CICD

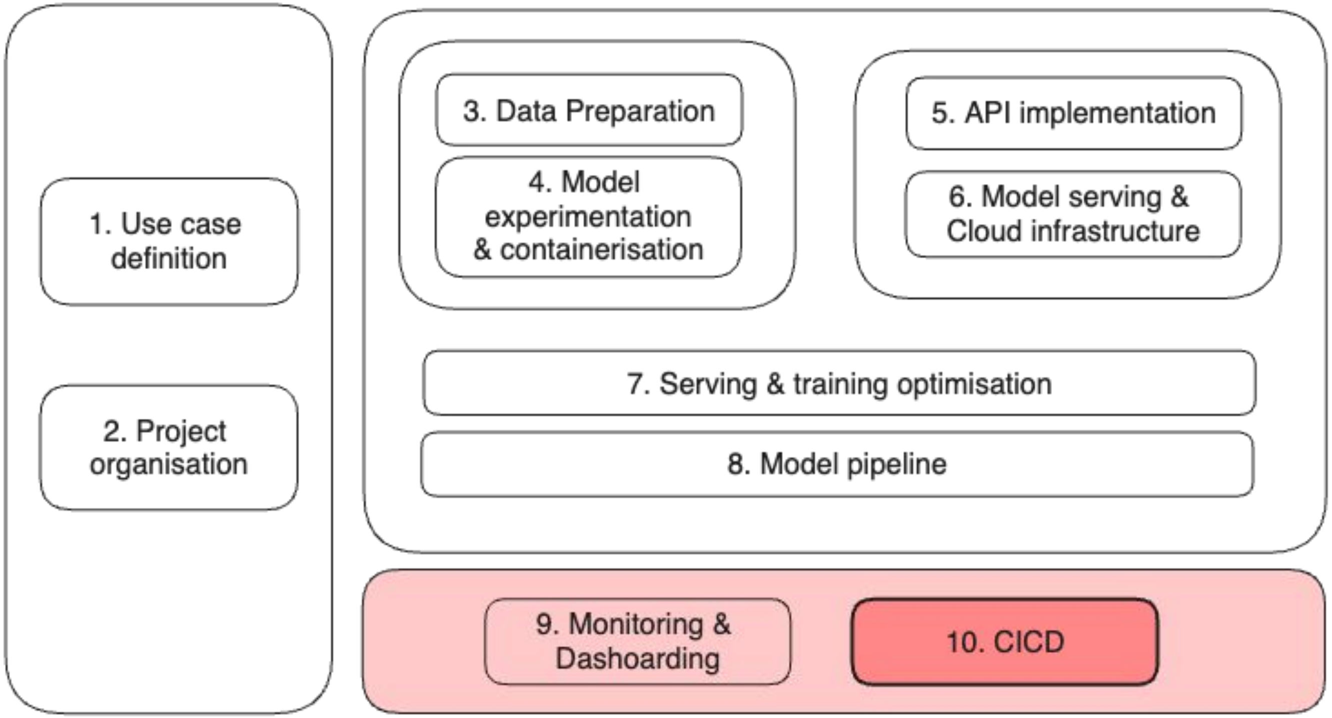
Sprint 5 - Week 10

INFO 9023 - Machine Learning Systems Design

2024 H1

Thomas Vrancken (t.vrancken@uliege.be)
Matthias Pirlet (matthias.pirlet@uliege.be)

Status on our overall course roadmap



LIÈGE université

Agenda

What will we talk about today

Lecture (1:15 hour)

1. CICD
2. Code testing
3. Environment management
4. Infrastructure as Code (IaC)

Lab (30 min)

5. Github Actions

Wrap-up last week

Which type of drift is this?

A model used to predict traffic patterns based on time of day, day of the week, and weather conditions might experience drift if there's a permanent change in road infrastructure (like the addition of new lanes or new traffic regulations) or a long-term change in weather patterns due to climate change.

LIÈGE université

Which type of drift is this?

In predictive maintenance of machinery, a model might predict failures based on sensor readings like temperature, vibration, and pressure. Drift could occur if the machinery is upgraded or if the materials used in the manufacturing process change, altering the failure modes despite similar sensor readings.

Which type of drift is this?

Suppose a machine learning model predicts credit card fraud based on features like purchase amount, location, and time. Over time, the introduction of new payment technologies and the shift in consumer spending behavior due to seasonal trends or economic changes can cause significant changes in these features' distributions.

LIÈGE université

Which type of drift is this?

In a real estate pricing model, the target variable is the price of properties. If there's an economic boom or downturn, the average selling price of houses might increase or decrease independently of the features such as location, size, or condition that the model uses to predict prices.

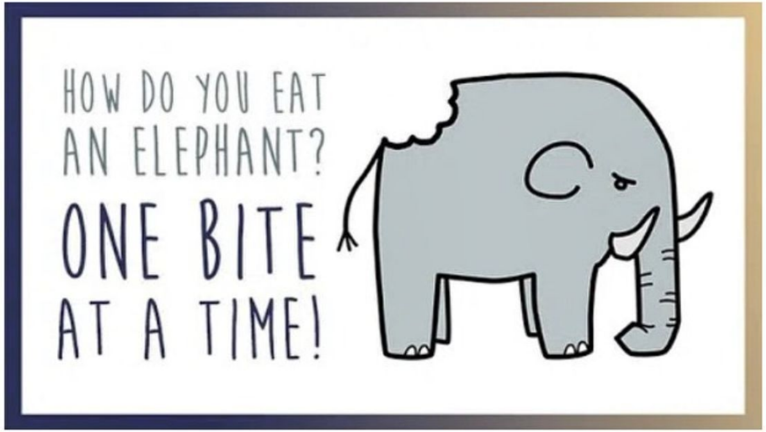
CICD

Why do we need CICD

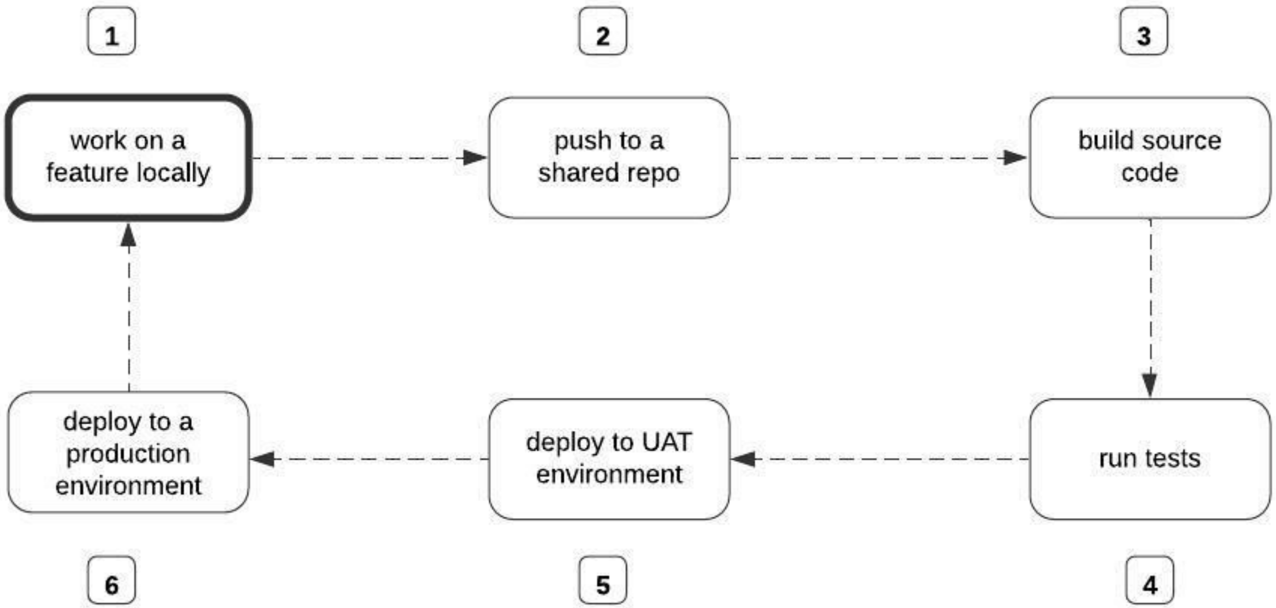
It is better to ship a small change than shipping a GIANT change.

Making frequent changes mean having to *frequently redeploy* your solution ⇒ Automate **deployment**

You want to make sure that the *system works before deploying it* ⇒ Automate **testing**



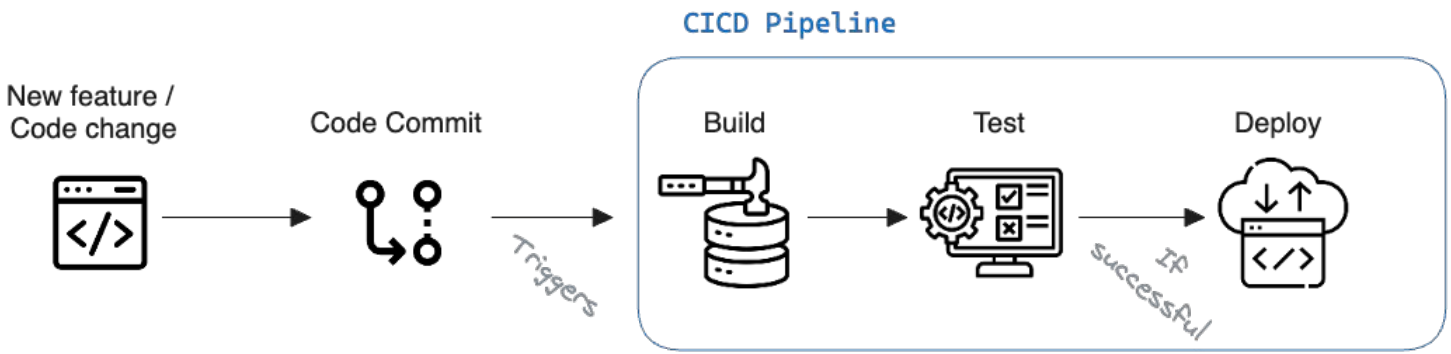
Example of a CICD workflow



https://www.freecodecamp.org/news/how-to-setup-a-ci-cd-pipeline-with-github-actions-and-aws/

Continuous Integration and Continuous Delivery / Deployment

Allows you to continuously work on your application and efficiently deploy new changes to it.



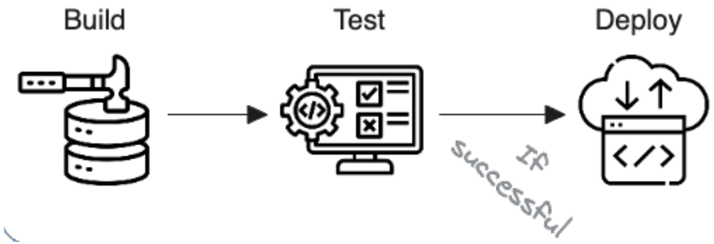
Definition of CICD

Continuous Integration (**CI**), is a software development practice in which all developers *merge code changes* in a central repository multiple times a day.

Continuous Delivery (**CD**) which on top of Continuous Integration adds the practice of automating the entire *software release process*.

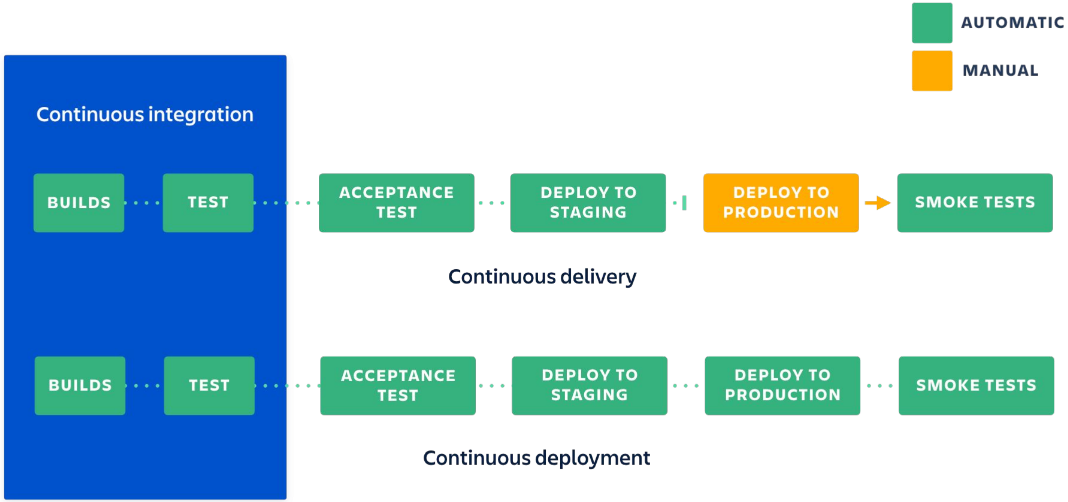
Similar to **Continuous Deployment** but in this latter case the pipeline also *deploys resources to production* automatically (which is done manually for Continuous Delivery)

Three standard *stages* of a CICD pipeline:



https://semaphoreci.com/blog/cicd-pipeline

Continuous Delivery vs Continuous Deployment



https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

CICD: Build stage

We combine the source code and its dependencies to build a runnable instance of our product that we can potentially ship to our end users.

Build the **Docker containers** we covered in a previous lecture.

Failure to pass the build stage is an indicator of a fundamental problem in a project's configuration, and it's best to address it immediately.

CICD: Test stage

(covered in next section).

CICD: Deploy stage

If our codes passed the previous steps, we can **deploy** the feature we built.

For example, if we updated the logic of an API this part of the pipeline would deploy the API in the Cloud. That could also be a ML model training pipeline.

In development we typically have resources in different **environments**.

Example CICD pipeline with Github Actions

```
name: Build and Test

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2
    - name: Set up Python Environment
      uses: actions/setup-python@v2
      with:
        python-version: '3.x'
    - name: Install Dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run Tests
      run: |
        python manage.py test
```

```
deploy:
    needs: [test]
    runs-on: ubuntu-latest

    steps:
    - name: Checkout source code
      uses: actions/checkout@v2

    - name: Generate deployment package
      run: zip -r deploy.zip . -x '*.git*'

    - name: Deploy to EB
      uses: einaregilsson/beanstalk-deploy@v20
      with:

        // Remember the secrets we embedded? this is how we access them
        aws_access_key: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws_secret_key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

        // Replace the values here with your names you submitted in one of
        // The previous sections
        application_name: django-github-actions-aws
        environment_name: django-github-actions-aws

        // The version number could be anything. You can find a dynamic way
        // Of doing this.
        version_label: 12348
        region: "us-east-2"
        deployment_package: deploy.zip
```

LIÈGE université

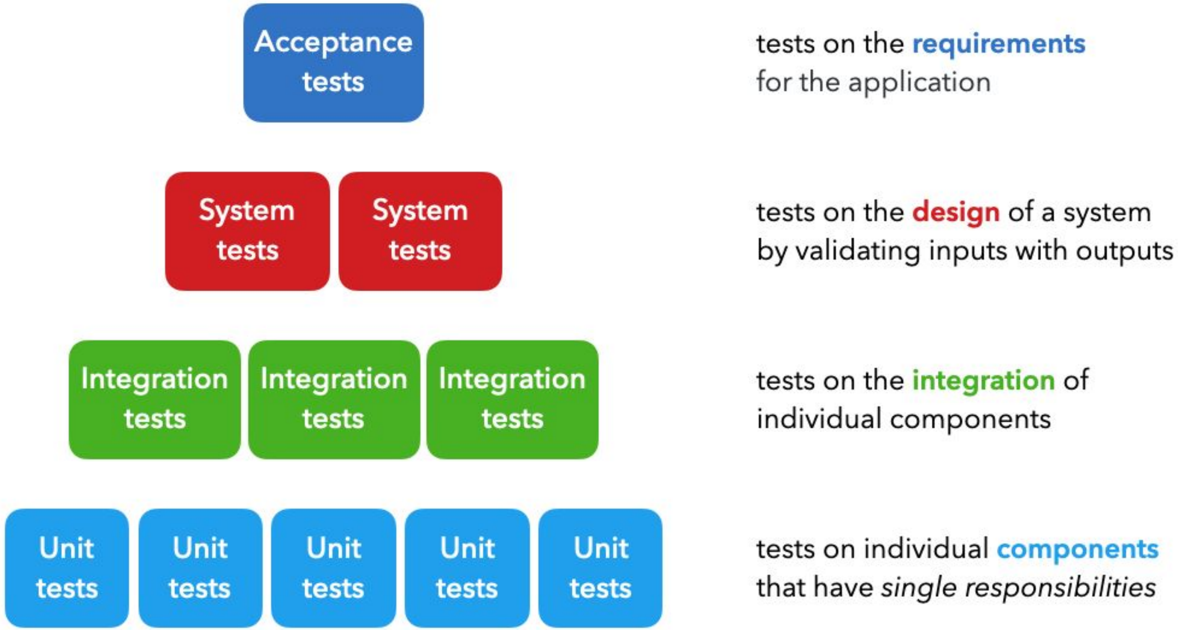
https://www.freecodecamp.org/news/how-to-setup-a-ci-cd-pipeline-with-github-actions-and-aws/

Code testing

Why should we have automatic code tests?

- **Catch bugs early**: Identify problems early in the development cycle, making it easier and cheaper to fix them.
- **Facilitate Refactoring**: Makes it safer to apply changes, as developers know that there is a safety net to deploying changes.
- **Improve Code Quality**: Writing tests forces developers to consider edge cases and error conditions, leading to more robust code.
- **Documentation**: Tests can serve as documentation, showing how a piece of code is intended to be used.

Different types of code testing levels



Qualitative. *E.g. user testing*

Quantitative. *E.g. training, inference, …*

E.g. Data processing

E.g. A single responsibility function

https://madewithml.com/courses/mlops/testing/

How to compose tests?

Input → Component → Output

Arrange	Act	Assert
Set up the different inputs to test on.	Apply the inputs on the component we want to test.	Confirm that we received the expected output.

https://wiki.c2.com/?ArrangeActAssert

🧠 Testing best practices

- **Atomic**: Single Responsibility Principle (SRP) states that “a module (or function) should be responsible to one, and only one, actor” → Allows for *clear testing*
- Compose: Create tests as you implement methods! Catch errors early on and reliably.
- Reuse: Reuse similar tests across different projects (can maintain a single repo for tests)
- Regression: Test against known errors. If a new error occur → Create a new test for it to prevent it from happening in the future.
- Coverage: We want to ensure 100% coverage for our codebase. This doesn't mean writing a test for every single line of code but rather accounting for every single line.
- Automate: Not only run tests manually but also as part of, for example, your CICD pipelines.

https://madewithml.com/courses/mlops/testing/

CICD: Test stage

Let’s look at two **tools** to enable two **types** of testing

1. Functionality tests with **Pytest**
2. Code quality check with **Pylint**

Pytest

Pytest makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries.

By default, pytest identifies files starting with test_ or ending with _test.py as test files.

```python
# my_math_module.py

def add(a, b):
    """Add two numbers together."""
    return a + b
```

```python
# test_my_math_module.py

from my_math_module import add

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(-1, -1) == -2
```

LIÈGE université

https://docs.pytest.org/en/8.0.x/

Pytest

You can then run your tests by just using the `pytest` command in your root directory.

```
● ➜  pytest git:(main) ✗ pytest
======================================== test session starts =========================================
platform darwin -- Python 3.12.2, pytest-8.1.1, pluggy-1.4.0
rootdir: /Users/thomasvrancken/Documents/project/ulg/github/info9023-mlops/my_labs/misc/pytest
collected 1 item

test_my_math_module.py .                                                                        [100%]

========================================= 1 passed in 0.00s ==========================================
○ ➜  pytest git:(main) ✗
```

Pytest

```python
# test_my_math_module.py

from my_math_module import add

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(-1, -1) == -2
    assert add(-1, 10) == -2
```

Pytest

```python
# test_my_math_module.py

from my_math_module import add

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(-1, -1) == -2
    assert add(-1, 10) == -2
```

```
⊗ → pytest git:(main) ✗ pytest
========================================== test session starts ==========================================
platform darwin -- Python 3.12.2, pytest-8.1.1, pluggy-1.4.0
rootdir: /Users/thomasvrancken/Documents/project/ulg/github/info9023-mlops/my_labs/misc/pytest
collected 1 item

test_my_math_module.py F                                                                          [100%]

================================================= FAILURES =================================================
_________________________________________________ test_add _________________________________________________

    def test_add():
        assert add(2, 3) == 5
        assert add(-1, 1) == 0
        assert add(-1, -1) == -2
>       assert add(-1, 10) == -2
E       assert 9 == -2
E        +  where 9 = add(-1, 10)

test_my_math_module.py:9: AssertionError
========================================= short test summary info ==========================================
FAILED test_my_math_module.py::test_add - assert 9 == -2
============================================= 1 failed in 0.04s ============================================
```

Pytest

```
import pytest

# Use the parametrize decorator to test a list of arguments
@pytest.mark.parametrize('num1, num2, expected',[(3,5,8),
(-2,-2,-4), (-1,5,4), (3,-5,-2), (0,5,5)])
def test_sum(num1, num2, expected):
        assert sum(num1, num2) == expected
```

One can use the **parametrise decorator** to run the test on a series of parameters. Define a set of tests to run using methods defined in your codes (e.g. API logic).

```
collected 6 items

test_example.py::test_sum[3-5-8] PASSED                                [ 16%]
test_example.py::test_sum[-2--2--4] PASSED                             [ 33%]
test_example.py::test_sum[-1-5-4] PASSED                               [ 50%]
test_example.py::test_sum[3--5--2] PASSED                              [ 66%]
test_example.py::test_sum[0-5-5] PASSED                                [ 83%]
test_example.py::test_sum_output_type PASSED                           [100%]

======================== 6 passed in 0.04 seconds ========================
```

https://medium.com/testcult/intro-to-test-framework-pytest-5b1ce4d011ae

Pytest

- Tests are defined as functions prefixed with `test_` and contain one or more statements that `assert` code produces an expected result or raises a particular error.
- Tests are put in files of the form `test_*.py` or `*_test.py`
- Tests are usually placed in a directory called `tests/` in a package's root.

```
pycounts
├── .readthedocs.yml
├── CHANGELOG.md
├── CONDUCT.md
├── CONTRIBUTING.md
├── docs
│   └── ...
├── LICENSE
├── README.md
├── poetry.lock
├── pyproject.toml
├── src
│   └── ...
└── tests
    ├── test_pycounts.py
```

Pylint

Pylint is a **static code analyser** for Python

⇒ Pylint analyses your code without actually running it. It checks for errors, enforces a coding standard, looks for code smells, and can make suggestions about how the code could be refactored.

Code convention: Python PEP8

(Reminder of code convention)

- Ensures a consistent code quality across a team.
- Set of rules on styling, such as
  - Indentation (4 spaces)
  - Max line length (79 characters)
  - Number of blank lines
    - Top-level function and class definitions: two blank lines
    - Method definitions inside a class: single blank line
    - Extra blank lines may be used (sparingly) to separate groups of related functions
  - Ordering imports
  - …
- Integrate/automate in your code editor (IDE) to make it easy. Often enforced PEP8 during PR submission.
- Developers can be a bit judgy… Make your life easy, adapt clean codes 🦸

https://peps.python.org/pep-0008/

Pylint

```
#!/usr/bin/env python

import string

shift = 3
choice = raw_input("would you like to encode or decode?")
word = (raw_input("Please enter text"))
letters = string.ascii_letters + string.punctuation + string.digits
encoded = ''
if choice == "encode":
    for letter in word:
        if letter == ' ':
            encoded = encoded + ' '
        else:
            x = letters.index(letter) + shift
            encoded=encoded + letters[x]
if choice == "decode":
    for letter in word:
        if letter == ' ':
            encoded = encoded + ' '
        else:
            x = letters.index(letter) - shift
            encoded = encoded + letters[x]

print encoded
```

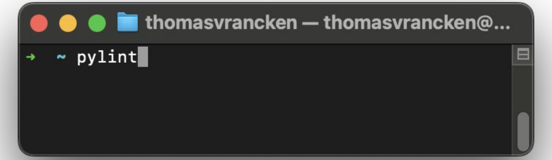
Any mistakes?

https://pylint.pycqa.org/en/1.6.0/tutorial.html

Pylint

Checks for codestyle mistakes. It helps enforcing code best practices (such as PEP8).

```python
1  #!/usr/bin/env python
2
3  import string
4
5  shift = 3
6  choice = raw_input("would you like to encode or decode?")
7  word = (raw_input("Please enter text"))
8  letters = string.ascii_letters + string.punctuation + string.digits
9  encoded = ''
10 if choice == "encode":
11     for letter in word:
12         if letter == ' ':
13             encoded = encoded + ' '
14         else:
15             x = letters.index(letter) + shift
16             encoded=encoded + letters[x]
17 if choice == "decode":
18     for letter in word:
19         if letter == ' ':
20             encoded = encoded + ' '
21         else:
22             x = letters.index(letter) - shift
23             encoded = encoded + letters[x]
24
25 print encoded
```




```
robertk01 Desktop$ pylint simplecaeser.py
No config file found, using default configuration
************* Module simplecaeser
C:  1, 0: Missing module docstring (missing-docstring)
W:  3, 0: Uses of a deprecated module 'string' (deprecated-module)
C:  5, 0: Invalid constant name "shift" (invalid-name)
C:  6, 0: Invalid constant name "choice" (invalid-name)
C:  7, 0: Invalid constant name "word" (invalid-name)
C:  8, 0: Invalid constant name "letters" (invalid-name)
C:  9, 0: Invalid constant name "encoded" (invalid-name)
C: 16,12: Operator not preceded by a space
            encoded=encoded + letters[x]
                   ^ (no-space-before-operator)
```


https://pylint.pycqa.org/en/1.6.0/tutorial.html

Best practice: Local pre-commit

pre-commit package.

```
(venv) ➜  madewithml git:(dev) ✗ git add .
(venv) ➜  madewithml git:(dev) ✗ git commit -m "added pre-commit hooks"
trim trailing whitespace.................................................Passed
fix end of files.........................................................Passed
check for merge conflicts................................................Passed
check yaml...............................................................Passed
check for added large files..............................................Passed
check yaml...............................................................Passed
clean....................................................................Passed
```

When should you use testing?

Is it more important when using **Gitflow** or **trunk based** code versioning?

Environment management

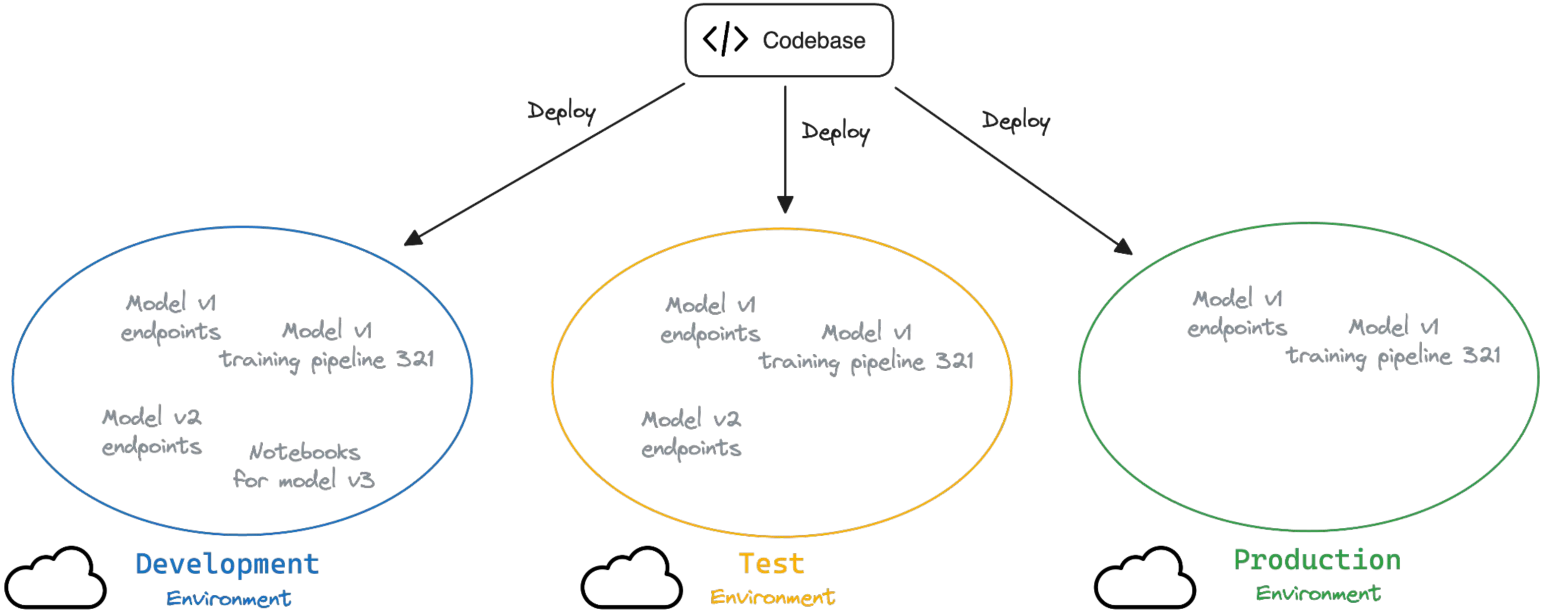
Using multiple environments

In CICD we want to **quickly** and **safely** deliver new features/changes to production. You can deliver small changes frequently, which reduces the risk of problems.

Other factors affect "deployment pain to production", including your adoption of multiple delivery/deployment environments.

A **multi-environment approach** lets you build, test, and release code with greater speed and frequency to make your deployment as straightforward as possible. You can remove manual overhead and the risk of a manual release, and instead automate development with a multistage process targeting different environments.

Using multiple environments



List of common environments

Environment	Description
Development	Your development environment (dev) is where changes to software are developed.
Test	Your test environment allows either human testers or automated tests to try out new and updated code. Developers must accept new code and configurations through unit testing in your development environment before allowing those items to enter one or more test environments.
Staging	Staging is where you do final testing immediately prior to deploying to production. Each staging environment should mirror an actual production environment as accurately as possible.
Acceptance	User Acceptance Testing (UAT) allows your end-users or clients to perform tests to verify/accept the software system before a software application can move to your production environment.
Production	Your production environment (production), sometimes called *live*, is the environment your users directly interact with.

https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/considerations/environments

Best practices for multi environments

A few best practices

- **Test environments** are important because they allow platform developers to test changes before deploying to production, which reduces risk related to delivery in production.
- Keep your **environments** as **similar** as possible! Helps for reproducibility and finding environment related errors.
- If there are discrepancies in the configuration of your environments, "**configuration drift**" happens, which can result in data loss, slower deployments, and failures.
- Consider adopting methods like **A/B** or **Canary** Deployments that make new features available only to a limited set of test users in production and help reduce the time to release into production.
- **Avoid silos** by allowing all developers to access all environments. (--> Careful with prod)
- You can speed up deployments, improve environment consistency, and reduce "configuration drift" between environments by adopting **Infrastructure as Code** (IaC).

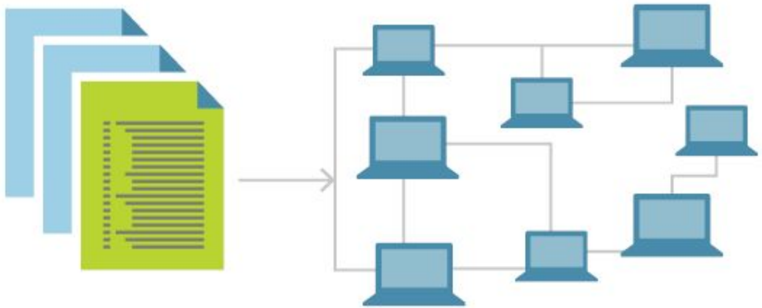
https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/considerations/environments

Infrastructure as Code (IaC)

Best practice is to create resources using codes instead of manual steps.

Infrastructure as code (IaC) uses DevOps methodology and versioning to define and deploy infrastructure, such as networks, virtual machines, load balancers, and connection topologies using **codes**.

Just as the same source code always generates the same binary, an IaC model generates the same environment every time it deploys.



https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code

IaC benefits

- **Speed** and **efficiency**: Infrastructure as code enables you to *quickly* set up your complete infrastructure *across different environments* by running a script(s).
- **Consistency**: Manual processes result in mistakes and discrepancies… Configurable scripts are much safer.
- **Accountability**: Since you can version IaC configuration files like any source code file, you have full traceability of the changes each configuration suffered.
- **Lower Cost**: Lowering the costs of infrastructure management. Free up developer time from doing repetitive manual tasks

https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/

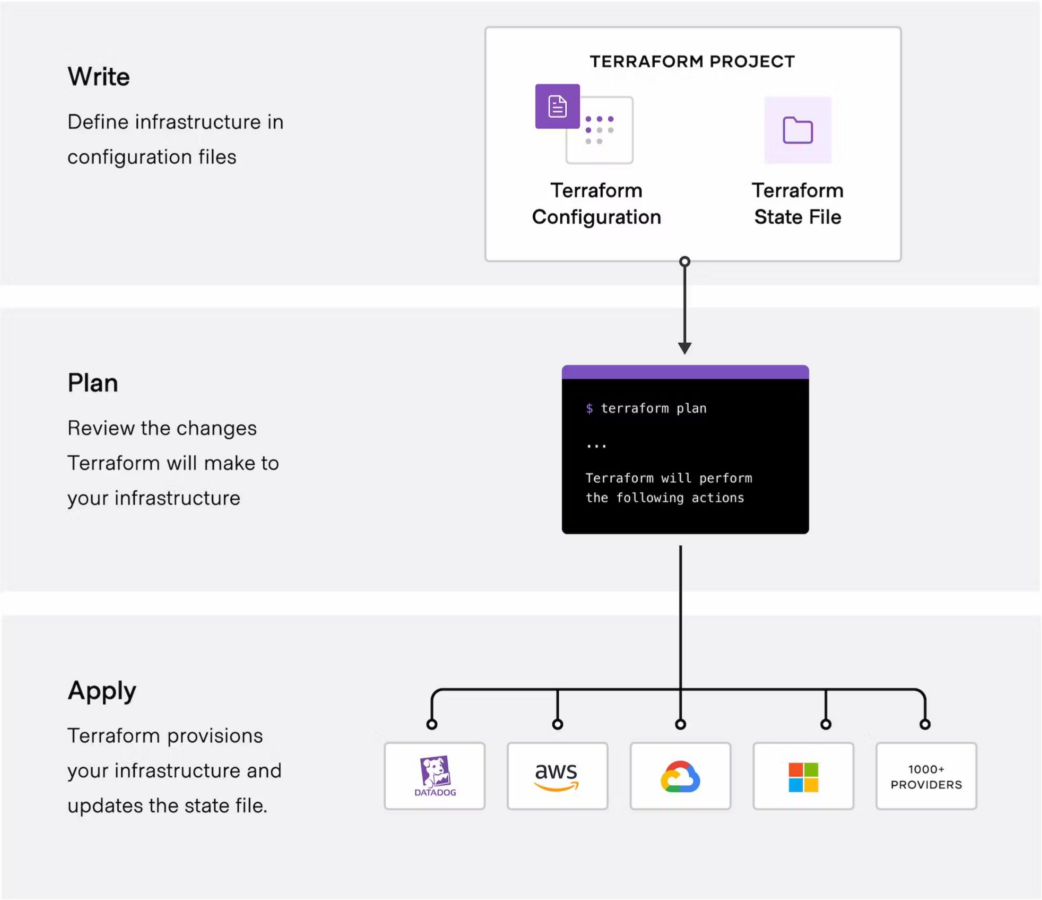
Terraform

Terraform is an **infrastructure as code** tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share.

- Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.
- Terraform creates and manages resources on cloud platforms and other services through their APIs. Providers enable Terraform to work with virtually any platform or service with an accessible API.

https://developer.hashicorp.com/terraform/intro

Terraform



https://developer.hashicorp.com/terraform/intro

Terraform: Simple example for Cloud Run

```
#main.tf

# Enable the Cloud Run API
resource "google_project_service" "run_api" {
  service = "run.googleapis.com"
  disable_on_destroy = true
}

# Create the Cloud Run service
resource "google_cloud_run_service" "run_service" {
  name = "app"
  location = "us-central1"

  template {
    spec {
      containers {
        image = "us-central1-docker.pkg.dev/someproject-123/docker-repo/fast-api:1.0"
      }
    }
  }

  traffic {
    percent         = 100
    latest_revision = true
  }

  # Waits for the Cloud Run API to be enabled
  depends_on = [google_project_service.run_api]
}
```

```
# output.tf

output "service_url" {
  value = google_cloud_run_service.run_service.status[0].url
}
```

thomasvrancken — thomasvrancken@Thomass-MacBook-Pro — ~ —...

```
➜  ~ terraform init # initializing terraform plugins
terraform plan # checking the plan
terraform apply --auto-approve # Deploying resources
```

Why Terraform?

Manage any infrastructure: Terraform is versatile and supported by all the main Cloud providers.

Track your infrastructure

- Terraform generates a plan and prompts you for your approval before modifying your infrastructure. It also
- keeps track of your real infrastructure in a state file, which acts as a source of truth for your environment.

Automate changes:

- Terraform configuration files are **declarative** (describe the end state of your infrastructure, not the steps needed to get there)
- Terraform builds a resource graph to determine resource dependencies and creates or modifies non-dependent resources in parallel ⇒ provision resources efficiently.

Standardize configurations: Define reusable **modules** that define configurable collections of infrastructure,

Collaborate: Since your configuration is written in a file, you can commit it to Git and use Terraform Cloud to efficiently manage Terraform workflows across teams

https://developer.hashicorp.com/terraform/intro

Lab: Github Actions

Notes

- Explain that code versioning tools such as Github, Gitlab or bitbucket all offer this type of pipelines
- https://mlops.githubapp.com/

Bonus: A guide to trustworthy AI

Credits where credits are due

Pauline Nissen
Ethical AI Lead @ ML6

Trustworthy Artificial Intelligence
Introduction

Trustworthy AI describes AI that is **lawful** compliant, **ethically** responsible and technically **secure**.

lawful

ethical

secure

The concept is grounded on the premise that AI will reach its full potential only when **trust** can be established **throughout its entire lifecycle**, from conception and development to deployment and usage.

The High-Level Expert Group on AI presented the 7 dimensions **framework** for Trustworthy AI.

2019
April

They presented the Assessment List for Trustworthy AI (ALTAI), which is a practical tool that translates the 7 dimensions into a **self-assessment checklist**.

2020
July

ML6

Trustworthy Artificial Intelligence

Internal framework

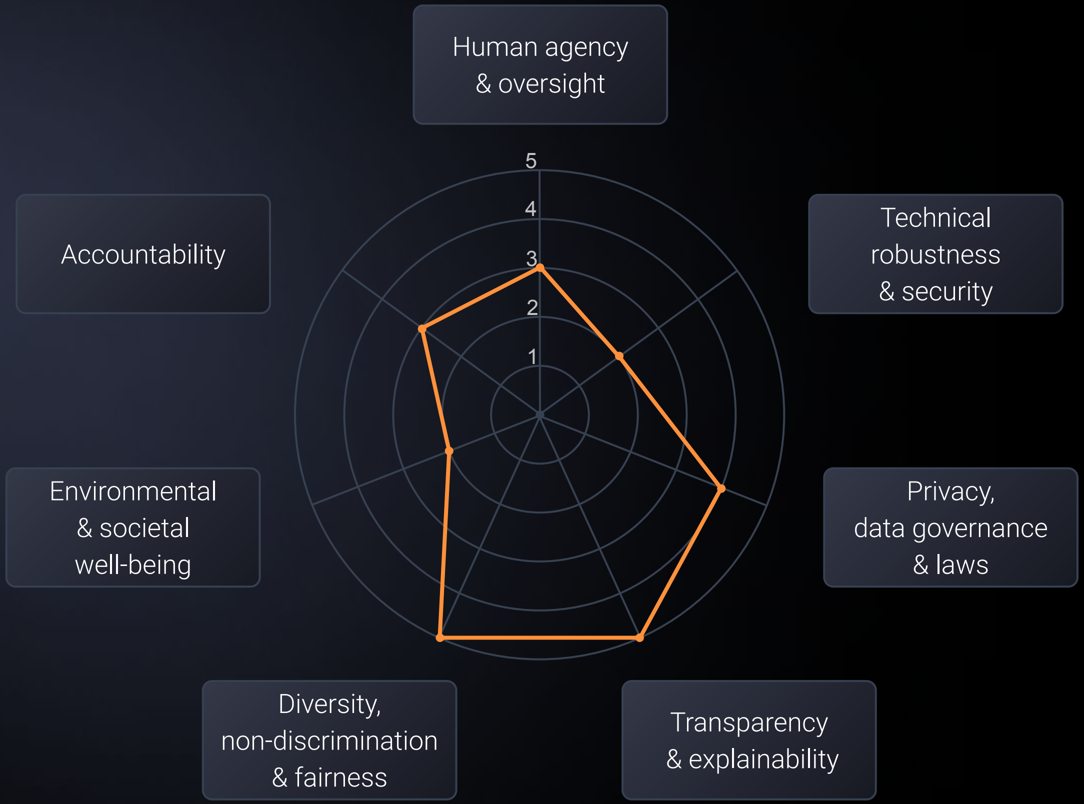
1. **Human agency & oversight:** Including fundamental rights, human agency and human oversight.
2. **Technical robustness & security:** Including security, safety, accuracy, reliability and reproducibility.
3. **Privacy, data governance & laws:** Including respect for privacy, quality and integrity of data, access to data and laws..
4. **Transparency & explainability:** Including traceability, explainability and communication.
5. **Diversity, non-discrimination & fairness:** Including the avoidance of unfair bias, accessibility and universal design
6. **Environmental & societal well-being:** Including sustainability, social impact, society and democracy.
7. **Accountability:** Including auditability, minimisation and reporting of negative impact, trade-offs and redress.

ML6

Trustworthy Artificial Intelligence

In practice at ML6

Our internal framework for Trustworthy AI can be used to **identify, prevent and mitigate** potential legal, ethical and security risks, through ongoing evaluations **using a 5-point scale**.



ML6

1. Human agency & oversight

Human agency
Context

The main difference between the industrial revolution and the AI revolution is that AI systems are **more involved in decision-making**.

With the rise of automation, humans' role has shifted from direct control to **supervising AI systems**, influencing their sense of control.

Human agency deals with risks, such as **overreliance**, and **unintended interference**, that may arise when **interacting with AI systems**.

Human agency

Recognizing & mitigating overreliance on AI

Overreliance on AI happens when the users rely too much on AI systems. They place **excessive trust in AI outputs**, sometimes neglecting to use their own expertise to critically evaluate the AI recommendation.

This can lead to problems, such as **overlooking potential AI errors** and **skills atrophying** (eg. GPS navigation).

	Human rejects AI decision	*Human accepts AI decision*
AI correct	Underreliance	Appropriate reliance
AI incorrect	Appropriate reliance	**Overreliance**

An underlying reason for overreliance may be that the users don't want to challenge each individual AI outcome, losing time and efficiency. Instead, the users develop **general heuristics** about whether and when to follow the AI outcome. A strategy for overreliance is to disrupt the quick heuristic decision-making (i.e. **cognitive forcing)**. Examples:

- Adding a confidence indicator to the AI output.
- Asking the user to consider alternative solutions where the AI output may be wrong.
- Adding an indicator of expected output, with warnings ithe the output is outside its boundaries.

Human oversight
Context

Humans play a crucial role in ensuring that AI systems work with **ethical boundaries** and serve the interests of users in a balanced way.

Depending on the degree of autonomy and potential for AI risks, there are **different modes of human oversight** that can be enacted.

Detection and **response mechanisms** can be used to identify and manage adverse effects when users interact with AI systems.

Human oversight

Modes of human oversight

Human in-the-loop

The **human** is **actively involved** in the decision making process, relying on AI systems to enhance their capabilities. The human has the authority to accept or override AI suggestions.

E.g. AI tool suggesting a draft article using data, but the journalist writes the final article.

Human in-command

The human has **full control and authority** over decision-making. The AI system only performs actions that are explicitly authorised by a human. All major **decisions** are **made by humans**.

E.g. AI robot suggesting a surgical approach, but not performing unless authorised by surgeon.

Human on-the-loop

The human **monitors the AI system's** actions and **intervenes as needed**. They have a supervisory role, ensuring the AI aligns with broader goals and can handle unexpected scenarios.

E.g. AI system alerting fraudulent activities to human who can investigate and intervene.

Human out of-the-loop

The human is **not actively involved** in daily operations. They may have set initial parameters or guidelines for the AI, but once deployed, the system works **without human oversight**.

E.g. AI system providing weather forecasts without direct human involvement.

Reliability, reproducibility & fall-back plans

Context

Reliability is the ability of an AI system to **consistently produce trustworthy results** under various conditions. **Reproducibility** is the capability **replicate results** using the same settings.

Unreliability or low degree of reproducibility can have a **negative impact**. Monitoring, verification and documentation are important.

Fallback plans are important to ensure that, when **anomalies** are detected, there is a **clear protocol** to mitigate potential damage.

Reliability, reproducibility & fall-back plans

Best practices for fall-back plans

Monitoring and alerts: Continuous monitoring of the AI system can detect anomalies or performance drops. Automated alerts can notify relevant teams immediately when predefined thresholds are breached.

Decision protocols: Clearly defined protocols should be in place to determine when to switch to the fallback system. This could be based on the severity of the malfunction, the potential impact or a combination of factors.

Regular drills: Just like fire drills, organisations should conduct regular failsafe drills. This ensures that in the event of a real crisis, team know exactly what to do, minimising response times.

Feedback loops: After activating a fallback plan, there should be mechanisms to gather data on what went wrong with the primary system. This feedback can be invaluable for preventing future failures.

Stakeholders communication: Clear communication channels should be established to inform stakeholders about any disruption and the activation of fallback plans. Transparency in such situations can mitigate panic and confusion.

Review and update: Fallback plans should not be static. They should be regularly reviewed and updated based on technological advancements.

Reliability, reproducibility & fall-back plans
ALTAI questions

18. Could the AI system cause critical, adversarial, or damaging consequences (e.g. pertaining to human safety) in case of **low reliability and/or reproducibility**?
    a. Did you put in place a well-defined process to monitor if the AI system is meeting the intended goals?
    b. Did you test whether specific contexts or conditions need to be taken into account to ensure reproducibility?

18. Did you put in place verification and validation methods and documentation (e.g. logging) to evaluate and ensure different aspects of the AI system's **reliability** and **reproducibility**?
    b. Did you clearly document and operationalise processes for the testing and verification of the reliability and reproducibility of the AI system?

18. Did you define tested **failsafe fallback plans** to address AI system errors of whatever origin and put governance procedures in place to trigger them?

21. Did you put in place a proper procedure for handling the cases where the AI system yields results with a **low confidence score**?

21. Is your AI system using (online) **continual learning**?
    a. Did you consider potential negative consequences from the AI system learning novel or unusual methods to score well on its objective function?

Transparency & explainability

Context

Transparency is important for **building and maintaining user's trust** in AI systems.

Transparency comes in 2 ways: transparency on the outcomes (**what**) and transparency on the processing of getting that outcome (**how**)

Transparency can be broken down in 3 aspects: **traceability**, **explainability** and **communication**.

Avoidance of unfair bias

Context

AI systems should ensure **equitable distribution of benefits and costs**, ensuring freedom from bias, discrimination and stigmatisation.

Decisions by AI systems should be **transparent** and **contestable**, with clear accountability and explainable processes.

Ensuring AI systems are trained on **diverse** and **representative** datasets is essential for producing unbiased results.

Avoidance of unfair bias

Definition of fairness, bias & discrimination

“ **Fairness** exists when AI system’s outcomes do **not disproportionately favor or disadvantage** any subgroup of a dataset based on attributes that are independent of the selection criterion.
Example: an hiring solution ensures fairness by evaluating candidates solely on job-related criteria, without favoring or discriminating against any demographic group based on irrelevant attributes like gender or ethnicity.

“ **Bias** occurs when AI systems’s outcomes consistently deviate from the actual values it’s trying to estimate.
Example: an hiring solution contain bias if the outcomes unintentionally discriminate against female candidates, because the algorithm is trained on historical data from a company where there has been a gender imbalance in hiring practices.

“ Bias is often equated with **discrimination** which refers to the unjust or prejudicial treatment of individuals based on attributes, like race or gender. It carries a heavy connotation, suggesting malicious intent towards certain groups.
Example: Workplace discrimination occurs when employees are treated differently based on factors like race, gender, or age, affecting their professional advancement.

Bonus: Gen AI and its impact

LLMs and multimodal models are greatly changing ML engineering

Text classification

- **Statistical era**: Use statistical techniques.
  - TF-IDF or Bag of Words with any type of classifier (XGBoost)
- **Deep learning era:** Train your own model
  - E.g. LSTM on Keras. Maybe using Word2Vec or so for word embeddings.
- **Transformers era**: Fine-tune a pre-trained Transformers model
  - E.g. BERT on Huggingface - already incorporates a lot of language understanding
- **LLM era**: Few-shot prompting with an LLM
  - GPT4 with a few examples included in the prompt.

LLMs and multimodal models are greatly changing ML engineering

Consequence

Potential

AI is reaching a new potential!
This giant leap in performance creates a lot of opportunities to do good with ML !
There is a high demand for LLM integration.

Less custom training

Machine Learning Engineers don't need to spend as much time training or fine-tuning custom models.

Open-source

Initial wave of LLMs are private (kind of a first in the ML world!).

Platforms such as Huggingface make it attractive and easy to open source and benchmark models.

Hard to compete with tech giants in terms of data and compute access.

Integration

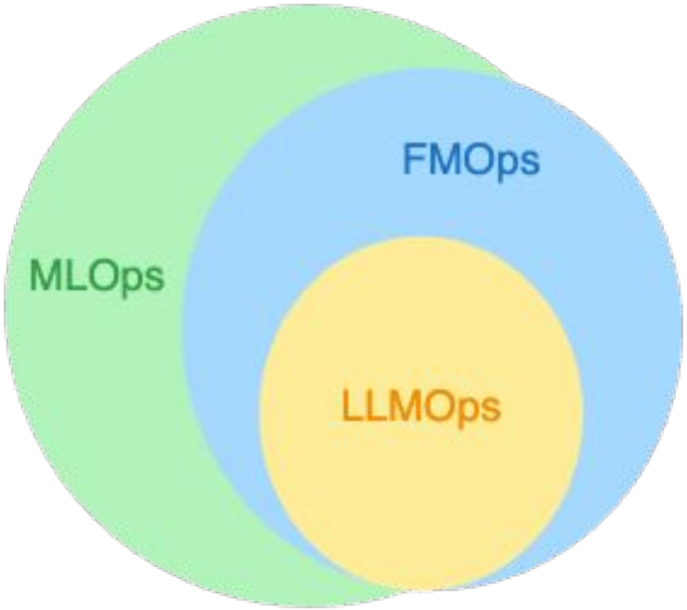
Larger models mean more optimisation require in terms of training and serving.

FMOps & LLMOps

MLOps *(Machine Learning Operations)* - Productionize ML solutions efficiently.

FMOps *(Foundational Model Operations)* - Productionize GenAI solutions efficiently (text, image, sound, videos, …). Based on large foundational models.

LLMOps *(Large Language Model Operations)* - Productionize large language models solutions.



https://pages.awscloud.com/rs/112-TZM-766/images/AWS_FMOps_LLMOps_Operationalise_GenAI_using_MLOps_principles.pdf

FMOps introduced a few new concepts

Processes & People
Providers, fine-tuners, & consumers

Select & Adapt the FM on a Specific Context
- Fine-tuning, parameter-efficient fine-tuning, prompt engineering
- Proprietary, open source based on the application

Evaluate & Monitor Fine-tuned Models
Human feedback, prompt management, toxicity/bias...

Data & Model Deployment
Data privacy, multi-tenancy, & cost, latency, and precision

Technology
MLOps, data, & application layers

Wrap-up

Lecture summary

Topic	Concepts	To know for…	
		Project	Exam
CICD	• What is CICD • Difference between continuous integration, continuous delivery and continuous deployment • CICD stages: Build, Test and Deploy		Yes
Unit testing	• Pytest • Pylint		Yes
Environment Management	• Types of environments • When to use which		Yes
Infrastructure as Code	• What IaC is • Terraform		
Bonus	• A guide to trustworthy AI • Bonus: Gen AI and its impact (FMOps and LLMOps)		
Lab: Git Actions		Yes	

Project objective for sprint 5

You can decide which steps of your CICD pipeline are relevant to implement.

#	Week	Work package	Requirement
5.1	W09	Build a dashboard that runs either locally on in the Cloud to show your results	Optional
5.2	W10	Build a CICD pipeline using Github Actions (or other tool) to automatically run some of the following steps. Include at least one step. The rest is optional. Up to you to decide what is relevant.	Required
5.3	W10	Include step in CICD: Automatically launch model training pipeline	Optional
5.4	W10	Include step in CICD: Automatically launch model deployment	Optional
5.5	W10	Include step in CICD: Pylint	Optional
5.6	W10	Include step in CICD: Pytest for any unit test you think is relevant	Optional

LIÈGE université