**Module 7:**

# ML WORKFLOW BEST PRACTICE - READER

Welcome to the ML4EO Module 7 on machine learning (ML) Workflow Best Practice. This module heavily builds on the skills and practices presented in our previous modules.

As the field of ML continues to evolve, it is crucial to stay up-to-date with best practices and techniques for efficient model development, experimentation, and deployment. This module is designed to provide you with a thorough understanding of key concepts and methods to streamline your ML projects, with a particular focus on the earth observation (EO) context. These methods go beyond what is usually covered in academic curricula, concentrating on workflows utilized widely in industry.

In **section 7.1** we touch first on a further technical aspect of ML when training models based on large EO datasets. For this we will explore **Efficient Hyperparameter Tuning**. This session will help you understand the importance of optimizing resource usage in computer vision tasks and explain common approaches to efficient hyperparameter tuning.

Next, we will dive into **Experiment Tracking in section 7.2**, where we emphasize the importance of taking a systematic approach to machine learning projects. We will cover the core elements of experiment tracking, discuss common approaches and specific toolkits, and demonstrate the process of training and tuning hyperparameters of a model using an experiment tracking workflow.

Lastly, in **section 7.3 we discuss Continuous Integration and Continuous Deployment (CI/CD) for ML workflows**. In this session, you will learn about the benefits of applying CI/CD principles to ML workflows and understand the unique aspects of CI and CD for ML pipelines.

By the end of Module 7, you will have a solid understanding of efficient hyperparameter tuning, experiment tracking, and CI/CD in ML workflows. These skills will prove invaluable as you navigate the rapidly evolving landscape of machine learning and Earth observation data analysis.

**Learning objectives** - Participants will:

* Understand the importance of optimizing resource usage in computer vision tasks
* Explain common approaches to efficient hyperparameter tuning
* Understand the importance of taking a systematized approach to ML
* Explain the core elements of experiment tracking
* Describe common approaches to experiment tracking including specific toolkits
* Demonstrate capacity to train and tune the hyperparameters of a model using an experiment tracking workflow
* Understand the benefits of CI/CD for ML workflows
* Explain specific aspects of CI and CD for ML workflows

# 7.1 Efficient Hyperparameter Tuning for EO Data

Later in this module we will discuss systemized approaches to training different iterations of ML models using various configurations of hyperparameters. As a data scientist, you will hopefully already be familiar with the practice and objective of this concept – i.e. hyperparameter tuning. In previous experience, you may have tuned hyperparameters using different strategies, including manual values or grid search to varying levels of granualarity. In the context of computer vision applications such as EO data modelling, it is important to take a resource-conscious approach to hyperparameter tuning.

**Motivation**

EO datasets are often large, high-dimensional, and multi-spectral, which can significantly impact the training time and compute resource requirements for machine learning models. Efficient hyperparameter tuning methods, such as Bayesian optimization or genetic algorithms, can help reduce the search space and computational load, making it more feasible to optimize models on such complex data.

The figure below demonstrates the efficiency of various hyperparameter tuning methods. As shown, some methods converge much earlier than others and result in better outcomes (ref. the random method, which never reaches the same level of performance as the other algorithms).

A picture containing text, diagram, screenshot, line

Description automatically generated

Figure 1: Comparison of training duration vs. performance for multiple automated hyperparameter tuning methods[[1]](#footnote-2)

**Common Approaches**

Here we summarize common efficient hyperparameter tuning methods with particular regard to the context of models trained on EO data. You may already be familiar with grid search and random search, but when deploying these simple methods with large datasets, it is important to consider the resource impact of the search pattern.

* **Grid Search:** Grid search is a basic method of hyperparameter tuning where you define a subset of possible values for each hyperparameter and evaluate all possible combinations in the subset. For Earth observation tasks, it is common to use a smaller grid and fewer hyperparameters to reduce computational expense. A coarse grid can be used initially, followed by a refined search around the best-performing region.
* **Random Search:** Random search is more efficient than grid search, as it randomly samples the hyperparameter space and evaluates a predefined number of combinations. For Earth observation tasks, this method can often find good hyperparameter settings with fewer evaluations, reducing the computational burden.

While this may seem counterintuitive, consider the figure below. The white oval indicates the optimal hyperparameter value space. By random chance, one of the randomly assigned hyperparameters has hit inside the optimal valuespace, which will result in a better model performance for the model using those hyperparameters. This occurence is more likely for randomly assigned values for the following reason: In grid search, **3 unique values** are assigned (manually) for each hyperparameter. For random search, we can assume that **up to 9 unique values** are assigned for each hyperparameter (as the assignment function samples from a given range). This dynamic provides a wider spread of hyperparameter values compared to the fixed assignments of the grid search. Bottom line: the random search method covers more ground than grid search, for the same resource cost.

A picture containing text, screenshot, design

Description automatically generated

Figure 2: Grid search vs random search in hyperparameter tuning[[2]](#footnote-3)

* **Bayesian Optimization:** Bayesian optimization is well-suited for Earth observation tasks, as it uses a probabilistic model to estimate the performance of different hyperparameter combinations. This method balances exploration and exploitation, focusing on promising regions in the hyperparameter space to achieve better performance with less computation.
* **Tree-structured Parzen Estimators (TPE):** TPE is another probabilistic method that models the likelihood of a set of hyperparameters yielding good performance for Earth observation tasks. By iteratively updating its estimation, TPE intelligently guides the search process, resulting in more efficient exploration of the hyperparameter space.
* **Population-based methods:**
  + **Genetic Algorithms (GA):** Genetic algorithms are well-suited for Earth observation tasks, as they evolve a population of hyperparameter combinations over multiple generations, prioritizing combinations with better performance. This method explores the hyperparameter space more efficiently, which is especially important for large and complex Earth observation models.
  + **Particle Swarm Optimization (PSO):** PSO is a population-based method particularly useful for Earth observation tasks, as it can efficiently explore the hyperparameter space by leveraging the collective knowledge of the swarm. This method is especially beneficial when tuning models with a large number of hyperparameters, such as deep convolutional neural networks applied to satellite imagery.
* **Early stopping and pruning:** To reduce the computational burden of hyperparameter tuning for EO tasks, early stopping and pruning techniques can be applied to terminate poorly performing evaluations before they complete. This allows you to allocate more resources to promising hyperparameter combinations and speed up the overall tuning process.

# 7.2 Experiment Tracking

Experiment tracking in the context of ML refers to the process of systematically recording, organizing, and monitoring various aspects of machine learning experiments during the **model development process**. As machine learning models are iteratively developed and refined, it's essential to keep track of different experimental setups, hyperparameters, data versions, code, performance metrics, and other relevant information. This helps data scientists and ML engineers to analyze the impact of changes, compare different models or approaches, and ultimately make informed decisions to improve the model's performance.

**Motivation**

Coverage of structured approaches to model training in academic courses is often limited to version control and code reproducibility. Data scientists might not encounter systematized approaches until they enter industry, and even then the level of sophistication can vary. It is common for individuals to develop their own custom approaches using Excel or other methods (ref. figure below). These approaches can be effective on the individual level, but don’t translate to well scenarios of higher complexity, and/or projects that require collaboration with other researchers and practitioners.

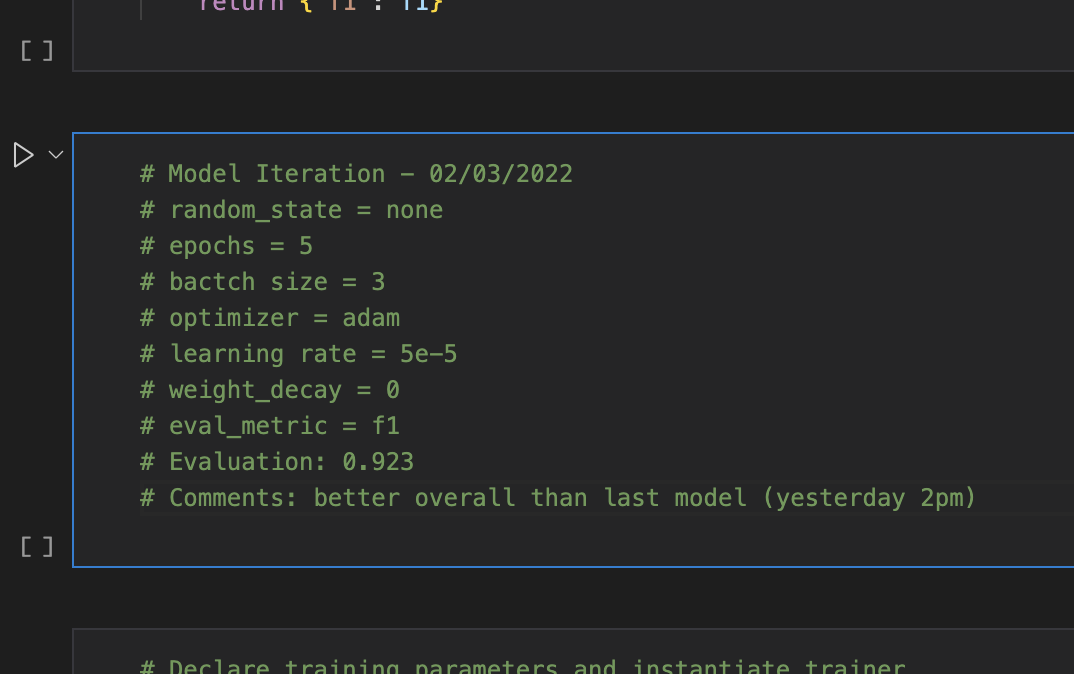


Figure 3: Suboptimal experiment tracking. Metadata is manually noted and unstructured.

In order to frame the use case for experiment tracking, imagine you are a data scientist working on a project to predict crop yields based on satellite imagery and other relevant geospatial data. Your goal is to develop an ML model that can help farmers and policymakers make informed decisions about agricultural practices, resource allocation, and food security.

The project involves several steps, including data collection, preprocessing, feature engineering, model selection, training, evaluation, and deployment. Throughout this process, you will likely try different approaches and techniques to optimize the model's performance, such as:

1. Experimenting with various data preprocessing techniques (e.g., data normalization, atmospheric correction, or image segmentation).
2. Engineering and selecting different features from the satellite imagery (e.g., spectral indices, land cover features, or time-series analysis).
3. Trying different ML algorithms (e.g., Random Forests, Support Vector Machines, or Convolutional Neural Networks) and model architectures.
4. Tuning hyperparameters of the selected models to optimize their performance.
5. Training and evaluating the models using different training/validation/test splits or cross-validation techniques.

Given the complexity of the problem and the **number of potential combinations** of data preprocessing, feature engineering, model selection, and hyperparameter tuning, it is important to approach the task using an organized, systematic workflow to understand the impact of each decision on the model's performance. Experiment tracking can facilitate this by doing the following:

1. Control versioning of your input data, including raw data sources and training datasets
2. Maintain a record of different data preprocessing techniques and feature engineering steps, making it easy to compare their impact on the model's performance.
3. Log hyperparameters and metadata relevant to each training iteration
4. Log and visualize various performance metrics (e.g., accuracy, precision, recall, F1-score) for each experiment, enabling you to identify the best-performing models and techniques quickly.
5. Save and version your trained models, ensuring that you can easily revert to a previous version or compare different models side-by-side.
6. Share your experiments and results with team members, fostering collaboration and knowledge sharing.
7. Improve the reproducibility of your work by capturing the code, environment, and other relevant details for each experiment.

In this scenario, using a systematized approach or an experiment tracking tool (e.g. Weights & Biases) would enable you to systematically explore different approaches, identify the most effective techniques, and ultimately develop a more accurate and robust model for predicting crop yields using satellite imagery and machine learning.

**Core Components of Experiment Tracking**

At the base level, experiment tracking just means systematically **documenting the inputs to a model training iteration, and recording the outputs**. In this section we will discuss the core aspects to consider when manually tracking your own experiments, as well as common tools that facilitate each aspect.The core elements of experiment tracking in ML include:

1. **Version Control (source code, data & model artifacts)**
2. **Documentation (Hyperparameters, Model Performance and Experiment Metadata)**
3. **Visualization**

You are likely already familiar with some of these concepts and may already be utilizing them in your workflows. We elaborate on these aspects in the sections below.

**1. Version Control**

Version control is an essential aspect of the software development process, allowing developers to track changes in their code and collaborate more effectively. In the context of machine learning, version control is more complex in some ways, due to the need to manage not only source code but also large data files and model artifacts.

* **Source Code Version Control:** In ML projects, source code can include scripts for data preprocessing, feature engineering, model training and evaluation. As you experiment with different techniques and fine-tune your models, you will likely modify your code multiple times. Version control systems help you keep track of these changes and allow you to revert to a previous version if needed.

You should already be familiar with the concept of version control using Git and its application on various platforms. This tool enables you to create snapshots of your codebase called "commits," each representing a specific version of the code. You can also create "branches" to work on new features or experiments without affecting the main codebase. When you're ready to incorporate changes from a branch into the main codebase, you can perform a "merge" operation. Git also facilitates collaboration by allowing multiple team members to work on the same codebase simultaneously.

Using Git in combination with online platforms like GitHub or GitLab provides additional features like issue tracking, code review, and continuous integration/continuous deployment (CI/CD) pipelines.

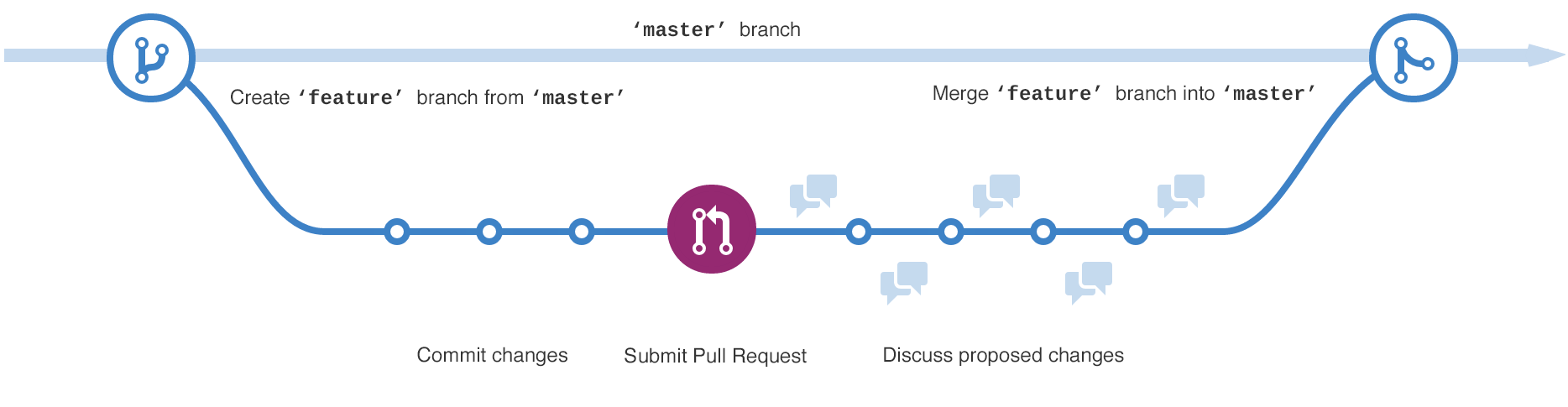


Figure 4: Simple Git Workflow (Github Flow[[3]](#footnote-4))

* **Data Version Control:** While traditional version control systems like Git work well for source code, they are not designed to handle large data files efficiently. In the context of ML projects, we must generally work with more customized version control systems.

Git Large File Storage (Git LFS) is an extension for Git that addresses the challenge of working with large files in a Git repository. While Git is an excellent version control system for source code, it can become inefficient and slow when dealing with large files such as datasets, images, or model artifacts commonly used in machine learning projects. Git LFS solves this problem by replacing large files with lightweight text pointers within your Git repository while storing the actual file contents on a separate server. This keeps the Git repository lightweight and manageable, without sacrificing the ability to track and version large files.

Data Version Control (DVC) is a popular tool **specifically designed for versioning data and model artifacts in machine learning** projects. DVC works alongside Git, using a similar command structure and interface. Instead of storing the actual data files in the Git repository, DVC creates small metadata files that reference the data stored separately (e.g., local storage, cloud storage, or remote servers). This approach keeps the Git repository lightweight and manageable.

With DVC, you can track changes to your datasets and model artifacts, revert to previous versions, and share data with your teammates. DVC also supports data pipelines, making it easy to reproduce experiments and ensure consistency across the project.

A picture containing text, screenshot, font, diagram

Description automatically generated

Figure 5: Data Version Control with DVC[[4]](#footnote-5)

* **Model Artifacts Management:** Systematic storage and management of model artifacts, such as weights, checkpoints, and final trained models, is another key component of experiment tracking. Model artifacts including checkpoints and trained models should be kept organized between experiments. Version control software can be used for this, but it should be noted that large models require similar considerations to the version control of datasets – i.e. standard version control toolkits are not practical and specialized approaches such as DVC or GitLFS must be used instead.

**2. Experiment Parameters Documentation**

Documentation of model training is an intuitive component of experiment tracking. The need for recording the configuration of modelling approaches and the corresponding performance for each experiment is critical for reproducibility, as well as to understand and compare model performance. Parameters logging can be broken down into three separate domains:

* **Hyperparameter Logging:** documenting the hyperparameters used in each experiment, such as learning rate, batch size, optimizer, and model architecture. This helps in understanding the impact of these settings on the model's performance and aids in fine-tuning the model.
* **Performance Metrics Logging:** recording performance metrics (e.g., accuracy, precision, recall, F1-score, etc.) for each experiment to evaluate and compare different models or approaches. This allows practitioners to identify the best-performing models and investigate areas for improvement.
* **Experiment Metadata Logging:** documenting additional metadata related to the experiment, such as the date and time of the experiment, the researcher who conducted the experiment, and any relevant notes or observations.

When done in a systematized way, documentation of these aspects of the ML workflow can contribute in the following areas:

* **Organization:** logging experiment parameters helps you maintain a well-structured record of your experiments, making it easier to navigate through your work and understand the decisions you made at different stages of the project.
* **Comparability:** by logging parameters, you can easily compare different experiments and identify which hyperparameter settings led to the best performance. This information can be used to guide future experiments and fine-tune your models more effectively.
* **Reproducibility:** an important aspect of ML, particularly in research, is the ability to reproduce results. Logging parameters allows you, or anyone else, to recreate a specific experiment and verify the results, ensuring the integrity of your work.
* **Collaboration:** when working in a team, logging parameters ensures that everyone is aware of the configurations used in different experiments. This fosters knowledge sharing and enables team members to build upon each other's work more efficiently.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 6: Experiment parameters logged using wandb

**3. Visualization**

Visualization in experiment tracking can help us understand relationships between different model iterations on many different metrics. Visualizations can provide insights into the performance of models, identify areas for improvement, and facilitate collaboration among team members. Specialized libraries and platforms (e.g. wandb) can be very helpful in visualizing multiple model iterations on the same plot. Some common types of visualizations in experiment tracking include:

* **Performance curves:** performance curves depict the change in model performance (e.g., accuracy, loss) over time or epochs during the training process. Multiple iterations can be plotted together to compare issues overfitting, underfitting, or slow convergence, and assist in tuning hyperparameters for better performance.

A picture containing text, line, diagram, plot

Description automatically generated

Figure 7: Validation accuracy for multiple iterations (wandb). The tooltip shows parameter values for all iterations at epoch 31

* **Confusion matrices:** A confusion matrix illustrates the performance of a classifier by showing the true and predicted class labels. Multiple iterations can be plotted together to provide comparison of performance. Multiple classes can also be visualized in this way (like a pairs plot). However, this becomes less practical as the number of classes increase.

A screenshot of a graph

Description automatically generated with low confidence

Figure 8: Confusion matrix showing multiple model iterations (wandb). The tooltip shows the classifier’s true positive performance for a particular iteration.

* **ROC curves:** Receiver Operating Characteristic (ROC) curves are used to evaluate the performance of binary classification models at different classification thresholds. They provide a comprehensive view of model performance across various operating points and help in selecting the optimal threshold. Multiple iterations can be visualized on the same plot. However, this becomes less practical as the number of iterations and classes increase.

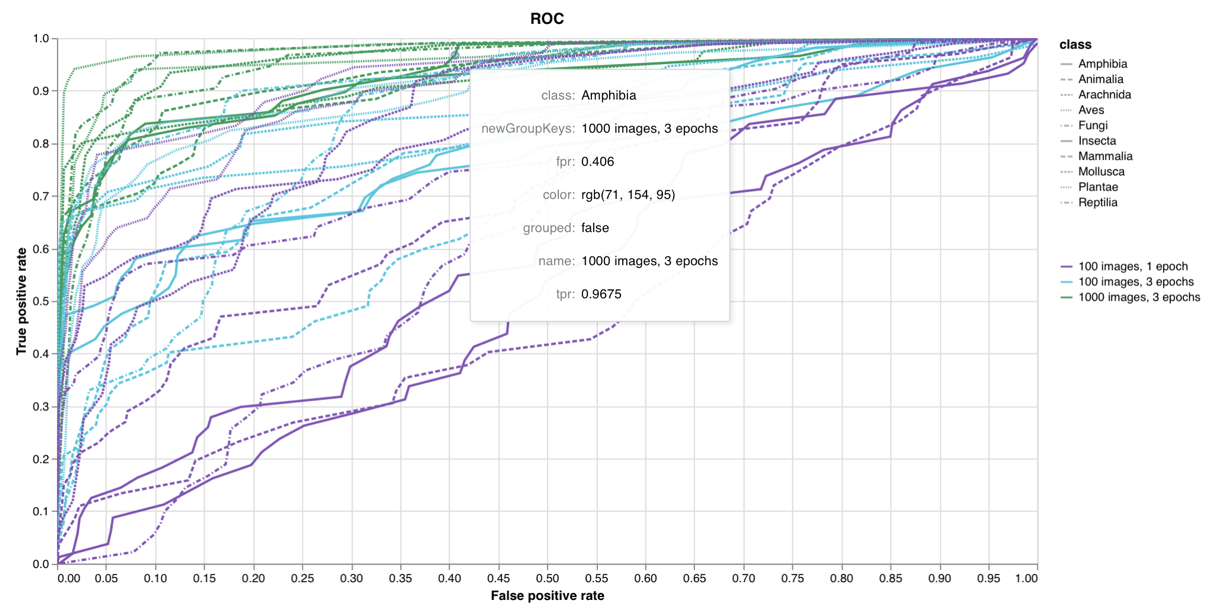


Figure : ROC Curve showing 3 model iterations of a multi-class classifer (wandb). The tooltip shows the ROC point data for a single iteration and class

* **Model parameter plots:** Visualizing the impact of different hyperparameter settings on model performance can provide insights into the most effective combinations and guide further hyperparameter tuning. Common hyperparameter plots include parallel coordinates, scatter plots, and line charts.

A picture containing line, plot, diagram

Description automatically generated

Figure 10: Multiple training iterations plotted against hyperparameters and output metrics (wandb). The tooltip shows parameter values for a single iteration

* **Parameter importance:** Similar to a feature importance plot for applicable models (e.g. random forests), visualizing the relative contributions of model inputs and hyperparameters can reveal which factors contribute most to a particular output metric.

A screenshot of a computer

Description automatically generated with low confidence

Figure 11: Model parameter importance in relation to accuracy (wandb)

Several experiment tracking tools, such as Weights & Biases (wandb), MLflow, and TensorBoard, offer built-in visualization features that make it easy to create, explore, and share visualizations in machine learning projects. These tools enable you to track, compare, and analyze multiple experiments simultaneously, providing insights into model performance, hyperparameter settings, and other aspects of your machine learning workflow. We discuss experiment tracking toolkits in more detail below. Ultimately, effective visualization in experiment tracking can lead to improved model performance, more efficient experimentation, and better collaboration among team members.

**Toolkits for Experiment Tracking**

Several popular toolkits are available to faciltiate experiment tracking in ML workflows. These toolkits enable tracking of model parameters, hyperparameters, performance metrics, artifacts, and more, facilitating efficient comparison and analysis of experiments. Similar to how git is deployed in Github and Gitlab, many experiment tracking toolkits are deployed as platforms offering free tiers for researchers and indivual practioners. Open source versions are available for self-hosted deployment. We provide here the following (non-exhaustive) list of common experiment tracking toolkits used in the ML field:

* [Weights & Biases](https://wandb.ai/site) (wandb): Wandb is a popular, lightweight, and user-friendly experiment tracking tool that enables logging and visualization of metrics, hyperparameters, and model artifacts. It offers a web-based dashboard for easy experiment management and comparison, as well as integrations with popular machine learning frameworks like TensorFlow, PyTorch, and Keras. The screenshots used in the previous section were taken from wandb, and the exercises in this module will focus exclusively on this tool.
* [TensorBoard](https://www.tensorflow.org/tensorboard): TensorBoard is a widely-used visualization tool for TensorFlow and PyTorch models, offering a rich set of features for tracking experiments. It provides real-time visualization of learning curves, histograms of weights and biases, model architecture diagrams, and other useful insights into model performance.
* [MLflow](https://mlflow.org/): MLflow is an open-source platform for managing end-to-end machine learning lifecycles. It provides components for experiment tracking, reproducibility, and model deployment. MLflow's experiment tracking functionality allows you to log parameters, metrics, and artifacts, as well as organize and compare experiments using its web-based UI or API.
* [Comet.ml](https://www.comet.com/site/): Comet.ml is a cloud-based platform for experiment tracking and collaboration in machine learning projects. It allows you to log and compare experiments, visualize model performance, and share results with team members. Comet.ml also supports integration with popular machine learning frameworks and tools like TensorFlow, PyTorch, and Jupyter Notebooks.
* [Neptune](https://neptune.ai/): Neptune is another cloud-based experiment tracking and collaboration platform. It provides features for logging and visualizing experiments, model artifacts, and performance metrics. Neptune offers integration with popular machine learning frameworks, version control systems like Git, and other tools like Jupyter Notebooks and Google Colab.
* [Data Version Control](https://dvc.org/) (DVC): DVC is an open-source tool for versioning data and model artifacts in machine learning projects. While primarily focused on data versioning, DVC also supports basic experiment tracking features, such as logging metrics and parameters, and can be integrated with other tools like TensorBoard or MLflow for more advanced tracking and visualization.

**Collaboration**

In manual experiment tracking methods, collaboration relevant to specifc components typically takes place ‘out-of-band’ – i.e. outside of the forum in which the work is being done. Specialized toolkits usually include collaboration features that allow team members to share, compare, and discuss experiments and model performance. Here are some common collaboration features found in popular experiment tracking toolkits:

* **Web-based dashboard:** Many experiment tracking tools, such as Weights & Biases (wandb), MLflow, Comet.ml, and Neptune, offer web-based dashboards that allow team members to access and visualize experiments, metrics, and artifacts from anywhere. These dashboards provide an intuitive interface to compare and analyze multiple experiments, fostering collaboration and knowledge sharing.
* **Sharing and permissions:** Cloud-based platforms like Comet.ml and Neptune allow users to share experiments, results, and visualizations with team members or stakeholders by providing shareable links or setting user access permissions. This functionality ensures that the right people have access to the relevant information, facilitating collaboration and decision-making.
* **Integration with version control systems:** Experiment tracking tools often integrate with popular version control systems like Git, enabling seamless collaboration on code, data, and model artifacts. Tools like DVC or MLflow leverage Git for versioning and collaboration, allowing team members to track, share, and collaborate on experiments and their associated artifacts through pull requests and issue tracking.
* **Comments and annotations:** Some experiment tracking platforms, such as Comet.ml and Neptune, provide features to add comments or annotations directly to experiments, metrics, or visualizations. This allows team members to discuss results, provide feedback, or highlight specific insights, promoting effective communication and collaboration.
* **Exporting and reporting:** Experiment tracking platforms often support exporting data, visualizations, or entire experiment summaries in various formats (e.g., CSV, JSON, or PDF). This makes it easier to share results and insights with stakeholders, create reports, or present findings to a wider audience.

# 7.3 Continuous Integration / Continuous Deployment with ML

Continuous Integration (CI) and Continuous Deployment (CD) are software development practices that help improve the efficiency and reliability of software projects by automating the build, testing, and deployment processes. In the context of ML, CI/CD can be adapted to support the unique requirements of ML workflows and facilitate collaboration, reproducibility, and model quality.

**Motivation**

As with experiment tracking methods, coverage of CI/CD in often limited in data science and machine learning academic courses. Data scientists might not encounter CI/CD until they enter industry. Here we give a brief overview of CI/CD in the context of ML.

**CI/CD for ML**

CI/CD is an approach that enables more frequent updates by automating various stages of the development process. In machine learning (ML), these stages differ from traditional software development, as a model relies on not only the code but also the data and hyperparameters. Additionally, deploying a model to production in ML is more intricate compared to standard software deployment.

The figure below demonstrates the CI/CD cycle. The cycle will be familiar to anyone who has worked in an agile or lean framework, as it uses a similar feedback mechanism to inform further iterations (releases).

A picture containing text, font, logo, graphics

Description automatically generated

Figure 12: The CI/CD cycle[[5]](#footnote-6)

**1. Continuous Integration (CI) for ML**

Continuous integration in ML entails that the ML pipeline is re-executed every time there is an update to the code or data. This process is carried out in a version-controlled and reproducible manner, enabling seamless sharing of the codebase across projects and teams. Each rerun may involve training, testing, or generating new reports, making it easier to compare different versions in production. Key aspects of CI in the ML context include:

* **Automate model training and validation:** CI can help automate the process of training and validating ML models when new data or code changes are introduced. This ensures that models are always up-to-date and helps identify potential issues early in the development process.
* **Version control for data, code, and models:** Integrating version control systems like Git and data versioning tools like DVC into the CI process helps track and manage changes to data, code, and model artifacts. This fosters collaboration, reproducibility, and traceability in ML projects.
* **Automated testing:** CI allows you to run automated tests on your ML code and models, including unit tests, integration tests, and model performance tests. This helps ensure code quality, model performance, and consistency across different experiments and team members.
* **Monitoring and alerting:** Implementing monitoring and alerting systems in the CI process can notify team members of potential issues, such as model performance degradation, data drift, or failed builds, allowing for quicker resolution and improved model quality.

**2. Continuous Deployment (CD) for ML:**

Continuous deployment is an approach that automates the deployment of new releases to production or other environments, such as staging. This practice facilitates receiving user feedback due to faster and more frequent updates, as well as providing new data for model retraining or the introduction of new models. Key aspects of CD in the ML context include:

* **Model deployment automation:** CD enables automated deployment of ML models to staging or production environments once they have passed the required validation and testing criteria. This helps maintain an up-to-date model serving infrastructure and reduces manual intervention.
* **Rollback and versioning:** CD practices should include rollback mechanisms and model versioning to ensure that you can easily revert to a previous version of a model if needed. This is especially important in ML, where model performance can be highly sensitive to changes in data or code.
* **Model monitoring in production:** Integrating model monitoring and performance tracking into the CD process helps ensure that models in production maintain their expected performance levels. Monitoring tools can track metrics like accuracy, latency, and resource utilization, and alert teams to potential issues or performance degradation.
* **Continuous improvement:** CD for ML enables an iterative approach to model development and deployment, allowing teams to continually refine their models and incorporate feedback from production systems, users, or new data. This promotes continuous improvement of model performance and overall project quality.

1. CNN tuning using the CIFAR10 dataset (<https://arxiv.org/abs/1810.05934>). This paper (and plot) present an approach to parallelization of hyperparameter tuning - but the principle communicated by the plot applies to single machine optimizaiton as well. [↑](#footnote-ref-2)
2. [Hyperparameter tuning using Grid Search and Random Search: A Conceptual Guide | by Jack Stalfort | Medium](https://medium.com/@jackstalfort/hyperparameter-tuning-using-grid-search-and-random-search-f8750a464b35) [↑](#footnote-ref-3)
3. https://guides.github.com/introduction/flow/ [↑](#footnote-ref-4)
4. https://dvc.org [↑](#footnote-ref-5)
5. [MLOps Guide (mlops-guide.github.io)](https://mlops-guide.github.io/) [↑](#footnote-ref-6)