was used to log the model.

```
signature:
  inputs: '[
    {"name": "sepal length (cm)", "type": "double"},
    {"name": "sepal width (cm)", "type": "double"},
    {"name": "petal length (cm)", "type": "double"},
    {"name": "petal width (cm)", "type": "double"},
    {"name": "class", "type": "string", "optional": "true"}]'
  outputs: '[{"type": "integer"}]'
  params: null
```

```
signature:
  inputs: '[{"name": "text", "type": "string"}]'
  outputs: '[{"name": "output", "type": "string"}]'
  params: '[
    {"name": "temperature", "type": "float",
     "default": 0.5, "shape": null},
    {"name": "top_k", "type": "integer",
     "default": 1, "shape": null},
    {"name": "suppress_tokens", "type": "integer",
     "default": [101, 102], "shape": [-1]}]'
```

MLflow Tracking UI

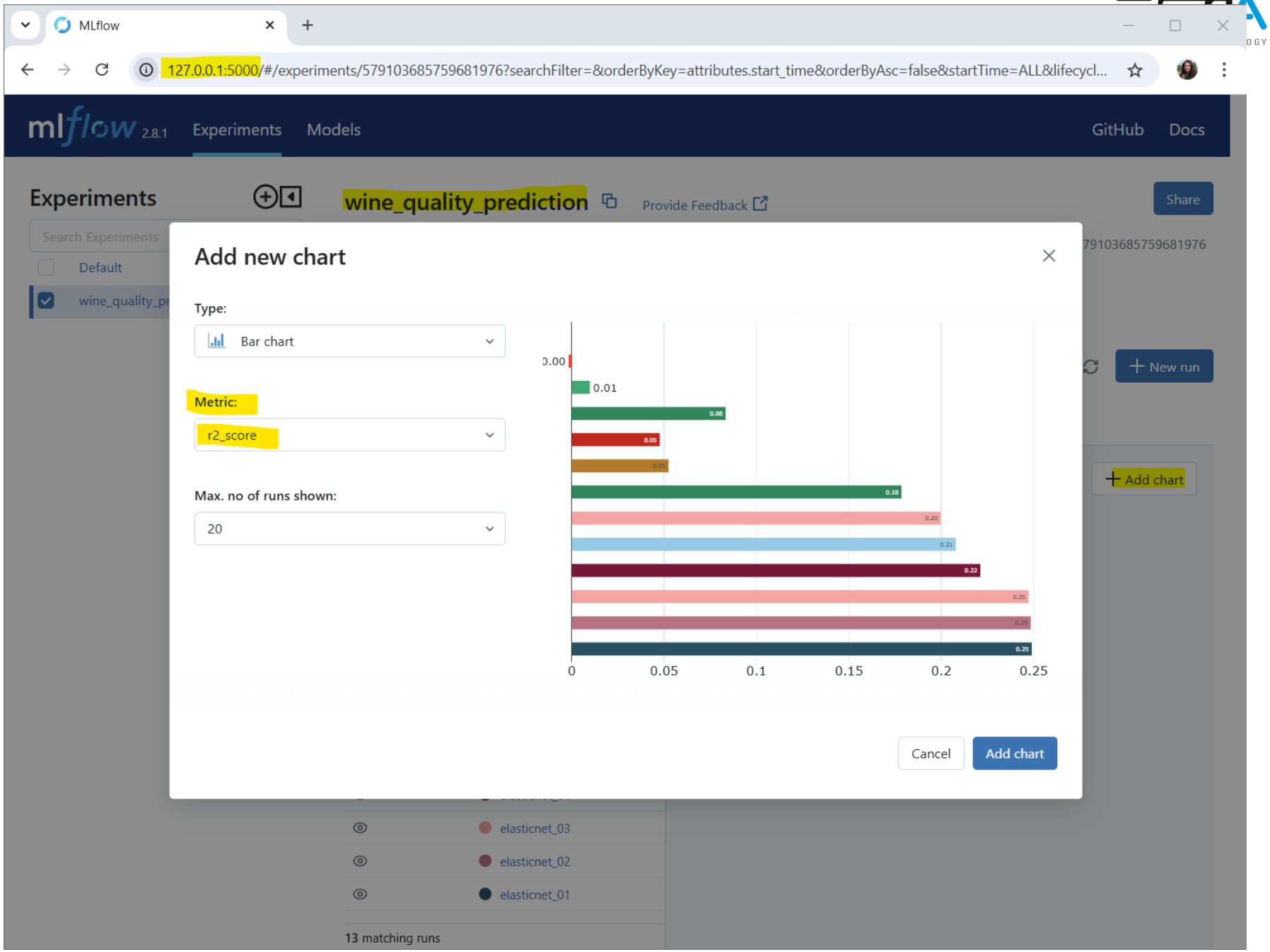
- Experiment-based run listing and comparison (including run comparison across multiple experiments)
- Searching for runs by parameter or metric value
- Visualizing run metrics
- Downloading run results (artifacts and metadata)

- If you log runs to a local mlruns directory, run the following command in the directory above it, then access http://127.0.0.1:5000 in your browser.

```
mlflow ui --port 5000
```

MLflow Tracking UI

Experiment-based run listing and comparison (including run comparison across multiple experiments)



ESTIA
INSTITUTE OF TECHNOLOGY

Demo #4
MLFlow

https://github.com/eishkina-estia/mlflow_example

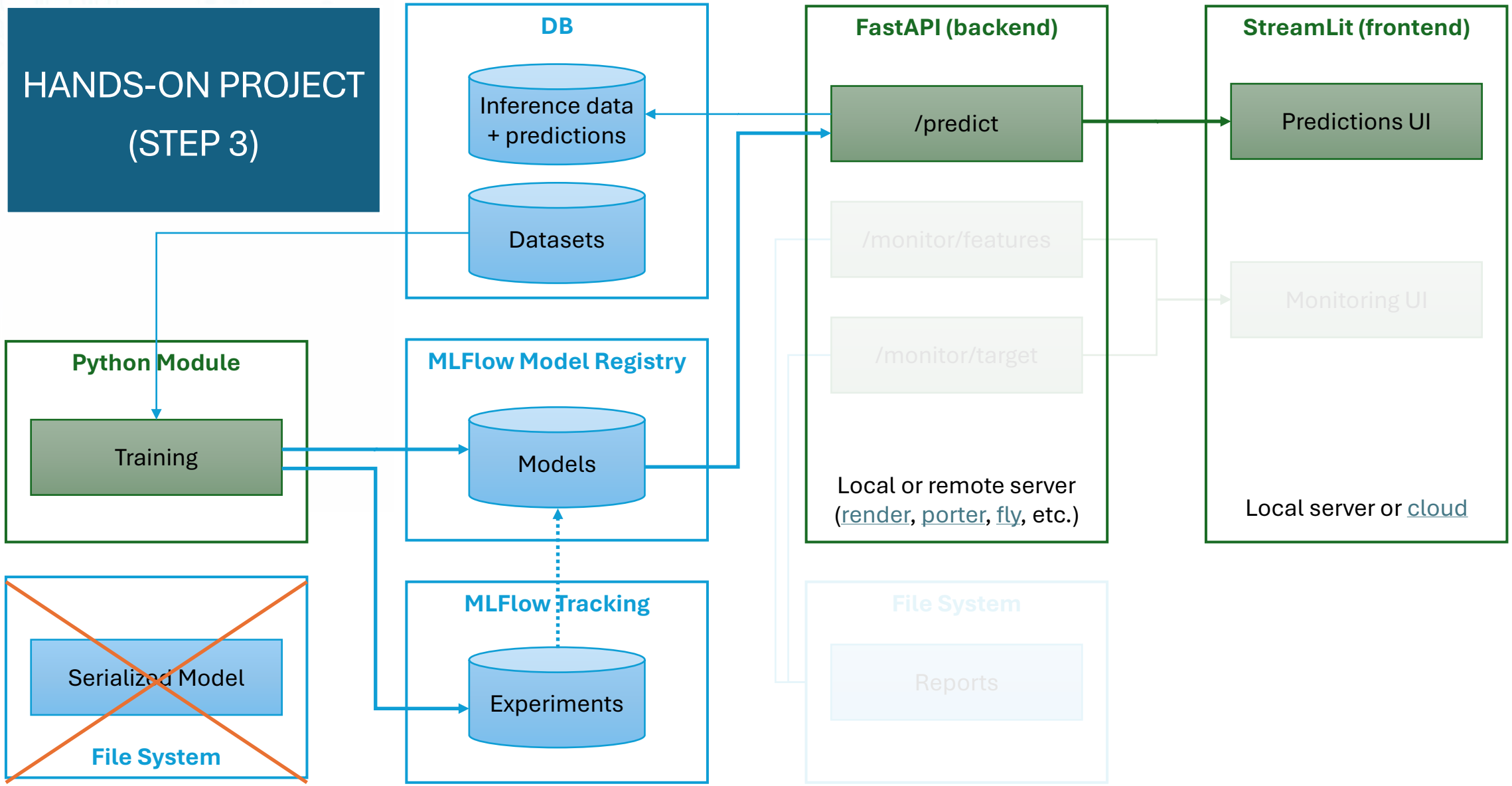
Exercise

- Clone the example https://github.com/eishkina-estia/mlflow_example
- Install mlflow package in your virtual environment created for the hands-on lab and run train_elasticnet.py using this environment
- Using the Tracking UI (mlflow ui), examine the run storage and the model registry in detail. Add a chart to compare experiments using a chosen metric.
- Create a new python script and implement another mlflow run with a regression algorithm of your choice.
- If the performance of this model is better than that of the ElasticNet model, save the model to the registry. Perform the comparison using mlflow.

HANDS-ON PROJECT
Step #3: Tracking Experiments & Model Registry

- Add MLFlow tracking to your project, according to the schema on the next slide, and save a few runs
- Use Mlflow model registry, according to the schema on the next slide, to save the best model and to load this best model for your API
- Find a solution to persist preprocessing logic with your model in mlflow

HANDS-ON PROJECT (STEP 3)



Monitoring

Data and concept drift

- Data drift: the distribution of input data changes, even if the relationship between input and output remains the same.
  - Covariate Shift: the distribution of input features changes, but the relationship between features and labels remains unchanged.
    Example: Self-Driving Cars in Different Weather Conditions
    A self-driving car model is trained using images collected in sunny weather. When deployed in snowy conditions, the visual features of the road (color, contrast, visibility) are significantly different from the training data, but the relationship between road features and driving decisions remains the same. The model might struggle because it hasn't seen these variations before.
  - Label Shift: the distribution of the target variable changes while the feature distribution remains stable.
    Example: Disease Diagnosis in Different Populations
    A medical diagnostic model is trained on a dataset where 10% of patients have a rare disease. When deployed in a new region with a higher prevalence of the disease (e.g., 30% of patients are affected), the prior probability of the disease has changed. The model may under-predict or over-predict the condition because it was calibrated on a different label distribution.
- Concept drift: the relationship between input features and the target variable changes over time.
  Example: Email Spam Detection
  Imagine a ML model trained to detect spam emails. Initially, spam emails often contain words like "free", "win", and "prize". However, over time, spammers adapt and start using different words (e.g., "limited offer", "exclusive deal"). The underlying concept of what defines spam has changed, causing the model to become less effective if not updated.

Additional reading

- https://content.dataiku.com/oreilly-mlops/data-drift-technical
- Monitoring and explainability of models in production
- Protecting Your Machine Learning Against Drift: An Introduction

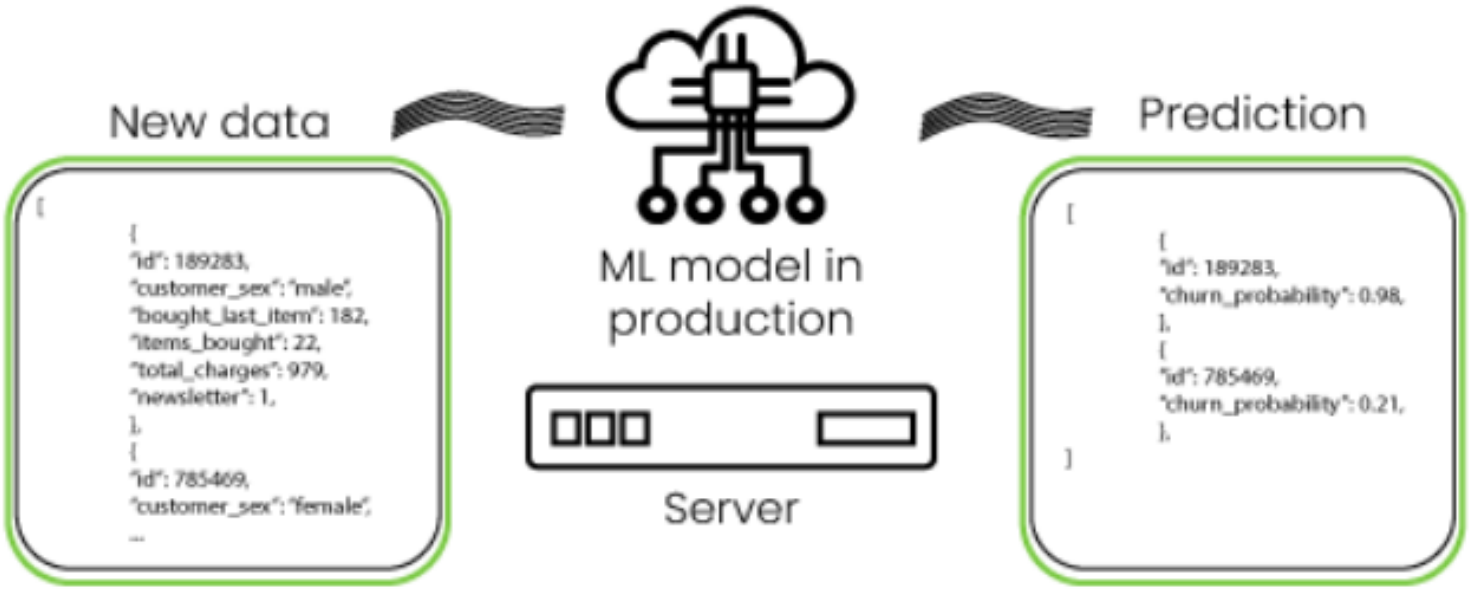
Detecting Data Distribution Shifts

- Monitor model's performance metrics – accuracy, F1 score, recall, AUC-ROC, etc. – in production to see whether they have changed (decreased). NB During model development, you have access to labels, but in production, you don't always have access to labels, and even if you do, labels will be delayed.
- Statistical methods: Kolmogorov–Smirnov test, Least-Squares Density Difference, etc.
- Alibi Detect: is an open source Python library focused on outlier, adversarial and drift detection. The package aims to cover both online and offline detectors for tabular data, text, images and time series.

Application	Attribution metric	Retraining periodicity (approximate)
Churn prediction	Purchase event after action taken on customer with high probability	Monthly
Customer lifetime value (CLV)	% of continued CLV group membership	Weekly
	Stability	
Transportation Industry	Revenue	Monthly
Demand/pricing	Purchase rate	
Recommendation engine (personalization)	Purchase rate or viewership rate	Hourly or daily
Image content labeling	% error in classification	Two to six months
Fraud detection	Loss event count	Biweekly
	Loss amounts	
	Undetected fraud event count	
Equipment failure	Maintenance costs (replacement)	Semiannually or annually
Prediction (survivability)	Count of unrequired maintenance	
Sales forecasting	Backtesting accuracy in projection	Daily or weekly

Source: Machine Learning Engineering in Action, by Ben Wilson, Manning, 2022

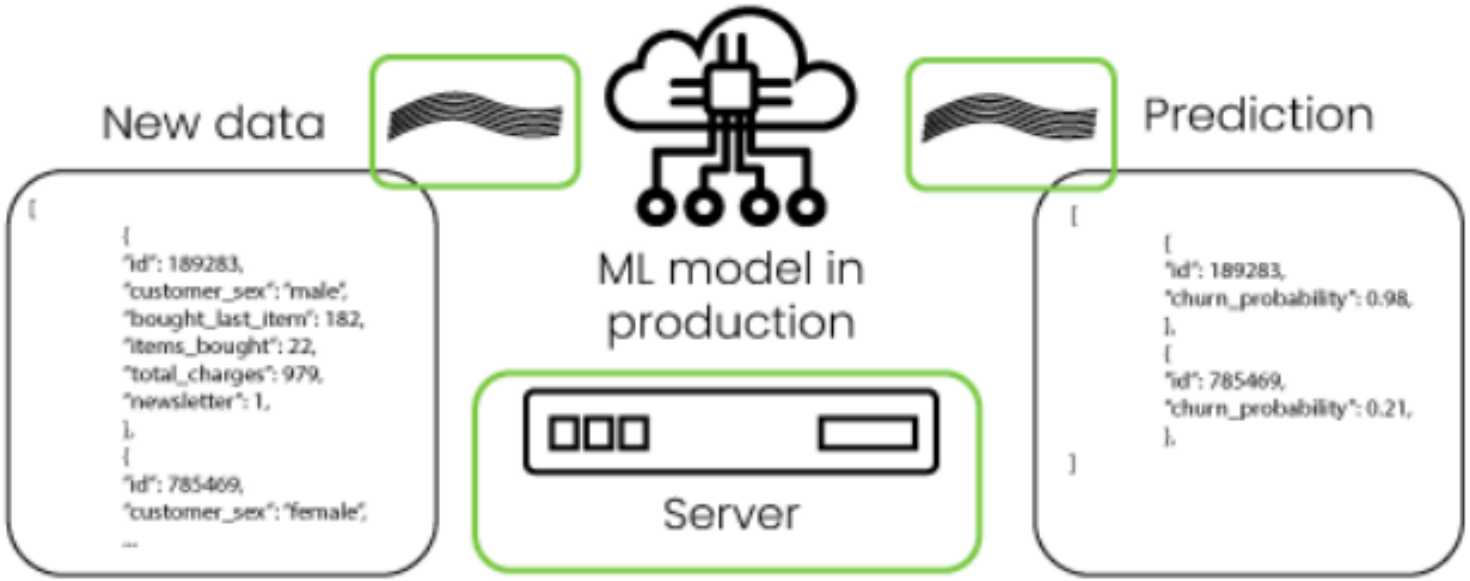
Monitoring



Statistical monitoring: focuses on the input and output data.

Examples: customer X has churn probability of 98%, customer Y has churn probability of 21%.

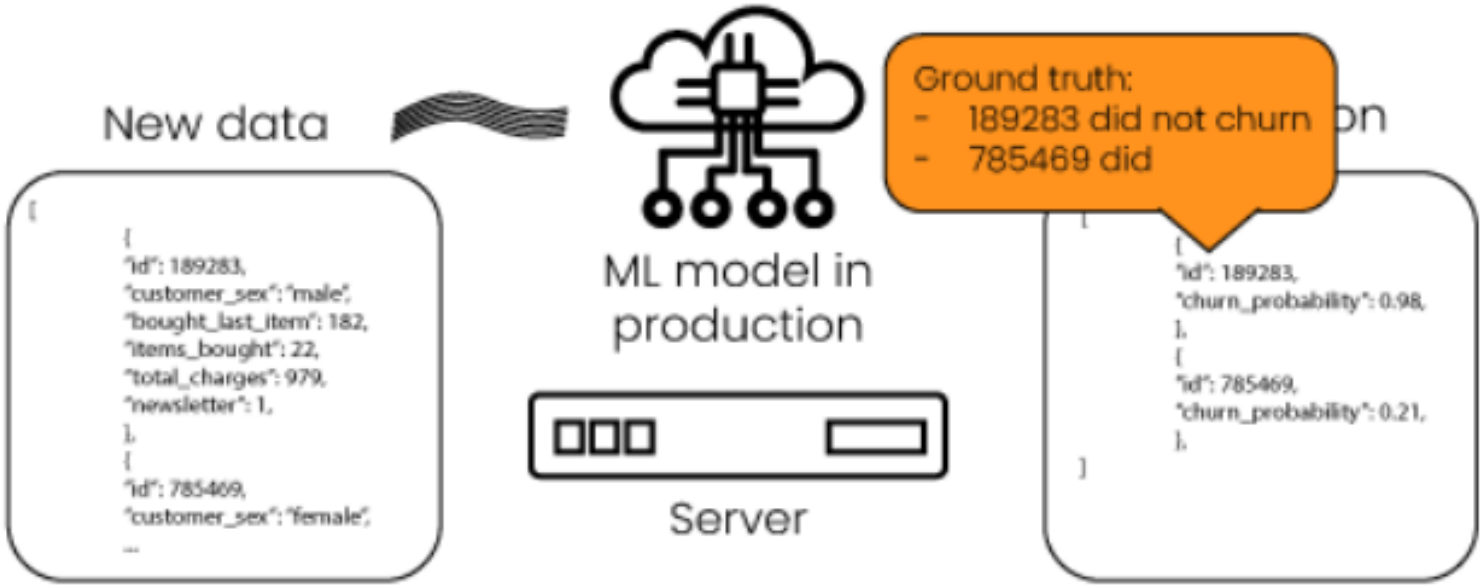
Monitoring



Computational monitoring: focuses on technical metrics.

Examples: server CPU usage, number of incoming requests, number of predictions, downtime of server.

Monitoring



Feedback loop: the process through which the ground truth is used to improve the machine learning model.

HANDS-ON PROJECT (STEP 4)