Lecture 1 - Introduction, MLOps

Advanced Machine Learning

Miroslav Čepek, Zdeněk Buk, Rodrigo da Silva Alves, Alexander Kovalenko, Vojtěch Rybář, Petr Šimánek

FIT CTU

20. 2. 2025

Contacts

Miroslav Čepek	miroslav.cepek@fit.cvut.cz
Zdeněk Buk	zdenek.buk@fit.cvut.cz
Rodrigo da Silva Alves	rodrigo.alves@fit.cvut.cz
Alexander Kovalenko	alexander.kovalenko@fit.cvut.cz
Vojtěch Rybář	vojtech.rybar@fit.cvut.cz
Petr Šimánek	petr.simanek@fit.cvut.cz

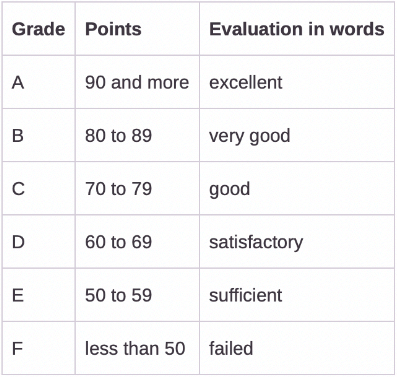
https://courses.fit.cvut.cz/NI-AML

Agenda

- What to expect
- Assessment rules, exam, organisation
- Repetability of Machine Learning

Grades

- Final grades are based on grand total of your points from semester and exam.
- Using standard conversion table:



Grade	Points	Evaluation in words
A	90 and more	excellent
B	80 to 89	very good
C	70 to 79	good
D	60 to 69	satisfactory
E	50 to 59	sufficient
F	less than 50	failed

Points

- 4 $\times$ Homeworks, each 5 points - (up to 20 points)
- Semestral Projects (up to 50 points)
- Oral Exam (up to 30 points)

- From homeworks and project you have to get at least 50 point.
- To pass the exam, you need to get at least 10 points.

Homeworks

- Finish experiments and explore implementation on given topic.
- Submission is piece of code solving the problem and possibly a short report.
- Target is spend two hours per each homework.

Semestral Project

- Topics will be available on course webpage: https://courses.fit.cvut.cz/NI-AML/semestral_projects.html (presently shows an incomplete version)
- Outcomes:
  - Final presentation
  - Report
  - Implementation

Exam

- Each of the lecturers will open ∼6 slots.
- You will pick one lecturer and register with him.
- Expect 15-20 minute chat **mostly** about topics given by particular lecturer. It may also include questions about your project.

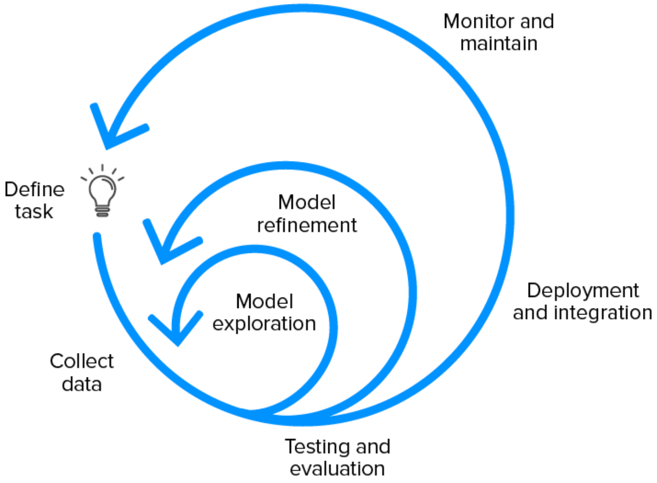
What I assume you know?

- We assume:
  - knowledge of Python.
  - familiarity with ML libraries (Torch/Tensorflow).
  - understanding of machine learning/AI principles and basic techniques.

Why repeatable models

- Training process review.
- Models need to be investigated when failed.
- Recreate the model.
- Model selection and parameter tuning.
- (Automated) refresh of the model on top of new data.

Lifecycle Of A Machine Learning Application



ML App Project Estimate

Reproducibility of ML Model

- The problem with ML project is that the result depends on too many and too diverse factors, including:
  - Dataset (and selection of training data)
  - Data preprocessing
  - Model type, architecture, optimisation procedure and hyper-parameters
  - Random seed
- In order to get the some model twice, you need to get all above right.

Reproducibility of ML Model (2)

- **Reproducibility through data versioning and experiment tracking** — Keeping track of data version, preprocessing, model architecture and training parameters of past experiments.
- **The incorporation of feedback loops** — Keeping track of changes done to a model architecture, training process, parameters, loss functions, . . . and the impact on model performance.
- **The continuity of the model training and monitoring process** — In production applications, data evolves over time. As it changes over time, the model must be retrained to include this new information and adapt accordingly to avoid issues such as data and concept drift.

Reproducibility Challenges (1)

Phase	Collecting Data
Challenges	Generation of the training data can't be reproduced (e.g due to constant database changes or data loading is random)
Ensure Reproducibility	1. Always backup your data. 2. Saving a snapshot of the data set (e.g. on the cloud storage). 3. Data sources should be designed with timestamps so that a view of the data at any point can be retrieved. 4. Data versioning.

Taken from ML Ops Principles from ml-ops.org

Reproducibility Challenges (2)

Phase	Feature Engineering
Challenges	Scenarios: 1. Statistic based feature engineering: Missing values imputation, normalisations parameters; Categorical $\rightarrow$ Numerical and vice versa conversions. 2. Non-deterministic feature extraction methods.
Ensure Reproducibility	1. Feature generation code should be taken under version control. 2. Require reproducibility of the previous step "Collecting Data"

Taken from ML Ops Principles from ml-ops.org

Reproducibility Challenges (3)

Phase	Model Training / Model Build
Challenges	Non-determinism
Ensure Reproducibility	1. Ensure the order of features is always the same. 2. Document and automate feature transformation, such as normalization. 3. Document and automate hyperparameter selection. 4. For ensemble learning: document and automate the combination of ML models.

Taken from ML Ops Principles from ml-ops.org

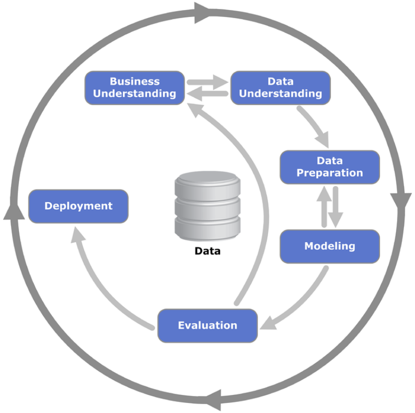
Reproducibility Challenges (4)

Phase	Model Deployment
Challenges	1. Training the ML model has been performed with a software version that is different to the production environment. 2. The input data, which is required by the ML model is missing in the production environment.
Ensure Reproducibility	1. Software versions and dependencies should match the production environment. 2. Use a container (Docker) and document its specification, such as image version. 3. Ideally, the same programming language is used for training and deployment.

Taken from ML Ops Principles from ml-ops.org

CRISP-DM

- Attempt to define standardised steps of machine learning process.
    1. Business Understanding
    2. Data Understanding
    3. Data Preparation
    4. Modeling
    5. Evaluation
    6. Deployment
- Set of actions and checkboxes to be marked in each stage before you move further (or back).
- Some commercial tools supports the flow.



Wikipedia Image
CRISP-DM The New Blueprint for Data Mining

Machine Learning Operations

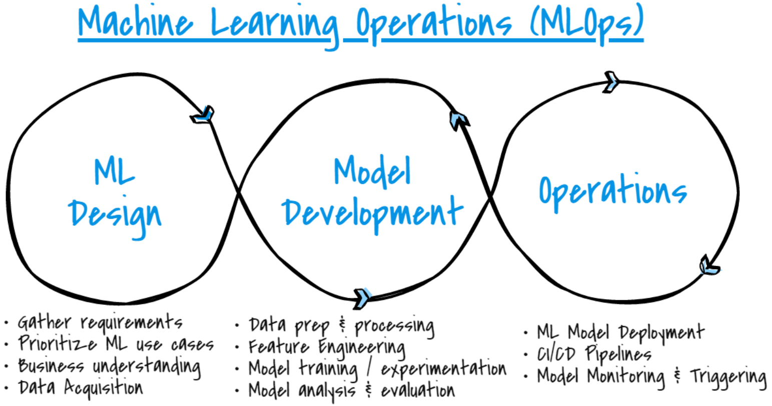
- Evolution of the reproducible ML idea.
- Envelop the whole process into series of steps and support it with software tools.

Machine Learning Operations

- MLOps is a paradigm - including best practices - as well as a development culture when it comes to the end-to-end conceptualisation, implementation, monitoring, deployment, and scalability of ML products.
- It is a SW engineering practice that leverages three contributing disciplines: ML, software engineering and data engineering.
- MLOps aims to facilitate the creation of machine learning products by leveraging these principles: CI/CD automation, workflow orchestration, reproducibility; versioning of data, model, and code; collaboration; continuous ML training and evaluation; ML metadata tracking and logging; continuous monitoring; and feedback loops.

Machine Learning Operations (MLOps): Overview, Definition, and Architecture

Machine Learning Operations



Pratik Sherma - 10 Best MLOps Tools in 2022.

MLOps vs DevOps

- DevOps – describes processes and interactions between developers and operations people. Ways to hand over code, automate tests, validations and deployment into production. Monitoring service availability and quality.
- MLOps – describes processes between ML/DS/AI practitioners and rest of the organisation. Improve record keeping, simplify creation, deployment and monitoring of the machine learning models.
- In both cases it focuses on automation and reproducibility.

MLOps Areas

- Experiment Tracking
- Model Registry
- Data Versioning
- Feature Store
- ML Model Deployment and Monitoring
- Pipeline (Project) Management

Experiment Tracking

- is a process and tool(s) to save all important experiment related information. So you can later return, review and compare different experiments.
- Values to store includes for example:
  - Code of the experiment,
  - System (environment) configuration, ie. libraries and their versions,
  - Versions of the data, training/evaluation split,
  - Hype-parameters
  - Evaluation metrics
  - Performance visualisations (confusion matrix, ROC curve, learning curves, example predictions, etc.)

Inspired by Neptune AI Blogpost

Model Registry

- A place to store all the trained models (their binary serialisation).
- Models are typically identified by ID.
- MR works closely with Experiment Tracking.
- Records model life-cycle stage (experimental, staging/testing, in production, retired).

Feature Store/Data Versioning

- Feature store is a data management system that manages and serves features to machine learning models.
- Provide single and unified way to calculate individual features.
- Goto place for training and deplyment data.
- Part/Layer above data lakes/databases.
- Data versioning – different snapshots of the data (ie different times).

Feature Store 101

ML Model Deployment and Monitoring

- System/Process to put ML model into production (make it available to customers).
- Some MLOps systems (ie MLFlow) contains a deployment service for this purpose.
- ML Model Deployment Strategies

- Monitoring is needed due to data drift (natural changes in world as captured in the data).
- Automatically evaluate performance metrics and alerting.

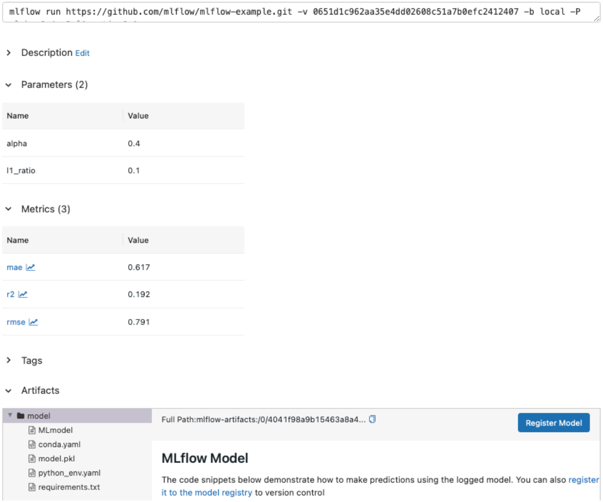
Weights and Biases

- Cloud based service available at `wandb.ai`.
- Track experiments, results (artefacts), automates selected tasks
  - Hyperparameter tuning
  - Reporting
  - Alerting
- Can be deployed locally.
- Free tier for personal use (with much more generous free offer for university students).
- Notebook with Example Experiment
- Publicly accessible WandB Tracking UI

MLFlow

- https://www.mlflow.org
- Experiment Tracking — is an API and UI for logging parameters, code versions, metrics, and artifacts when running your machine learning code and for later visualizing the results.
- Projects – to unify structure of different experiments into single shape (described by YAML file), which can be understood and executed by MLFlow.
- MLflow Models offer a convention for packaging machine learning models in multiple flavours (TF, Torch, GLUON, SK-L, ...), and a variety of tools to help you deploy them.
- MLflow Registry offers a centralized model store to manage the full lifecycle of an MLflow Model. It provides model lineage, model versioning, stage transitions, and annotations.

MLFlow



```
mlflow run https://github.com/mlflow/mlflow-example.git -v 0651d1c962aa35e4dd02608c51a7b0efc2412407 -b local -P
```

› Description Edit

⌄ Parameters (2)

Name	Value
alpha	0.4
l1_ratio	0.1

⌄ Metrics (3)

Name	Value
mae	0.617
r2	0.192
rmse	0.791

› Tags

⌄ Artifacts

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control

Source Code

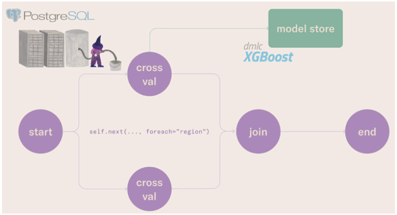
Neptune.ai

- Cloud based service available at `http://neptune.ai`.
- Experiment tracker and model registry.
- Allows you to record code, metrics and media (images, videos) and artefacts (serialised models).
- Free tier for personal use (with much more generous free offer for university students).

- Notebook with Example Experiment
- Publicly Accessible Neptune Tracking UI

ML Orchestrator - Metaflow

- Allows you to build a repeatable pipeline.
- Pipeline is a series of steps, that can be automatically executed by scheduler in serial/parallel manner (and can be distributed in cloud/k8s environment)



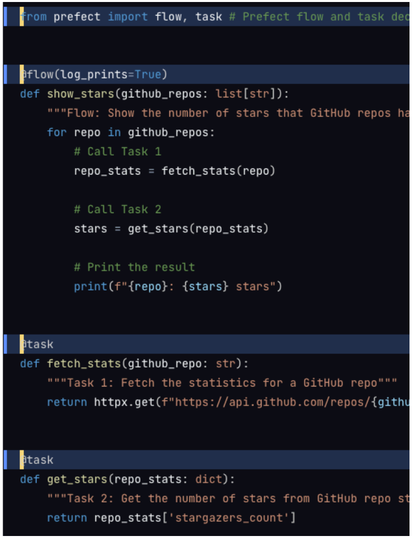
```
@step
def start(self):
    """
    This is the 'start' step. All flows must hav
    is the first step in the flow.

    """
    print("HelloFlow is starting.")
    self.next(self.hello)

@step
def hello(self):
    """
    A step for metaflow to introduce itself.

    """
    print("Metaflow says: Hi!")
    self.next(self.end)

@step
def end(self):
    """
    This is the 'end' step. All flows must have
    last step in the flow.

    """
    print("HelloFlow is all done.")
```

Metaflow Tutorial

ML Orchestrator - Perfect

- Pythonic – workflows in native.
- State & Recovery – state management tracking success, failure, and retries.
- Flexible & Portable Execution
- Event-Driven – trigger flows on schedules, external events, or via API.
- Dynamic Runtime
- Modern UI – flow run monitoring, logging, and state tracking.
- CI/CD First

www.perfect.io
kedro.org



```
from prefect import flow, task # Prefect flow and task dec


@flow(log_prints=True)
def show_stars(github_repos: list[str]):
    """Flow: Show the number of stars that GitHub repos ha
    for repo in github_repos:
        # Call Task 1
        repo_stats = fetch_stats(repo)

        # Call Task 2
        stars = get_stars(repo_stats)

        # Print the result
        print(f"{repo}: {stars} stars")


@task
def fetch_stats(github_repo: str):
    """Task 1: Fetch the statistics for a GitHub repo"""
    return httpx.get(f"https://api.github.com/repos/{githu


@task
def get_stars(repo_stats: dict):
    """Task 2: Get the number of stars from GitHub repo st
    return repo_stats['stargazers_count']
```

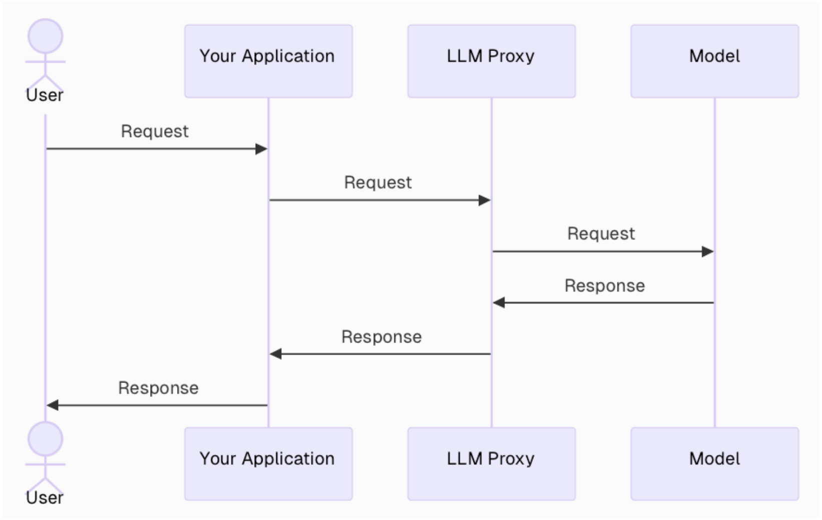
LLM Proxy

is a service siting between application and the LLM provider's API. LLM proxies is that they are very easy to incorporate. It usually requires changing the *Base URL* of the LLM provider.

- Caching: Store and reuse previous responses to reduce API calls, costs and latency.
- Cost management: Cost tracking, spent limits, select cheaper, good-enough model/provider.
- Unified API: easier provider switching.
- Security: Hides API keys/sec2rets from applications, limit who can access LLM and specific application. Pre-checking question and post-processing answers.
- Rate limiting: avoid exceeding provider's limits..
- Request routing: applications can use various model versions.

Great system is Lite LLM (Lite LLM Doc Website) or vLLM
langfuse.com blog

LLM Proxy Illustration



langfuse.com blog

Embedding Databases (LLM)

- Specialised feature store - focusing on storing vector representations and nearest neighbour search.
- Essential in NLP - semantic similarity search of documents – retrieval part of RAG systems.
- Image similarity, person identification in photos, ...

- Repurposed (NoSQL) Database, like ElastiSearch or REDIS.
- Specialised database:
  - ChromaDB - apart from embeddings, stores document's metadata to be used later by LLMs and can also be used for semantic search engines over text data.
  - Weaviate - It is designed to be scalable and easy to use.
  - Milvua

Feature Stores / Data Versioning

- Pachyderm – pachyderm.com
  - *Pachyderm is cost-effective at scale, enabling data engineering teams to automate complex pipelines with sophisticated data transformations.*
- Data Version Control – dvc.org
  - *DVC is a tool for data science that takes advantage of existing software engineering toolset. It helps machine learning teams manage large datasets, make projects reproducible, and collaborate better.*
- Feast – feast.dev
  - *Feast is a standalone, open-source feature store that organisations use to store and serve features consistently for offline training and online inference.*

https://lakefs.io/ https://github.com/feast-dev/feast
https://github.com/featureform/featureform