Infrastructure document

By: Mark Goertz

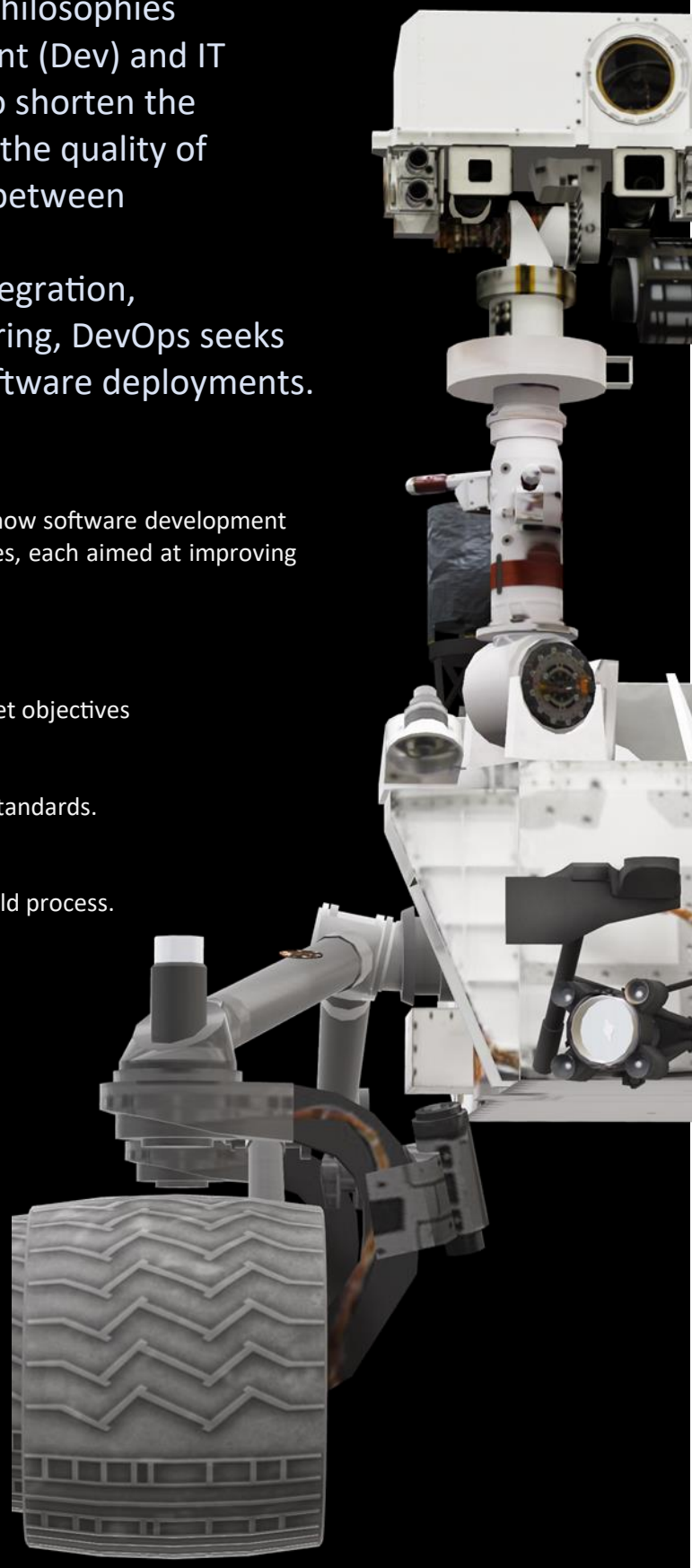
Master of Applied IT | Semester 2 (2024-2025)

OVERVIEW

This document provides a exploration of implementing and managing tools for machine learning development. Leveraging tools such as DVC (Data Version Control), CML (Continuous Machine Learning), and Docker, the document guides the reader through each stage of the infrastructure lifecycle, from analysis and advice to design, realization, and ongoing management.

What is DevOps

DevOps is a set of practices and cultural philosophies aimed at integrating software development (Dev) and IT operations (Ops). The goal of DevOps is to shorten the software development lifecycle, enhance the quality of software, and improve the collaboration between development and operations teams.
By promoting automation, continuous integration, continuous delivery, and efficient monitoring, DevOps seeks to improve the speed and reliability of software deployments.



DevOps Cycle

The DevOps cycle provides a framework for understanding how software development and operations are integrated. It encompasses several stages, each aimed at improving efficiency and quality:

1. **Plan**:
   Define project requirements, create project plans, and set objectives
2. **Code:**
   Write and commit code, ensuring adherence to coding standards.
3. **Build:**
   Compile code and build applications, automating the build process.
4. **Test:**
   Perform automated testing to ensure code quality and functionality.
5. **Release:**
   Deploy the application to staging or production environments.
6. **Deploy:**
   Automate deployment processes to ensure reliable and consistent releases.
7. **Operate:**
   Monitor application performance, manage infrastructure, and handle operational tasks.
8. **Monitor:**
   Continuously track application health, user feedback, and system metrics to identify and address

Difference between MLOps and DevOps

MLOps (Machine Learning operations) is a practice aimed at streamlining the process of developing, deploying, and maintaining machine learning (ML) models in production environments. It builds upon the well-established practice of DevOps (Development Operations), but it introduces additional layers of complexity to address the unique challenges faced in machine learning (ML) projects. MLOps integrates machine learning with the core principles of DevOps—automation, monitoring, and continuous deployment—while incorporating essential processes like data management, model versioning, and continuous training. [1]

MLOps Levels

CATEGORY	LEVEL 0	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4
STRATEGY	- No data scientists hired - Skeptical of value of ML among executive team	- Small and siloed data science and data engineering teams - A small number of ML champions in executive team	- Small Data science, data engineers and software development teams starting to be coordinated - ML development efforts still unstructured and discrete	- Large, well-integrated teams across data science, engineering and software development - Chief Data Officer, and C-suite level sponsorship exists - New team members brought up to speed in weeks - Project checkpoints to ensure ML is considered for major projects	- ML seen as strategic, driving company initiatives - Well-governed process for ML delivery - Engineers & researchers are embedded on the same team
ARCHITECTURE	- Data Silos with one-off integration - Data not prepared nor ready for ML	- Basic Enterprise data ready for ML - Data architecture still immature - Tacit commitment to meaningful enterprise data in the cloud.	- Data architecture is mature - Most enterprise data ready for ML in the cloud - Overt commitment to cloud	- Enterprise data is well cataloged and managed - Automated data pipelines in place - ML configuration and infrastructure is managed - ML models automatically provisioned as microservices	- Comprehensive architecture to effectively govern all data - Consistent data storage and consumption pipeline across projects - Target ML infrastructure monitored for cost-effectiveness and optimal utilization
MODELING	- Manual process for model training - Limited pilot studies	- Manual ML model training and live pilots - Basic experiment tracking, no model management	- Experiment tracking and model management in place - Models dependencies not well understood	- Models cataloged through lifecycle, supporting reproducibility and reuse - Output from ML is predictable and consistent, with auditable and reproducible outcomes	- Interdependencies between models are monitored and managed - Impact of small changes to ML models can be measured
PROCESSES	- No DevOps practices adopted - No clearly defined success criteria for ML projects	- DevOps practices like CI/CD have been adopted for non-ML components - No consistency in measures for ML or MLOps success	- Development iterative but CI/CD not in place for models - ML infrastructure expertise not broadly available - ML configuration is an afterthought - Metrics/ measures in place but not consistent across projects	- Data tested for model applicability and monitored for changes in distribution - All artifacts (data sets, tests, models) under version control - DevOps practices like CI/CD, code reviews in place for ML code - Production MLOps pipeline flow includes packaging, deployment, serving and operational monitoring	- Comprehensive MLOps pipeline supporting frequent model updates - New algorithmic approaches can be tested at full scale - Automatic metrics gathering, alerts, issues analysis (such as data drift) and automated retraining of systems is in place
GOVERNANCE	- Not considered	- Not considered, though the organization may have broader views - No notion of the concept of bias in models	- Model explainability not considered - Models may harbor prediction bias - Model releases are tracked	- Security policies applied to models, data - Ethics and explainability consideration for models and ML Systems - Good faith attempts to remove biased variables from models - Potential for malicious use of ML considered in ML lifecycle	- Cybersecurity experts engaged in ML operations - ML systems protected from external manipulation. - End to end audit trail for ML – who, why, when

Figure 1: MLOps Maturity model [2]

Analysis of Tools and challenges

This document explores the current challenges and inefficiencies in managing the development and experimentation of a machine learning project focused on stress classification using physiological data. The project involves analyzing signals from wearable devices, such as Electrodermal Activity (EDA), Blood Volume Pulse (BVP), skin temperature (TEMP), and accelerometry (ACC), to detect stress. As the project progresses and the complexity of data, models, and experiments increases, it is essential to address the infrastructural challenges that arise.

Context of the Project

The primary goal of the project is to develop an effective model capable of accurately detecting stress from physiological signals. The dataset used for training and evaluation is the WESAD dataset [3], collected from wearable sensors. Given the nature of the data and the research objectives, the project requires experimentation with different signal combinations, preprocessing techniques, and model configurations.

As the machine learning project grows, several infrastructural challenges have emerged that hinder the efficiency of the development process. These challenges need to be addressed to ensure that the project remains manageable, reproducible over time.

Current Challenges in Infrastructure

The current state of the infrastructure reflects a need for more automation, and the management of the workflows. The current development of the machine learning project rely heavily on manual processes for versioning and experiment tracking, leading to inefficiencies and increased risk of errors. Furthermore, the lack of centralized tools for handling data and model management makes it difficult to maintain consistency and track changes across the development cycle.

1. Automation

One limitation of the current infrastructure is the lack of automation in managing machine learning experiments. Many tasks, such as versioning datasets, tracking model configurations, and managing preprocessing steps, are done manually. This creates inefficiencies and increases the likelihood of human error, particularly when conducting multiple experiments. There is a clear need for automated pipelines to handle data and model versioning, track experiments, and present results to streamline the workflow.

2. Complexity

Machine learning projects often involve complex dependencies between data, models, preprocessing steps, and configurations. As new models are trained and evaluated, manually managing and tracing these dependencies becomes increasingly difficult. Additionally, keeping track of different versions of models, code, and data is crucial for ensuring consistency across experiments and organizing model configurations.

3. Experiment Reproducibility and Traceability

A requirement for any research-oriented project is the ability to reproduce experiments and trace changes over time. Given the iterative nature of machine learning experiments, the ability to reproduce previous results is fundamental for validating findings and ensuring the robustness of the model. The current system does not adequately support experiment tracking, making it difficult to review past experiments or to ensure that results can be consistently reproduced. For this project, where multiple physiological signals are used for stress detection, different combinations of these signals are tested to identify the most effective configuration.

4. Centralised Environment

Collaboration among different teams, such as data scientists, engineers, and domain experts, should remain optional for in the future of the project. However, the lack of a centralized system for managing datasets, models, and experimental configurations makes it difficult to collaborate effectively. Clear version control and proper experiment tracking are needed to ensure that all teams are aligned and working efficiently.

Available tools and platforms

This section outlines the tools and platforms available for managing machine learning workflows, experiments, and collaboration. These tools assist in automating tasks such as data versioning, model tracking, and experiment management, ensuring efficiency, reproducibility, and traceability. Platforms that support collaboration among teams, version control, and seamless integration of various machine learning components are key to streamlining the research and development process.

ML Model Registry

MLflow	Model registry, lifecycle management, stage transitions (staging/production), integration with experiment tracking.	[4]
Kubeflow	End-to-end ML platform with integrated model registry, Kubernetes-native, scalable model deployment.	[5]
Weights & Biases	Tracks and manages models alongside experiments, logs model metadata and performance metrics.	[6]
Neptune.ai	Model registry integrated with experiment tracking, version control, and comparison of model versions.	[7]
DVC Studio	Version control for models and data, integrates with Git for model registry, can track experiments.	[8]
ClearML	Platform for experiment tracking, dataset management, and orchestration of ML workflows.	[9]
Comet	Tool for experiment tracking, model optimization, and dataset exploration with integrations for CI/CD pipelines.	[10]

Data Version Control

Data Version Control	Git-like tool, version large datasets and model artifacts, stores data in external storage (e.g., S3, GCS, Azure, on-premises servers)	[8]
Pachyderm	Pachyderm is a data versioning and pipeline management tool designed specifically for data-centric machine learning workflows.	[11]
Git LFD	Git LFS is an extension for Git that allows you to manage large files, such as datasets and models, by storing them outside the Git repository but keeping them versioned.	[12]

Experiment Tracking

MLflow	Logs and tracks experiments, metrics, hyperparameters, and artifacts. Includes model registry capabilities.	[4]
Weights & Biases	Detailed experiment logging, tracks hyperparameters, performance metrics, collaborative visualizations, model tracking.	[6]
TensorBoard	Visualization toolkit for training metrics, model graph, compatible with TensorFlow and other ML frameworks.	[13]
Neptune.ai	Experiment tracking with model and dataset versioning, hyperparameter optimization, and collaboration tools.	[7]
DVC	Tracks experiments, hyperparameters, and metrics, integrates with Git and supports reproducibility.	[8]

Multi-criteria Table of tools and platforms

This table provides a structured overview for comparing and evaluating different options based on multiple criteria. Each criterion represents a key aspect of assessment, allowing for a comparison analysis. The table can be used to identify trade-offs across the options, helping to inform decision-making.

Feature/Criteria	MLFlow	Weight & Biases	DVC	Comet	Neptune	TensorBoard	ClearML
Experiment Tracking	✔	✔	✔	✔	✔	✔	✔
Model Versioning	✔	✔	✔	✔	✔	✘	✔
Dataset versioning	✘	✘	✔	✘	✘	✘	✔
Hyperparameter Tracking	✔	✔	✔	✔	✔	✔	✔
Artifact Storage	✔	✔	✔	✔	✔	✔	✔
Collaboration Features	✔	✔	✔	✔	✔	✘	✔
Visualization Tools	Basic	Advanced	Basic	Advanced	Advanced	Basic	Advanced
Integration with MLOps	✔	✔	✔	✔	✔	✘	✔
Ease of Use	Medium	High	Easy	High	High	Medium	High
Cloud Support	✔	✔	✔	✔	✔	✘	✔
Open Source	✔	✘	✔	✘	✘	✔	✔
Cost	Free/Open Source	Free trial & paid	Free/Open Source	Paid	Free trial & paid	Free	Free & paid

DVC was chosen over alternatives because DVC offers a more comprehensive, integrated solution for managing data, models, and experiments. While others are possibly good as well, DVC provides the added benefit of **native data versioning** alongside experiment tracking, making it easier to maintain a full end-to-end machine learning workflow. DVC's integration with Git allows us to handle both code and data versions seamlessly, reducing complexity and improving collaboration across teams.

Data Version Control & Studio

DVC (Data Version Control) is an open-source version control system designed for machine learning projects that deal with large datasets and model artifacts. It extends Git to handle versioning for not just code but also data, models, and experiments, ensuring consistency and reproducibility in machine learning workflows. By allowing the tracking of datasets, model binaries, and intermediate files without committing them to Git directly, DVC helps machine learning teams keep track of everything needed for a project.

Key Features of DVC:

- **Data Version Control:** DVC allows datasets and large files to be versioned alongside code in a Git repository. However, instead of storing these large files directly in Git (which can cause performance issues), DVC stores them in external storage such as cloud services (e.g., AWS S3, Google Cloud Storage, Azure Blob Storage) or local systems. Git stores pointers (small metadata files) that link to these large files. This keeps the Git repository lightweight and fast while still ensuring that each dataset version is properly tracked.
- **Model versioning:** DVC treats machine learning models as first-class entities, tracking each model version along with its associated code, datasets, and parameters. By using DVC, you can ensure that all the components that contributed to a model (e.g., data used for training, code that defines the model, and hyperparameters) are kept together in a versioned manner, making it easy to track changes, revert to earlier versions, and understand model evolution over time.
- **Pipeline management:** DVC allows you to define machine learning pipelines—sequences of data preprocessing, model training, evaluation, and other processes—using simple YAML files. DVC ensures that these pipelines are reproducible by storing the exact configuration and dependencies of each step. When a step or dependency changes (for instance, a new dataset or code update), DVC can automatically rerun the affected steps, ensuring consistency and reproducibility in the experiment.
- **Experiment Management:** One of DVC's most powerful features is its ability to track experiments. It logs hyperparameters, datasets, and metrics associated with each experiment, allowing data scientists to easily compare the outcomes of different experiments. With DVC, you can roll back to a previous experiment version if needed, facilitating experimentation and iteration in a way that's traceable and systematic.
- **Storage Agnostic:** DVC is not tied to any specific storage system. It can integrate with a variety of cloud storage platforms (e.g., AWS S3, Google Cloud Storage, Azure Blob Storage) as well as on-premises systems or local storage. This flexibility allows teams to manage their datasets and models efficiently, without worrying about storage limitations or vendor lock-in.

DVC in the contexts of the development cycle.

In MLOps, tools like DVC are critical for handling the challenges associated with data versioning and model management. Traditional DevOps tools typically handle code and application deployment but are not built for the intricacies of machine learning projects, which require managing large and evolving datasets. DVC addresses these needs by:

- **Data Version Control:** In machine learning, the data used for training and testing is dynamic and evolves over time. DVC ensures that each dataset version is properly tracked, which guarantees that models are trained and tested on consistent data. This versioning prevents issues where models are inadvertently trained on incorrect or outdated data, making the results more reliable and easier to reproduce.
- **Experiment Management:** DVC provides robust tracking of experiment metadata, such as the specific datasets used, hyperparameters, and performance metrics. This allows teams to quickly reproduce experiments by referencing a specific dataset version, code commit, and hyperparameter configuration. Furthermore, it helps in organizing and comparing results from multiple experiments, making it easier to identify the best-performing model.

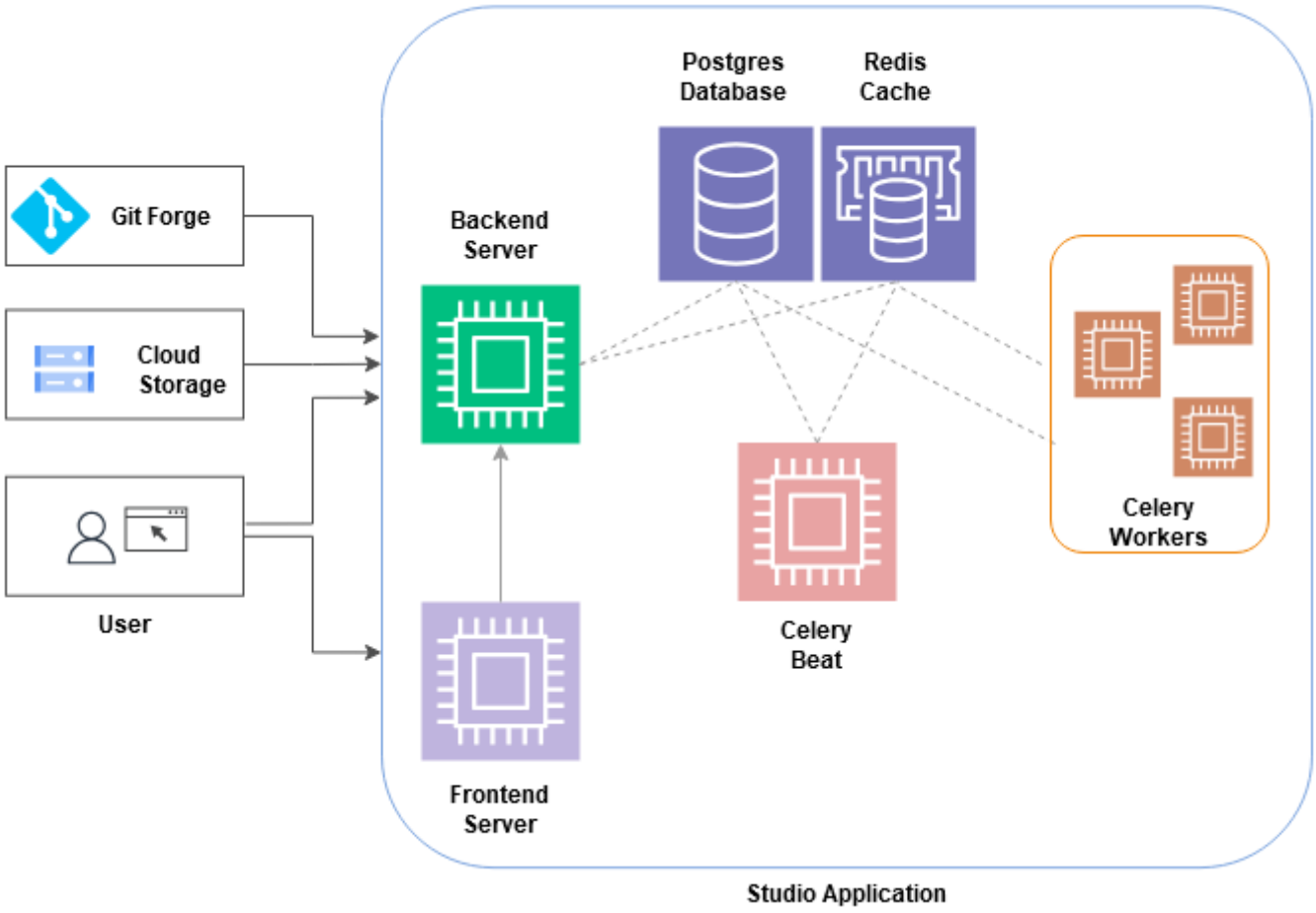
- **Integration with CI/CD Pipelines:** DVC integrates seamlessly with continuous integration/continuous deployment (CI/CD) systems. This integration ensures that as changes are made to the codebase (e.g., a new model or updated preprocessing step), the data and model artifacts are appropriately versioned, tested, and deployed in an automated, repeatable fashion. This streamlines collaboration between data scientists and engineers, ensuring smoother transitions from development to production.

DVC Studio for Enterprise

DVC Studio provides a web-based interface for managing and visualizing machine learning projects, making it easier for teams to collaborate on experiments and track changes in datasets, models, and results. For enterprise users, DVC Studio offers the option to host the platform on their own infrastructure, either on-premises or in their cloud environment. This flexibility ensures that organizations can maintain full control over their data and model management processes, while still leveraging the powerful features of DVC Studio for tracking and collaboration. [14]

With DVC Studio, enterprise users can easily:

- **Visualize experiment** results and metrics over time.
- **Compare different versions** of models and datasets.
- **Collaborate efficiently** by sharing experiment configurations and results.



Privacy & Security Compliances

When handling medical data in the Netherlands, especially when using tools like **DVC** for data versioning and **Azure** for cloud storage, it is essential to comply with **GDPR (General Data Protection Regulation)** and **Dutch-specific laws**. This is critical not only to protect individuals' privacy but also to ensure that data is securely stored, processed, and transmitted according to European and Dutch standards.

General Data Protection Regulations (GDPR)

1. *Lawfulness, Fairness, and Transparency:*

Lawful Basis for Processing

Under GDPR Article 5, data processing must be based on one of the lawful grounds specified by the regulation. In this case, since we are using publicly available medical datasets for research purposes, the public interest or scientific research grounds can be a lawful basis (Article 5(1)(e) and Article 9(2)(j)).

The datasets, WESAD[3] will be used for **scientific research** to analyze and predict stress from physiological data. This research is of public interest, contributing to the understanding of human physiology, and aligns with the lawful processing grounds outlined in the GDPR. Consent from individuals is required as the data has been made publicly available for research purposes, but it is important that the data was properly anonymized and processed for research use.

Because the WESAD dataset has been made publicly available for research purposes and is likely anonymized, **we do not need to seek additional consent from participants**. The participants have already been informed and consented to their data being used for research.

However, to use the dataset, we must adhere to its terms of use, which state:

"You may use this data for scientific, non-commercial purposes, provided that you give credit to the owners when publishing any work based on this data."

Therefore, our paper must include proper credit to the dataset's owners.

Fairness and Transparency

Transparency is a fundamental GDPR requirement. Even if the data is publicly available, the individuals whose data is used must have been informed about how their data will be processed.

The original datasets (WESAD[3]) were made publicly available under the premise of research use, and individuals whose data was collected were informed through their participation. In our research, we will provide transparency by explaining the dataset's purpose in the paper, the research it supports, and how the data will be processed.

DVC and Azure can contribute to logging, since DVC and Azure can show logs on how data has been processed and accessed throughout the workflow.

2. *Purpose Limitation:*

Personal data must be collected for specified, legitimate purposes and not further processed in a manner that is incompatible with those purposes.

We define and document the **research objectives** clearly (the detection of stress levels using physiological signals), ensuring that data is only processed for this purpose. We utilize **DVC's version control** features to ensure that datasets are tracked and stored for specific research purposes (for pre-processing techniques such as applying different Normalizations techniques).

Azure Blob Storage allows us to set **retention policies** for data storage, ensuring that data is kept only for as long as necessary to achieve the research goals. Expired or irrelevant data are deleted after 21 days. (GDPR Article 5(1)(b) & GDPR Article 89 (1))

3. *Data Minimization:*

Personal data must be adequate, relevant, and limited to what is necessary for the purposes for which it is processed.

Using **DVC**, we ensure that only the necessary datasets required for model training are tracked and stored. This minimizes the processing of unnecessary data. Additionally, DVC's version control prevents the inclusion of irrelevant or extraneous datasets in the workflow, ensuring that only data relevant to the research objectives is stored and used.

Azure Blob Storage provides scalable storage with features that support data filtering, allowing for the selective retention of only the most relevant datasets needed for training and evaluation.
(GDPR Article 5(1)(c))

4. *Accuracy:*

Under the GDPR, Article 5(1)(d) It requires that personal data must be accurate, maintained up to date, and corrected without any delay when inaccuracies are identified. In this context, the use of publicly available datasets, such as **WESAD** and **Affective Road**, for the development of a **1D CNN** model necessitates a focus on ensuring the data used is accurate and appropriately processed to avoid errors in the model's predictions.

Verification of Data Quality
While the datasets utilized are publicly available, it is essential to verify their quality before use. This involves checking for missing values, outliers, and inconsistencies that could negatively impact the accuracy of the model. Data preprocessing steps, such as normalization or standardization, could be applied to ensure the dataset is consistent and suitable for use in model training. The principle of accuracy requires that data used for research purposes must be thoroughly checked for integrity Article 5(1)(d). DVC assists in tracking changes, ensuring that the dataset in use is the most recent and appropriate version for the research project. This also aligns with Article 5(1)(e), which states that personal data should be accurate and kept up to date.

Preprocessing and Preparation
The accuracy of a model depends significantly on how the data is preprocessed. Data used for training a **1D CNN** must undergo careful preparation to ensure it is clean and in the correct format. This includes inspecting the data for any errors or misalignments and addressing them before the data is input into the model. Given the nature of the datasets, which involve physiological signals, it is particularly important to remove noise or distortion that could degrade the model's accuracy.

Use of Version Control to Track Modifications:
While the original dataset may not be altered directly, data preprocessing (such as normalization) involves modification. **DVC** ensures that all changes to the dataset are documented and versioned, allowing for full traceability of data transformations. Article 5(2) of the GDPR mandates that organizations must maintain transparent records of data processing activities, which includes tracking any modifications to the dataset.

Compliance with GDPR Principles

The GDPR Article 5(1)(d) principle of accuracy applies to all data processing activities, including the use of publicly available datasets for model training. Ensuring that the data used for research is accurate and well-processed, while maintaining transparency, is a critical step toward GDPR compliance.

The use of tools like DVC for version control, alongside careful documentation of all data handling practices, ensures full accountability and traceability of any data modifications, in line with Article 5(2), which emphasizes the accountability principle of the GDPR.

5. *Storage Limitation:*

Under GDPR's **storage limitation principle** (**Article 5(1)(e)**), data must only be retained for as long as necessary for the purpose it was collected. Once the data is no longer required, it must be securely deleted or anonymized to prevent unauthorized access or misuse.

Azure provides robust tools to operationalize storage limitation through **Lifecycle Management Policies**. These policies enable automated data transitions and deletions, ensuring compliance without manual intervention. [15]:

- **Set Retention Periods**: Specify durations for which raw, processed, or anonymized data should be retained, such as retaining raw data for a research project for one year and anonymized data for five years. Both Affective Road and WESAD datasets are anonymized and publicly available datasets.
- **Automated Data Deletion**: Azure can automatically transition inactive data to lower-cost storage tiers and delete it when the retention period expires. This supports GDPR compliance by preventing the storage of data beyond its necessary use case.

To maintain accountability and transparency, Azure provides comprehensive **audit trails** through tools like **Azure Monitor** and **DVC Logs**. These systems document key information about deletion activities, including timestamps, authorization details, and the specific deletion methods used. For example, when a dataset is flagged for deletion by a researcher, the process is fully logged to demonstrate compliance with GDPR's accountability principle.

This will require some set-up before everything is tracked and transparent, but once configured, the system will ensure that data retention, transition, and deletion processes are fully automated, auditable, and compliant with GDPR's storage limitation requirements. Regular reviews of the audit logs and lifecycle management policies will help maintain ongoing compliance, ensuring that data is only retained as long as necessary and securely deleted when no longer needed.

Relevant GDPR Articles:

1. **Article 5(1)(e)**: Personal data must be kept for no longer than necessary.
2. **Article 32(1)(d)**: Organizations must regularly evaluate processes to securely dispose of unnecessary data.
3. **Article 5(2):** The controller shall be responsible for, and be able to demonstrate compliance with, paragraph 1 ('accountability').

6. *Integrity and Confidentiality:*

GDPR requires that personal data be safeguarded through appropriate technical and organizational measures to ensure its integrity (i.e., data is accurate and protected from unauthorized alteration) and confidentiality (i.e., data is accessible only to authorized individuals).

Data Encryption:

Data must be encrypted both in transit and at rest to prevent unauthorized access. Azure provides robust encryption mechanisms, including **Azure Storage Service Encryption (SSE),** which encrypts data automatically as it is written to storage (Article 32(1)(a)).

The encryption is transparent to users and does not require any additional configuration or action. Data is encrypted using strong encryption algorithms (currently **AES-256**, which complies with the FIPS 140-2), and encryption occurs without affecting the performance of the storage. SSE ensures that data is encrypted at rest, meaning that the data is protected when it is stored in Azure and remains encrypted when it is not actively being used or transferred. By default, Azure uses Microsoft-managed keys to encrypt the data. [16], which supports GDPR's requirement for appropriate security (Article 32(1)(b), Article 5(1)(f), Article 32)

However, users also have the option to manage their own encryption keys through **Azure Key Vault**. This gives organizations greater control over their encryption keys if they require more severe key management policies (e.g., for compliance with certain standards like GDPR, HIPAA, or other security regulations). [17] Additionally, data transferred between services or to users is encrypted using **SSL/TLS protocol**s. This ensures that even if data is intercepted, it cannot be read or altered by unauthorized parties. [18]

Access Control and Role-based Access:

Active directory (AD) controls who can access and is authorized for the data. **Role-based access controls (RBAC)** can be enforced to ensure that only authorized personnel can access specific datasets. For example, only researchers directly involved with the analysis of stress data should have access to the datasets, while others may only access anonymized or aggregated data. This meets GDPR's requirement to limit access to personal data (Article 32).

Audit Trails and Logging:

DVC and Azure provide detailed logging and audit trails that track who accessed or modified the data and when. This allows for comprehensive accountability in the research process. These logs can be reviewed periodically to ensure that only authorized personnel are accessing sensitive medical data, and that the data is being used in compliance with established protocols. If there are any deviations or unusual activities, these logs can provide insight into potential breaches or unauthorized access attempts. Azure provides detailed logging and monitoring features, such as **Azure Monitor** and **Azure Activity Logs**, allowing you to track who accessed or modified data and when. This ensures accountability and supports compliance.

Data Integrity Checks:

Regular integrity checks should be conducted to ensure that data has not been tampered with or corrupted. For example, DVC tracks changes to datasets, ensuring that all modifications to data are documented and that we can trace any updates or changes made to the data. This provides an additional layer of accountability and helps ensure that the data remains consistent and intact throughout the research process.

Additionally, Azure supports lifecycle management policies for data stored in the account, helping to enforce GDPR's **storage limitation** principle (Article 5(1)(e)).

HIPAA (Health Insurance Portability and Accountability Act for US applications):

HIPAA is a United States federal law that mandates the protection of sensitive patient information. It sets standards for how medical data should be handled, stored, and transmitted to ensure privacy and security. Specifically, HIPAA applies to healthcare providers, insurers, and business associates who deal with Protected Health Information (PHI).

NEN 7510 (Dutch applications)

NEN 7510 is the Dutch standard for information security in healthcare. It outlines requirements for protecting healthcare data and systems against threats, ensuring the confidentiality, integrity, and availability of data. This standard is specifically designed for healthcare organizations and addresses issues such as data encryption, access control, and risk management. Compliance with NEN 7510 is essential for ensuring that healthcare data is securely handled, aligning with Dutch regulations on data protection.

Dutch Data Protection Authority (AP)

The Dutch Data Protection Authority (Autoriteit Persoonsgegevens, or AP) is responsible for enforcing privacy laws and regulations in the Netherlands, including the GDPR. The AP monitors organizations' compliance with privacy regulations and can impose fines or sanctions on entities that fail to protect personal data properly. If handling personal data in the Netherlands, especially in the healthcare sector, it is crucial to ensure compliance with the AP's regulations to avoid legal consequences.

SOC 2 Type I and Type II Reports:

SOC 2 (System and Organization Controls 2) reports are assessments of how well an organization manages data based on five key principles: security, availability, processing integrity, confidentiality, and privacy.

- **SOC 2 Type I Report**: This evaluates the suitability of the design of an organization's controls at a specific point in time.
- **SOC 2 Type II Report**: This evaluates the operational effectiveness of the organization's controls over a period, usually 6 to 12 months. These reports are crucial for ensuring that the organization complies with regulatory requirements regarding data protection and privacy, and they are often used by businesses to demonstrate their commitment to security and privacy standards.

DVC has also complied a SOC-report:

https://static.iterative.ai/iterative-inc-soc2-type1-report.pdf
https://static.iterative.ai/iterative-inc-soc2-type2-report.pdf

eHealth Regulations (for EU applications)

eHealth regulations encompass a variety of laws and guidelines governing the use of technology in healthcare. These regulations address issues such as data protection, patient consent, and the use of electronic health records (EHR). They aim to ensure that health information is securely handled, while promoting the safe and effective use of technology in healthcare delivery. These regulations often align with broader privacy laws like GDPR, but are tailored to address the unique needs and challenges of the healthcare sector.

Each of these regulations or standards is essential for ensuring that health data is processed securely, keeping patient privacy at the forefront of healthcare technology and research.

Summary of Analysis

To address the current inefficiencies in machine learning project management, I will use DVC (Data Version Control) for data versioning, model registry, and experiment tracking. DVC provides an automated solution for managing large datasets, model versions, and tracking machine learning experiments. It integrates with Git to ensure consistency across code, data, and models while supporting reproducibility. Additionally, DVC Studio will be used for centralized experiment management and collaboration. This setup will improve scalability, automation, and version control, addressing the challenges in the current infrastructure.

To set-up everything properly, the following requirements have been set-up:

Functional Requirements

A. DATA VERSION CONTROL

Requirement	Data Versioning & Experiment Tracking
Description	Manage versions of datasets, models, and experiment metadata.
Implementation	Use **DVC** for dataset and model versioning. Integrate **DVC Studio** for experiment tracking.
Acceptance Criteria	Datasets and models are versioned with DVC. All experiment parameters and metrics are logged and traceable.

B. MAINTAINABILITY & REPRODUCIBILITY

Requirement	Maintainability & Reproducibility
Description	Ensure all experiments and models are reproducible and maintainable.
Implementation	Store pipeline scripts and configurations in **Git**. Version control all datasets, models, and experiments with **DVC**.
Acceptance Criteria	Experiments and models can be reproduced with the same configurations and versions. Code and data updates do not break backward compatibility.

C. MODEL MANAGEMENT

Requirement	Model management
Description	Ensure models are stored, versioned, and managed efficiently.
Implementation	Use **DVC** for model storage and versioning.
Acceptance Criteria	Models are tracked, and versioned.

Non-Functional Requirements

A. DATA ENCRYPTION

Requirement	Data encryption
Description	Protect sensitive data with strong encryption mechanisms.
Implementation	- Use **TLS 1.2** or higher for all data transmission over the network. - Encrypt data at rest using **Azure Storage Service Encryption (SSE)** and **Azure Key Vault** for managing encryption keys.
Acceptance Criteria	Datasets and models are versioned with DVC. All experiment parameters and metrics are logged and traceable.

B. MAINTAINABILITY & REPRODUCIBILITY

Requirement	Access Control (RBAC)
Description	Ensure only authorized users and systems can access sensitive data and models.
Implementation	- Implement **Role-Based Access Control (RBAC)** for DVC to restrict access based on user roles.
Acceptance Criteria	Access to data is strictly controlled based on predefined roles. Only authorized personnel can access, modify, or delete data and models.

C. SECURE DATA STORAGE & RETENTION

Requirement	Secure Data Storage & Retention
Description	Ensure compliance with data retention policies and securely manage data throughout its lifecycle.
Implementation	- Use **Azure Blob Storage** with configurable **data retention policies** to automatically delete outdated or irrelevant data after a set period.

Acceptance Criteria	Data retention policies are strictly followed, and expired or irrelevant data is automatically deleted.

D. SECURE AUTHENTICATION & AUTHORIZATION

Requirement	Secure Authentication & Authorization
Description	Ensure that all systems and services are securely authenticated and authorized.
Implementation	Implement **multi-factor authentication (MFA)** for access to sensitive systems.
Acceptance Criteria	Systems and services are authenticated securely, and unauthorized access is blocked.

Proposed Architectural Design of development cycle.

This phase outlines the infrastructure architecture, tool integration, and security measures necessary for building a scalable and efficient machine learning pipeline. The architecture leverages integrating tools such as **DVC**, **DVC Studio**, **and Docker**, to ensure automation, reproducibility, and security across the development lifecycle.

Version control Architecture Design

A core component of the design is using **DVC (Data Version Control)** for tracking datasets and model artifacts throughout the machine learning lifecycle. DVC enables versioning of datasets, models, and configurations, ensuring a comprehensive history of changes, facilitating reproducibility, and enabling rollback to previous states when necessary.

DVC uses .dvc files to track metadata about datasets, such as their location and version history, without storing large data files in the Git repository itself. This avoids bloating the repository with large files. DVC stores datasets in external storage solutions (e.g., AWS S3, Google Cloud Storage, or local storage) while keeping lightweight references in the Git repository. Data changes are stored efficiently using checksum-based deduplication, minimizing redundant storage.

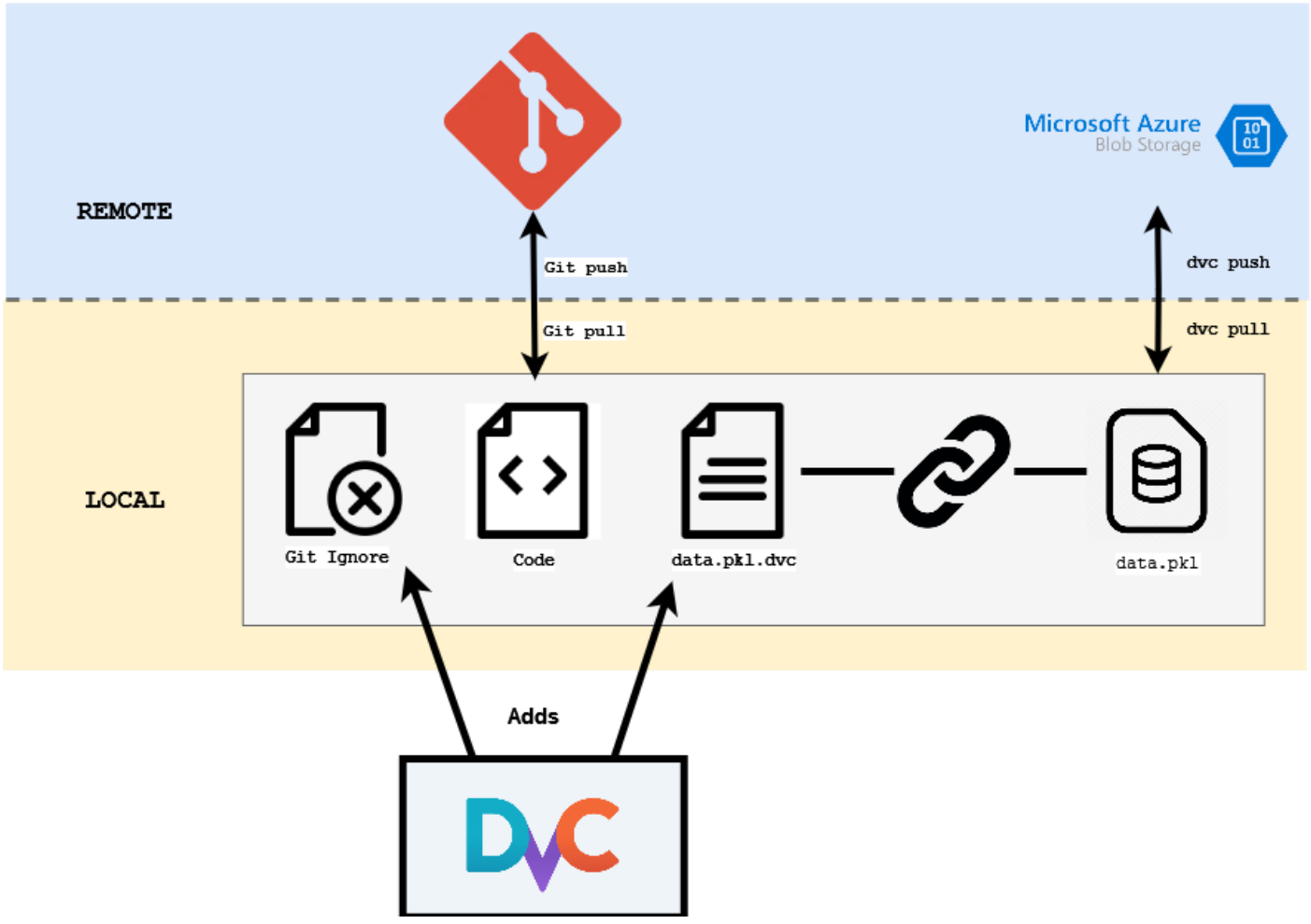


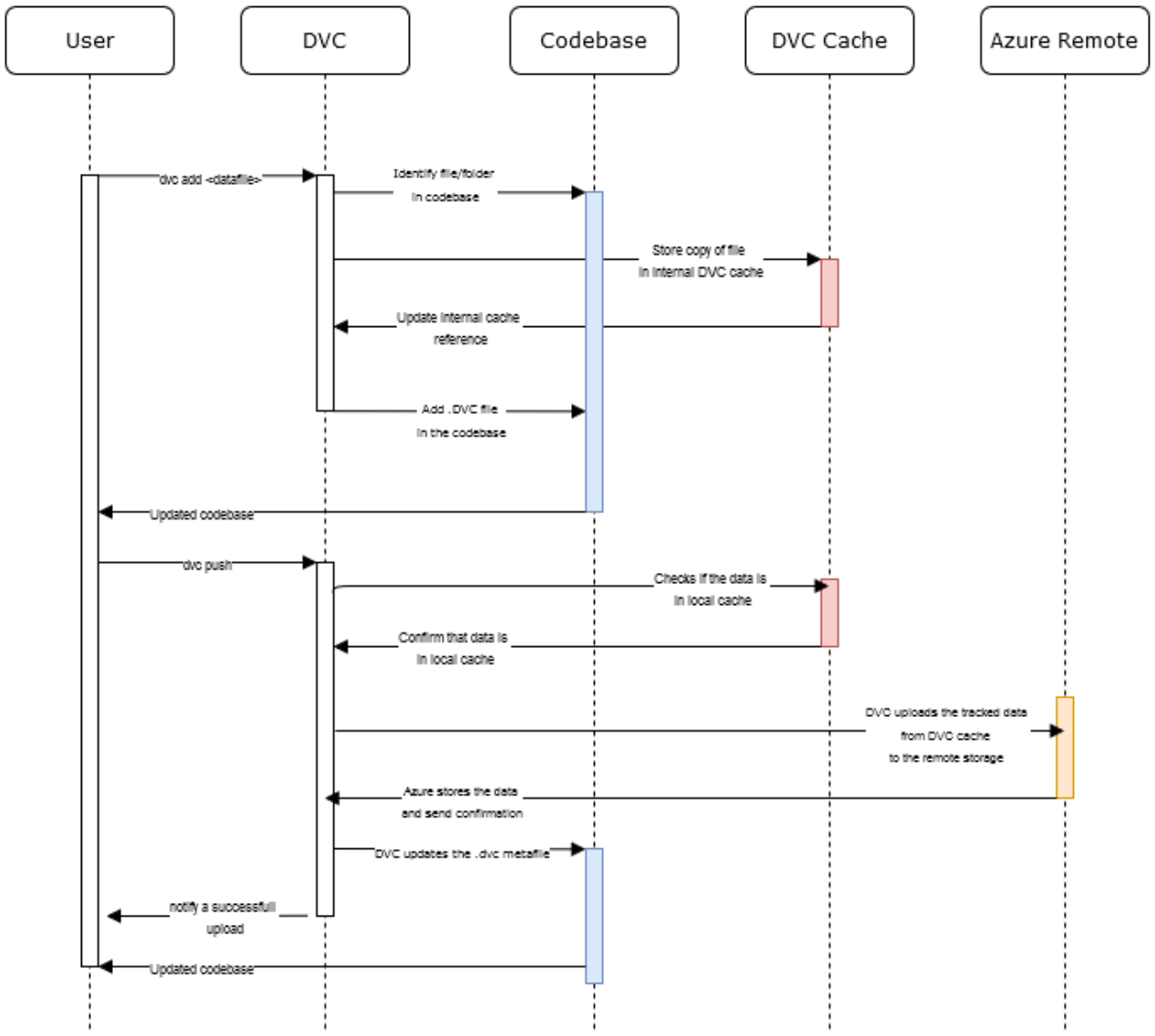
Figure 2: Diagram to visualize the workflow of Data Version Control

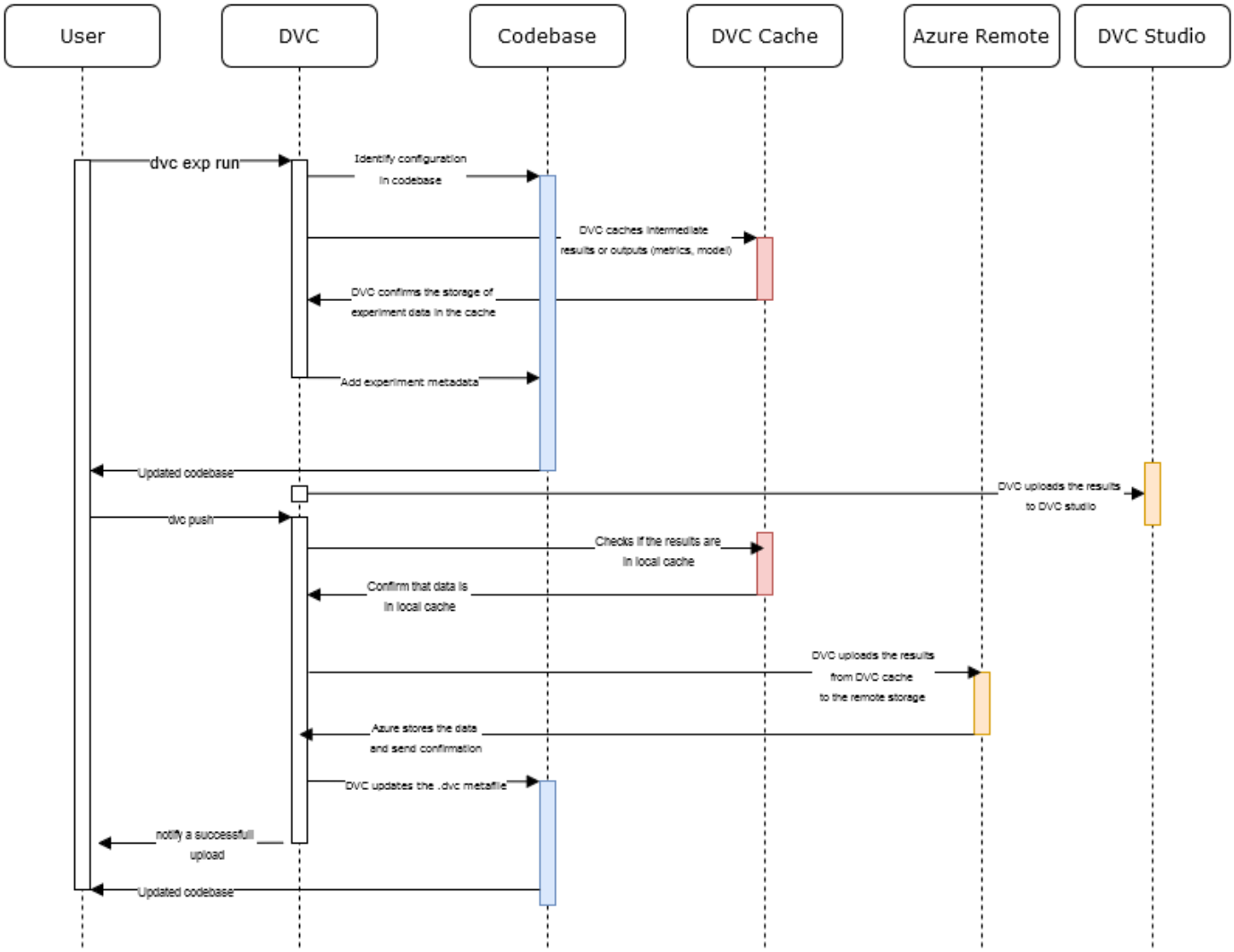
Storing Data under Data Version Control

The sequence diagram illustrates the workflow of using DVC (Data Version Control) to track data and experiments, store them in remote storage (Azure), and interact with the **Codebase** and **DVC Cache**.

1. **User Interaction**: The **User** initiates commands to add, commit, push, or pull data and experiment results from the **Codebase**. The **Codebase** represents the project's files, including code, configuration, and data that are tracked by DVC.
2. **DVC Tracking and Caching**: When a user adds a file (dvc add), DVC calculates the file's hash and stores it in its local **DVC Cache**. This cache helps prevent redundant uploads and downloads by storing data locally, enabling it to be easier to track versions of datasets, models, and outputs.
3. **Data Upload/Download**: When the user pushes or pulls data (dvc push), DVC interacts with Azure Blob Storage as the remote storage backend. Azure stores the actual data, while DVC updates the **Codebase** with metadata pointers to the remote storage.
4. **Experiment Tracking**: DVC also tracks experiments, including changes in configurations and metrics. The results of these experiments are stored in the **DVC Cache** and uploaded to Azure, allowing the user to compare and retrieve them later.

This process provides version control for both data and experiments, ensuring that users can easily manage, reproduce, and collaborate on machine learning projects and data-driven workflows.




User
DVC
Codebase
DVC Cache
Azure Remote
DVC Studio
dvc exp run
Identify configuration in codebase
DVC caches intermediate results or outputs (metrics, model)
DVC confirms the storage of experiment data in the cache
Add experiment metadata
Updated codebase
DVC uploads the results to DVC studio
dvc push
Checks if the results are in local cache
Confirm that data is in local cache
DVC uploads the results from DVC cache to the remote storage
Azure stores the data and send confirmation
DVC updates the .dvc metafile
notify a successfull upload
Updated codebase

CI for ML Workflows: Detailed Explanation

The proposed **CI/CD pipeline** (Figure 3: CML CI-pipeline workflow) is designed to automate model training, testing, and deployment. The following key processes are integral to the CI/CD architecture:

1. **Versioning and Collaboration**

- **DVC Integration with Git**: DVC tracks datasets and model artifacts, while Git is responsible for versioning code. Git functions as the source of truth for code changes, while DVC enables data and model versioning. Changes are synchronized with remote repositories like **GitHub**.

2. **Data and Model management**

- Datasets and models are versioned and stored in external storage (e.g., **Azure Blob Storage**). DVC handles the interaction with this storage, ensuring efficient data management without overloading the Git repository.

3. **Automated Testing and Experimentation:**

- **CI Pipelines (via CML)**: Upon changes to the main branch, **CML** (Continuous Machine Learning, CI-part of DVC) triggers automated pipelines that execute model training and testing. Models are evaluated using pre-defined metrics, with results automatically logged and visualized through **DVC Studio**.

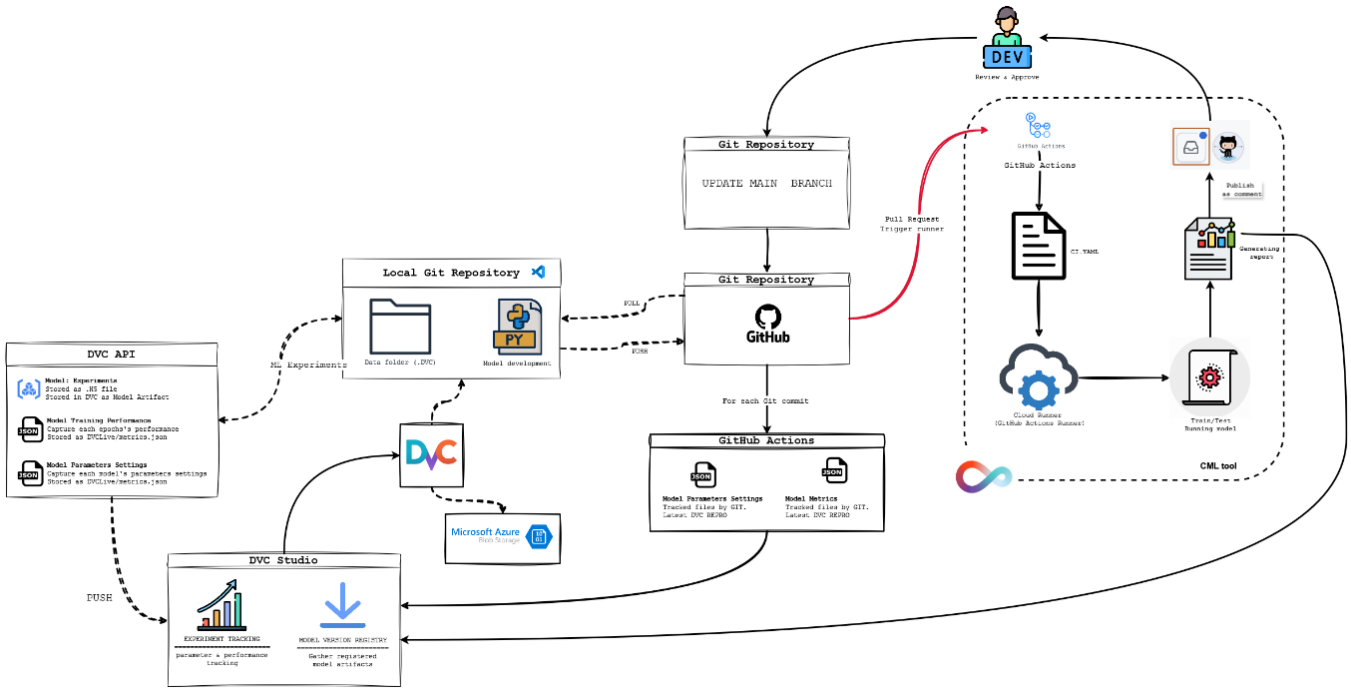


Figure 3: CML CI-pipeline workflow

Key Steps in the CI-Workflow:

1. Git Repository (Local and Remote)

- **Local Git Repository**:
  Developers store code (model definitions, training scripts, configurations) and use Git for versioning. This enables collaboration by isolating features or models in separate branches for independent development
- **Remote Git Repository (e.g., GitHub)**:
  Hosted on platforms like **GitHub** or **GitLab**, the remote repository serves as the central codebase, where teams push changes and manage pull requests (PRs) for merging updates to the main branch.

2. DVC (Data Version Control) Integration

DVC Integration with Git:

- DVC is integrated with Git to manage large datasets and model files that are versioned alongside the code. Instead of storing large data files in Git, DVC stores pointers to the data in the repository while managing the actual data in external storage (like cloud storage or on-premises servers).
  The diagram indicates that DVC tracks datasets and models locally, and the metadata is version-controlled in Git. This allows reproducibility of experiments and sharing of data among team members.

DVC Studio:

- DVC Studio is a web-based interface that helps visualize data pipelines, models, and experiments. It allows for collaboration and experiment tracking in a centralized platform, helping ML teams stay aligned and keep track of experiment results.

3. CI/CD Pipeline for ML (with DVC and Git)

Automation with CI/CD:

- In an ML pipeline, CI/CD tools (GitHub Actions) automate the testing, training, and deployment of models. The workflow begins with a code change (such as a new model or training script) pushed to the Git repository.

CI Integration:

- CI tools validate the models with the latest datasets before deployment, ensuring consistency. Once a model passes testing, its version and metadata are tracked by DVC. CML integrates detailed performance metrics into GitHub pull requests, keeping all stakeholders informed.

Result Visualization:

CML generates detailed reports, including model performance, which are integrated into Git comments or displayed in tools like DVC Studio. This can be seen in **Fout! Verwijzingsbron niet gevonden.**.

Docker Containerization for Deployment: Explanation

1. DVC Tracking

After completing the training within the CML pipeline, the results are visualized and presented in DVC studio. The developer can make a manual version tag to the model along with the metadata tracked in the Git Repository. This allows the team to easily access and track versions and the integrity of the data. When the model is approved and versioned, the CD pipeline can be triggered.

2. Trigger of CD pipeline

When the pull-request is accepted, GitHub will automatically trigger the CD workflow. And it will try to get the latest "production" model GTO-tag, as described in DVC studio. Once the model is obtained, it will continue to Docker steps.

3. Docker Containers for deployment

The pipeline triggers the Docker build process. It builds a Docker image that contains the application code, the trained model, and all necessary dependencies. This image encapsulates everything required to run the application in a consistent environment, whether locally, on a server, or in the cloud.

Once the Docker image is built, it is pushed to a **Docker registry** (such as Docker Hub and/or a GitHub registry). Docker images are portable, which means the application can run in any environment with Docker installed, ensuring consistency across different stages of the pipeline (development, testing, production).

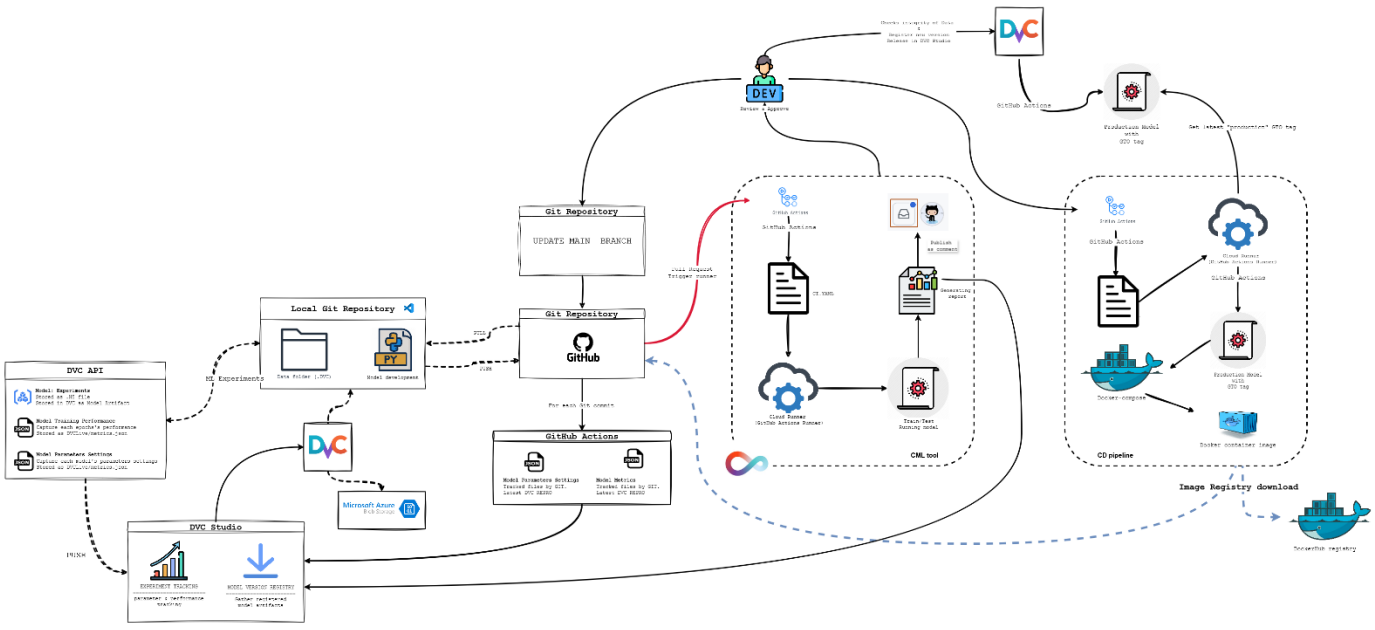


Figure 4: CD for ML workflows

This **CI/CD pipeline**, which integrates **DVC** and **Docker**, automates and streamlines the process of building, training, versioning, and deploying machine learning models or applications. By using **DVC** for data versioning and **Docker** for consistent containerized deployment. The entire pipeline—from code changes to automated deployment—ensures faster time to market, reduced human error, and more reliable application updates. The use of tools like **DVC Studio** and **CML** further enhances collaboration and performance tracking, leading to a more effective and transparent development process.

Realisation of the Proposed Architectural Design and Validation.

Setup and Configuration of DVC for Data and Model Versioning

To implement version control for large datasets and machine learning models, follow these steps to configure **DVC (Data Version Control)** in your project:

1. Configure DVC within a project to version control large datasets and models.

DVC can be installed via Python's package manager. Execute the following command in your project environment:

```
pip install dvc
```

This installs the DVC tool, which enables efficient management of large files and directories that cannot be handled effectively by traditional version control systems like Git.

After installation, initialize DVC in your project directory by running:

```
dvc init
```

This command performs the following operations:

- Sets up DVC configuration files and metadata.
- Creates a .dvc/ directory, which contains project-specific DVC configurations.
- Updates the .gitignore file to prevent tracked datasets or models from being included in Git.

DVC is now configured for your project and ready to track datasets and models efficiently.

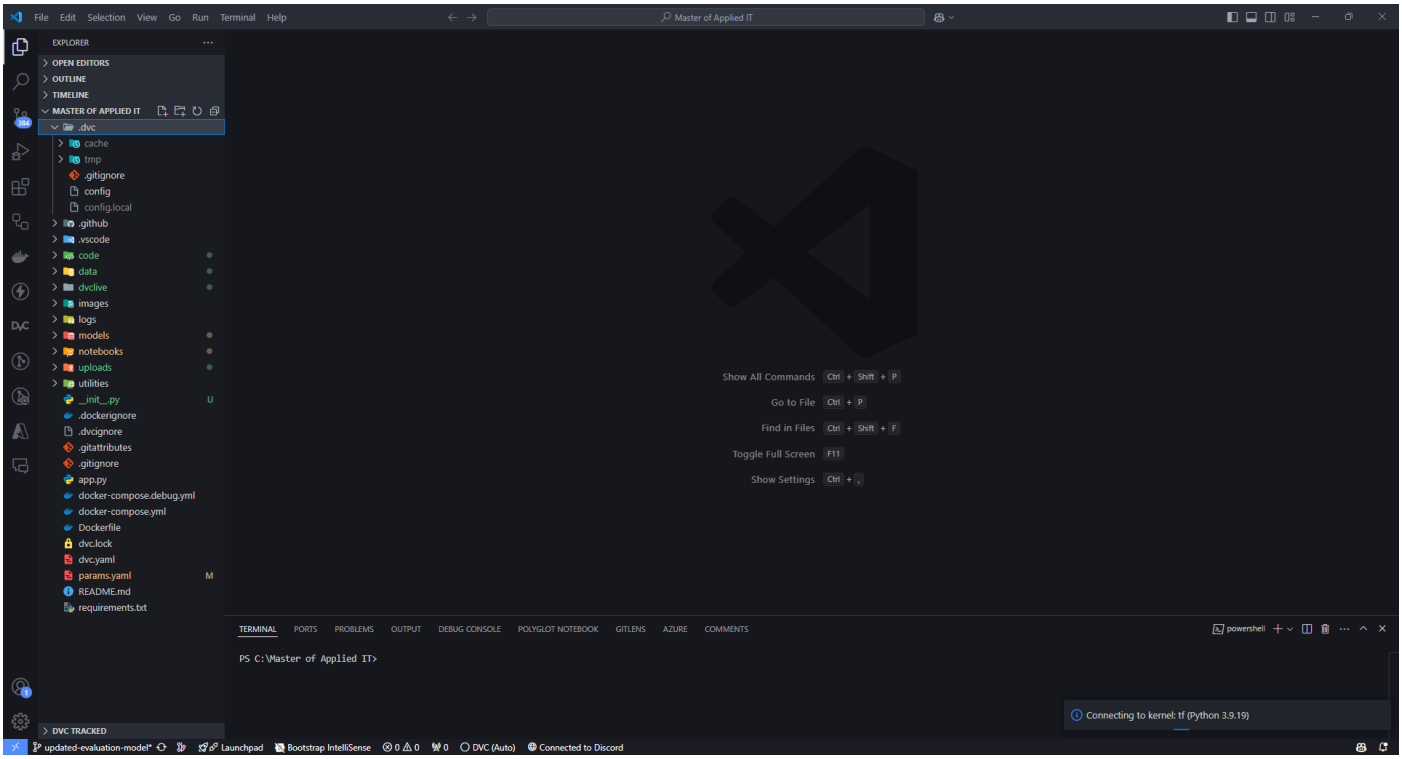


Figure 5: This will make a .DVC folder in the workspace

2. How to Configure DVC with Azure Blob Storage for Large Data Storage

This outlines the steps to configure **DVC (Data Version Control)** to use Azure Blob Storage for managing large datasets and models. By offloading data storage to Azure, you can efficiently track and version large files without bloating your Git repository.

Prerequisites

- **DVC Installed: Ensure DVC is already installed and initialized in your project.**
- **Azure Storage Account: An Azure Storage Account with a container created for storing your datasets and models.**

a. Install DVC with Azure Support

To enable DVC to interact with Azure Blob Storage, you need to install the Azure extension for DVC:

```
pip install dvc[azure]
```

This installs the necessary dependencies to interact with Azure Blob Storage.

b. **Configure an Azure Storage Account**

4. Go to the **Shared Access Signature (SAS)** section of the storage account.
5. Generate an SAS token and note the **Account Name** and **Account Key**.

Note: While **Azure Active Directory (AAD)** is a preferred method for secure authentication, SAS is used here due to specific restrictions (settings of Fontys).

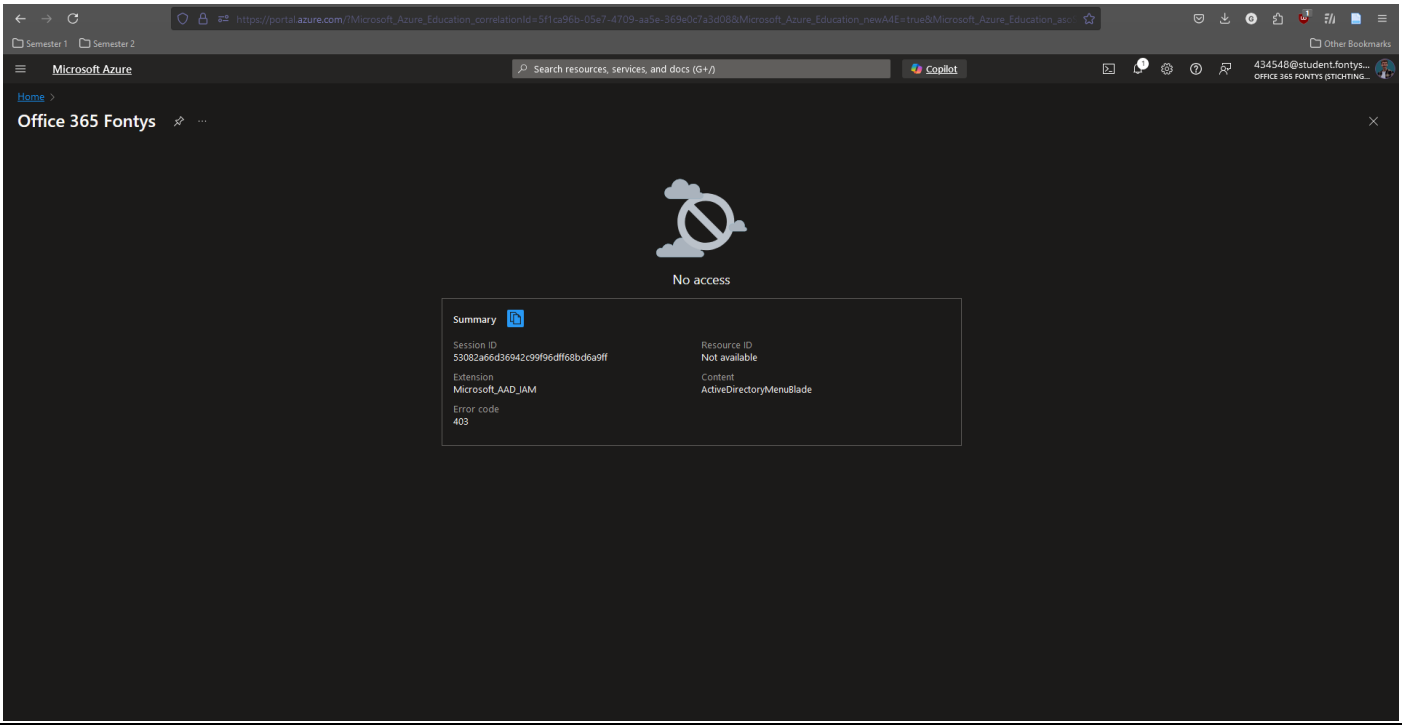


Figure 6: 403 on Active Directory with Azure Student Credit – unable to set-up Active Directory and manage SAS-tokens

The token can be set with a expiration date. Regularly regenerating SAS tokens and having an expiration policy is a good security practice to minimize the risk of unauthorized access. And setting up User roles.

3. Configure DVC to Use Azure Blob Storage

To configure the Remote Azure storage in DVC we need to add a the remote credentials:

```
dvc remote add -d myremote azure://<container-name>/<path-to-folder>
```

DVC needs your Azure Storage account credentials to interact with the Blob Storage.

Run the following commands, replacing <your-account-name> and <your-account-key> with your actual Azure account credentials:

```
1. dvc remote modify myremote azure_storage_account_name <your-account-name>
2. dvc remote modify myremote azure_sas_token "<your-sas-token>"
```

C. Start adding models and Data with DVC tracking

This explains how to track large datasets or models using DVC. The process involves adding files or folders to DVC, committing changes to Git, and enhancing the development environment with useful tools.

Add the Target Dataset or Model: If you have a folder named results/ that contains your dataset or model, add it to DVC using the following command:

```
dvc add results/
```

This will create a .dvc file (**results.dvc**) that tracks the file/folder. It will store metadata about the file in the **.dvc file** (like its hash value). And moves the actual large file(s) into a cache directory that DVC manages, so they aren't tracked directly by Git.

DVC modifies your Git repository by adding .dvc files. These need to be committed to Git so that the version control system tracks them.

```
git add results.dvc .gitignore
git commit -m "Track large dataset with DVC"
```

Additionally, we need to install some visual studio packages to make the development easier. Since we can see all tracked files within the workspace.

```
1. Azure storage
2. DVC
```

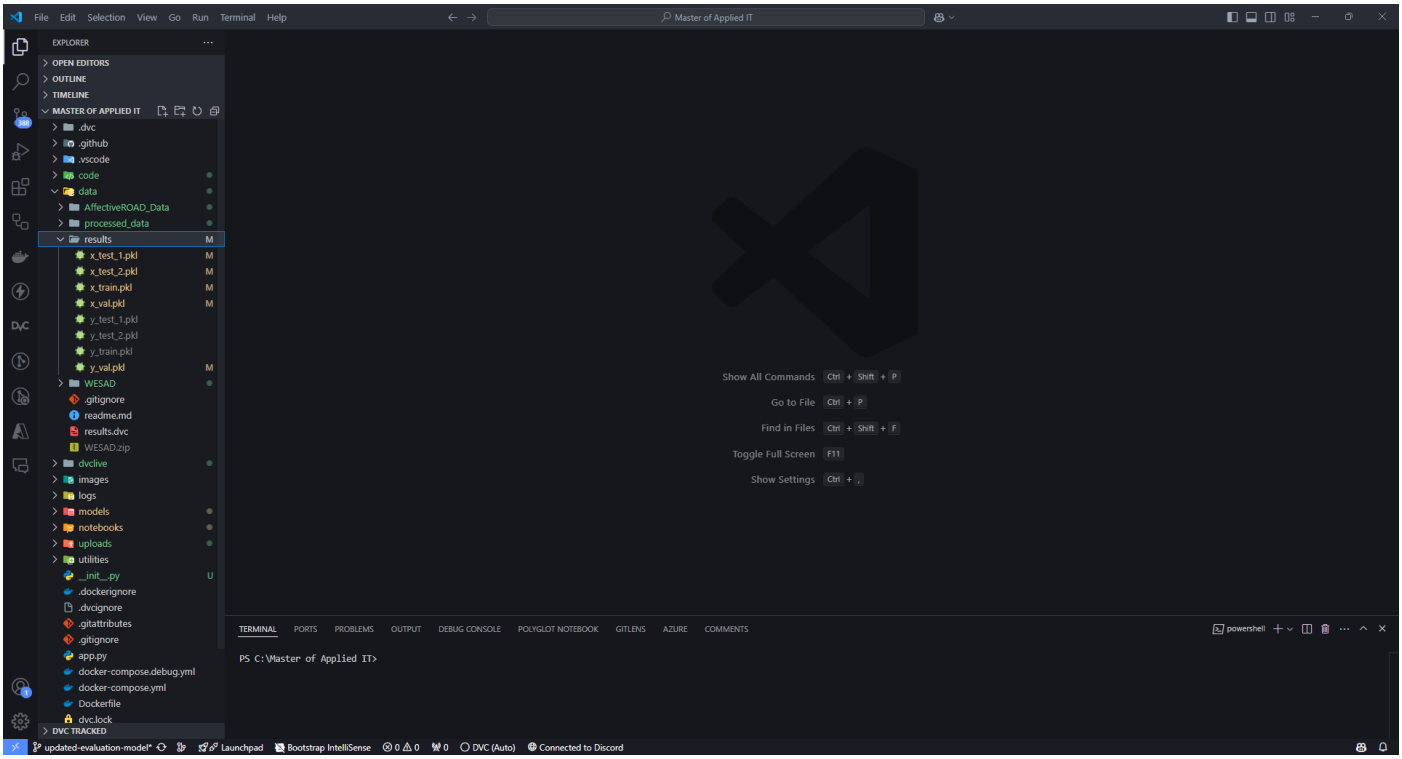


Figure 7: Note that results.dvc is tracked

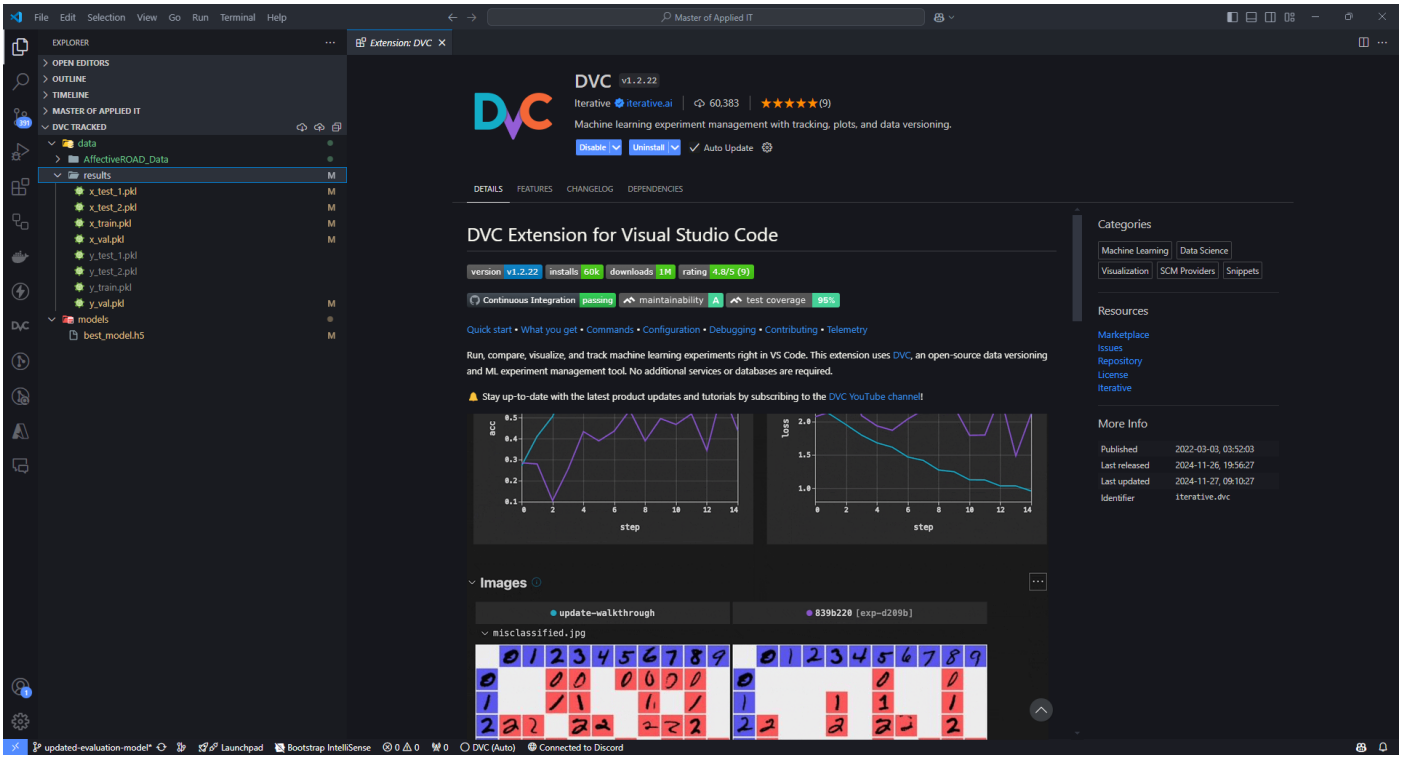


Figure 8: Tracked files by DVC on the workspace. We can pull these from Azure.

Experiment Tracking

A. How to use DVC for experiment tracking, where each experiment includes model performance metrics and parameters.

To showcase how experiment tracking works with DVC, integrate DVCLive into the project to log metrics and parameters dynamically during training. DVCLive simplifies tracking by capturing key details about each experiment while running your pipeline. DVCLive acts as a "reporter" that logs these metrics in real-time. When integrated with DVC, every experiment (with its data, code, and parameters) is automatically tracked, versioned, and easily reproducible.

You can run experiments with different parameter values directly from the command line:

```
dvc exp run --set-param kernel_regularizer =0.0001 epochs = 10
```

or by running the code, since DVCLive will report the performance, performance and used dataset to DVC.

```
PS C:\Master of Applied IT> dvc exp run
Reproducing experiment 'upper-sums'
Building workspace index                                                    |331 [00:00, 1.59kentry/s]
Comparing indexes                                                           |330 [00:00, 2.48kentry/s]
Applying changes                                                            |0.00 [00:00,     ?file/s]
Running stage 'training':
```

params.yaml M

```yaml
model:
  learning_rate: 0.001
  epochs: 10
  batch_size: 32
  activation: relu
  metrics: ['EDA', 'BVP','TEMP', 'ACC']
  kernel_size: 3
  shuffle_buffer_size: 1024
  kernel_regularizer: 0.0001
  # Input shapes for each metric
  input_shapes:
    EDA: 32    # For EDA, 4Hz over 8 seconds
    BVP: 256   # For BVP, 32Hz over 8 seconds
    TEMP: 32   # For TEMP, 4Hz over 8 seconds
    ACC: 256   # For BVP, 32Hz over 8 seconds
  input_features: 1
```

```
PS C:\Master of Applied IT> dvc params diff --target params.yaml --all
Path         Param                     HEAD                          workspace
params.yaml  model.activation          relu                          relu
params.yaml  model.batch_size          32                            32
params.yaml  model.epochs              30                            10
params.yaml  model.input_features      1                             1
params.yaml  model.input_shapes.ACC    256                           256
params.yaml  model.input_shapes.BVP    256                           256
params.yaml  model.input_shapes.EDA    32                            32
params.yaml  model.input_shapes.TEMP   32                            32
params.yaml  model.kernel_regularizer  0.01                          0.0001
params.yaml  model.kernel_size         3                             3
params.yaml  model.learning_rate       0.001                         0.001
params.yaml  model.metrics             ['EDA', 'BVP', 'TEMP', 'ACC'] ['EDA', 'BVP', 'TEMP', 'ACC']
params.yaml  model.shuffle_buffer_size 1024                          1024
PS C:\Master of Applied IT>
```

Figure 9: Parameters file which displays change in epochs and kernel_regularizer

After running experiments, you can compare these using:

```
dvc exp list
```

```
TERMINAL  PORTS  PROBLEMS  OUTPUT  DEBUG CONSOLE  POLYGLOT NOTEBOOK  GITLENS  AZURE  COMMENTS

PS C:\Master of Applied IT> dvc exp list
64a0c1c:
        5477e2f [bored-repp]
        253d9da [burry-seer]
        593a249 [heady-ryas]
        53f1cb3 [osmic-many]
        2ab1925 [sharp-hack]
        f1106a4 [trial-bump]
        f237b7e [balky-delf]
        5a3314b [beady-merk]
        860213b [biped-lien]
        2a392ff [brief-kern]
        a91df37 [freer-glen]
        8620351 [glial-dogs]
        72274e0 [known-fore]
        f7b6709 [lathy-math]
        9a4ab1d [manky-lays]
        0ca98c7 [muddy-fibs]
        53d11ed [optic-jass]
        3f58bca [radio-clog]
        cd9ddbd [raked-orcs]
        a6b0a2d [risen-berk]
        98e71ba [round-kina]
        64e557e [scrap-gaol]
        20db80e [sixth-rits]
        21a8033 [sonic-sand]
        0bffdbd [splay-rods]
        2314ca7 [straw-prow]
        2832867 [sulfa-rain]
        e2910df [taunt-paws]
        1c915ab [third-drop]
        9051608 [vapid-ease]
        f6823e9 [vital-tods]
        276b617 [young-cull]
```

Figure 10: A list of all performed experiments.

```
TERMINAL  PORTS  PROBLEMS  OUTPUT  DEBUG CONSOLE  POLYGLOT NOTEBOOK  GITLENS  AZURE  COMMENTS

PS C:\Master of Applied IT> dvc exp diff burry-seer straw-prow
Path                  Metric                   burry-seer    straw-prow    Change
dvclive\metrics.json  eval.auc                 0.97656       0.98492       0.0083537
dvclive\metrics.json  eval.binary_accuracy     0.94413       0.96924       0.025106
dvclive\metrics.json  eval.f1_score            0.7762        0.85761       0.081406
dvclive\metrics.json  eval.loss                0.19819       0.14137       -0.056813
dvclive\metrics.json  eval.precision           0.66829       0.83175       0.16345
dvclive\metrics.json  eval.recall              0.92568       0.88514       -0.040541
dvclive\metrics.json  test_1_auc               0.98954       0.87945       -0.11009
dvclive\metrics.json  test_1_binary_accuracy   0.96159       0.92034       -0.041252
dvclive\metrics.json  test_1_f1_score          0.84024       0.56923       -0.27101
dvclive\metrics.json  test_1_loss              0.099685      0.47194       0.37226
dvclive\metrics.json  test_1_precision         0.83529       0.80435       -0.030946
dvclive\metrics.json  test_1_recall            0.84524       0.44048       -0.40476
dvclive\metrics.json  test_2_auc               0.48256       0.46692       -0.015632
dvclive\metrics.json  test_2_binary_accuracy   0.87703       0.87568       -0.0013514
dvclive\metrics.json  test_2_loss              1.5649        1.849         0.28409
dvclive\metrics.json  train.auc                0.99612       0.99999       0.0038745
dvclive\metrics.json  train.binary_accuracy    0.97192       0.99948       0.027562
dvclive\metrics.json  train.f1_score           0.97266       0.99948       0.026828
dvclive\metrics.json  train.loss               0.022683      0.014194      -0.0084891
dvclive\metrics.json  train.precision          0.94785       0.99931       0.051461
dvclive\metrics.json  train.recall             0.99879       0.99966       0.00086135

Path                Param                     burry-seer              straw-prow                    Change
dvclive\params.yaml model.kernel_regularizer  0.001                   0.01                          0.009900000000000001
dvclive\params.yaml model.metrics             ['EDA', 'BVP', 'ACC']   ['EDA', 'BVP', 'ACC', 'TEMP']  diff not supported
params.yaml         model.kernel_regularizer  0.001                   0.01                          0.009900000000000001
params.yaml         model.metrics             ['EDA', 'BVP', 'ACC']   ['EDA', 'BVP', 'ACC', 'TEMP']  diff not supported
PS C:\Master of Applied IT>
```

Figure 11: Comparision between to experiments' performance

Implementation of Version Control and Traceability for Model Management

A. Implement DVC to create a version history of machine learning models and datasets for traceability.

To check versions of data and models, we can check this with a GIT function:

```
git log data/data.txt.dvc
```

With this function, we can see all commits happening. So, if we decide to reproduce one experiment, we can easily pull the data, model and code altogether. In this example, for easy explaining what I precisely did, I made a .txt file called data. This file is tracked by DVC and has two versions: “version 1” and “version 2”.

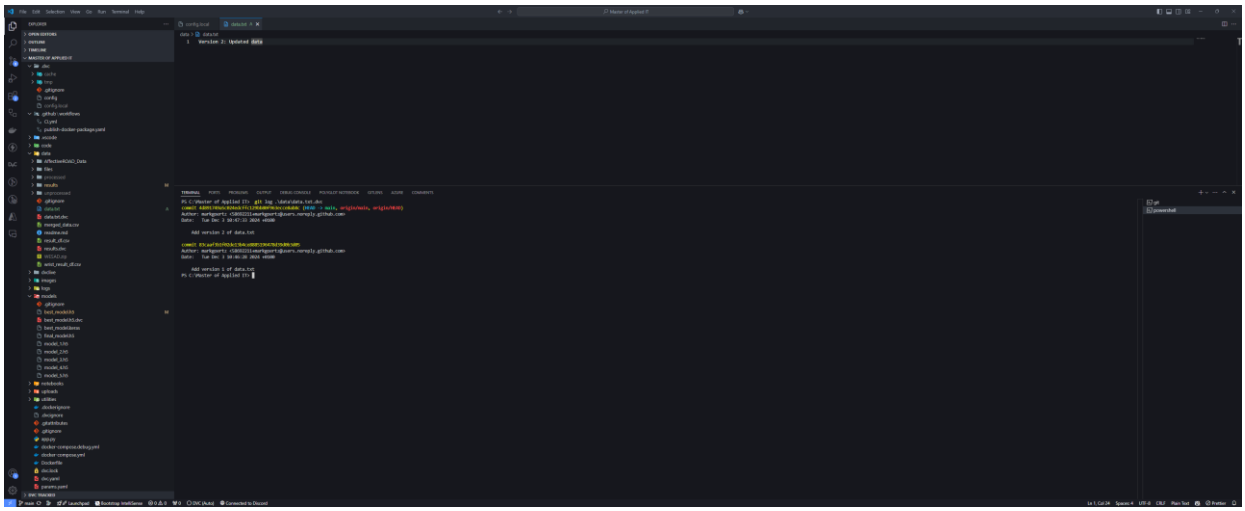


Figure 12: We can see the commit versions of GIT. With version 1 and version 2 as commit message.

To gather the files of version 1. We have to GIT checkout the commit to get all code and DVC metafiles, and the workspace will become a detached HEAD. Once this is successfully completed, we can pull the files from DVC to get all data and models with the corresponding commit:

```
git checkout commit 1
dvc pull
```

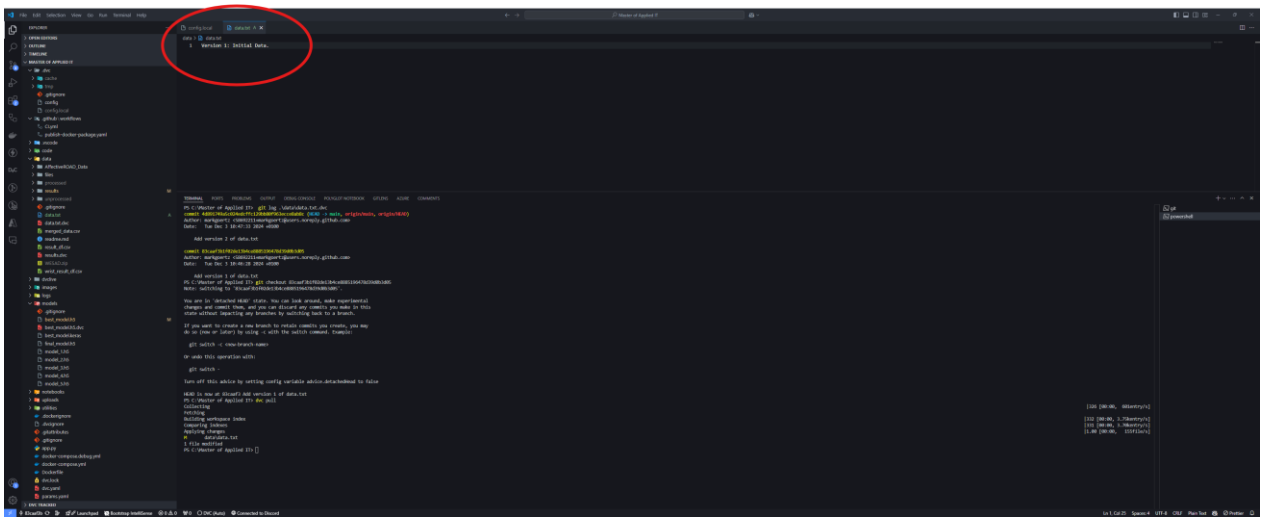


Figure 13: Version 1 of the data.txt file

When we want to change commits, we can easily do the same commit. This will update the code, and metafiles with the corresponding commit. And we can do the DVC command again to get all data and model files with the commit.

```
git checkout commit 2
dvc pull
```

The same file of data.txt has changed with version 2.

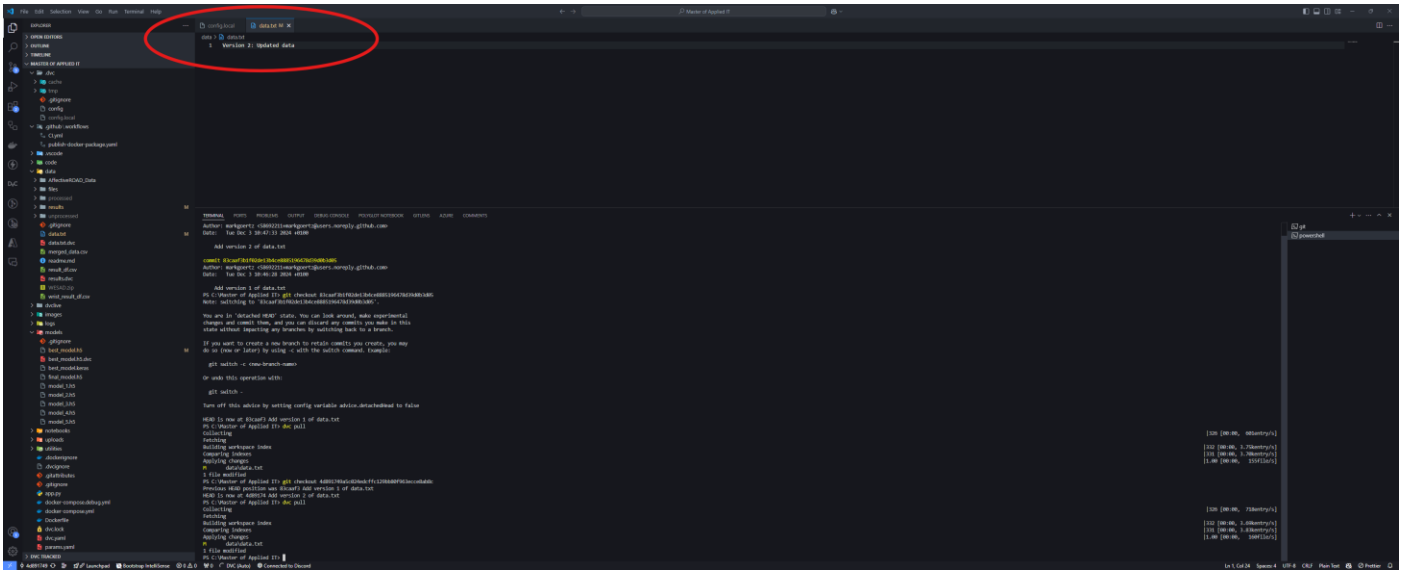
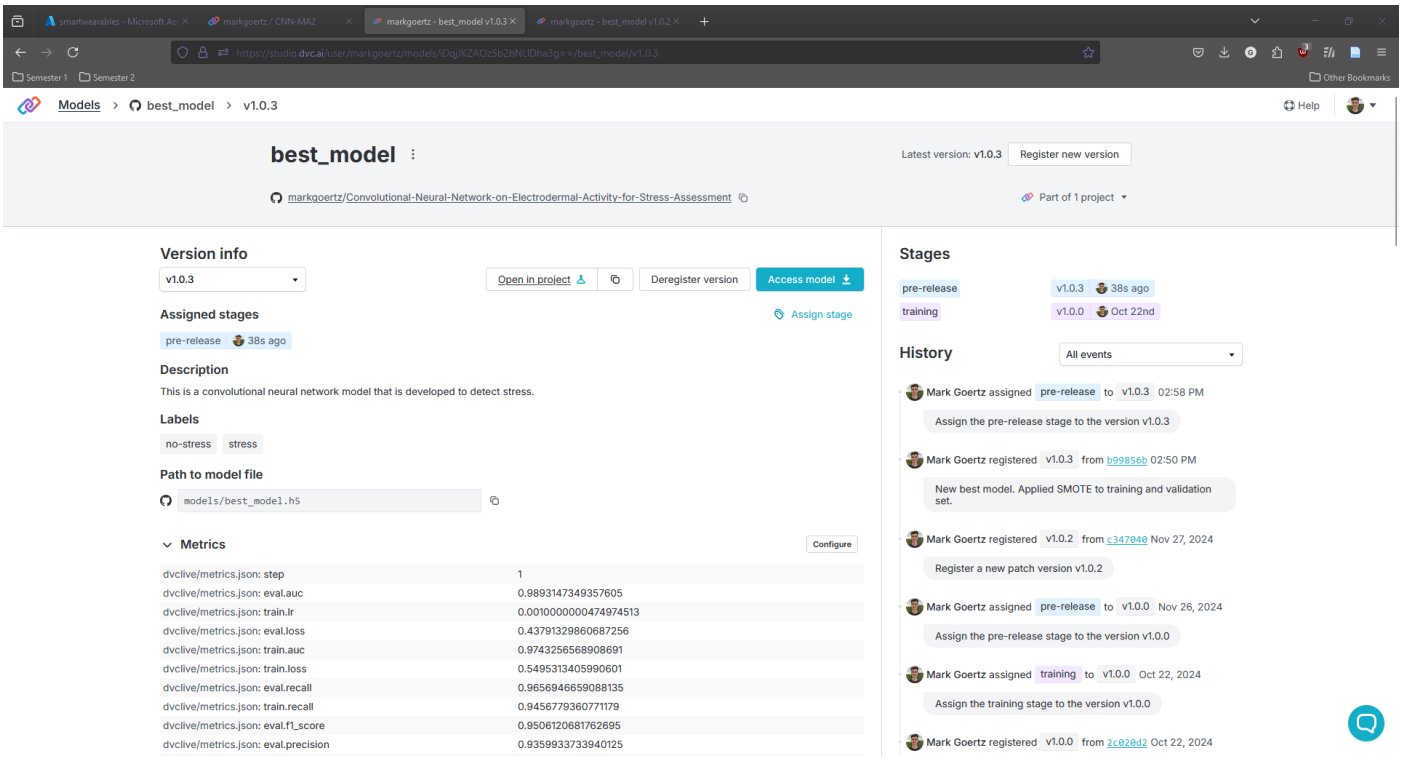


Figure 14: Note that the same file is now Version 2, due to version control

B. Track and manage model training parameters, hyperparameters, and datasets used in experiments.

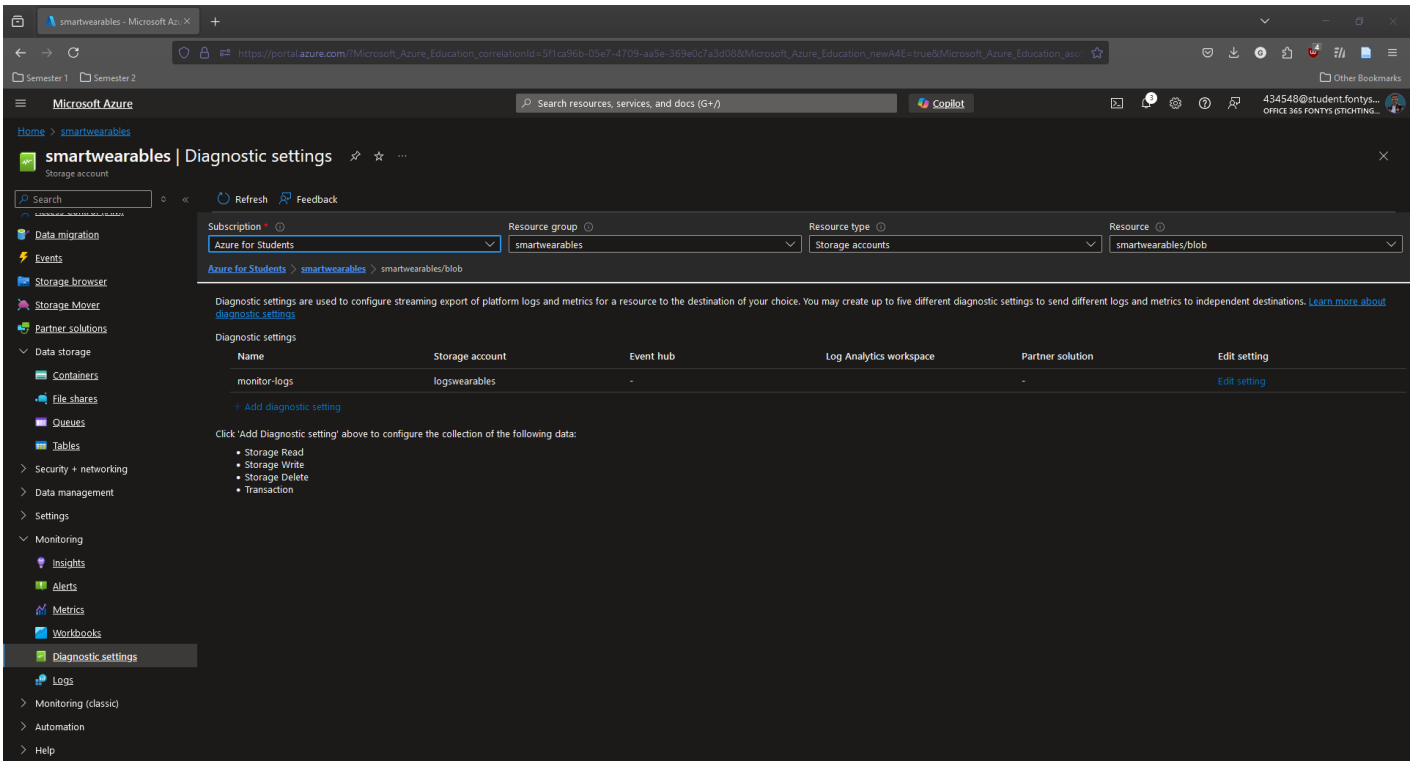


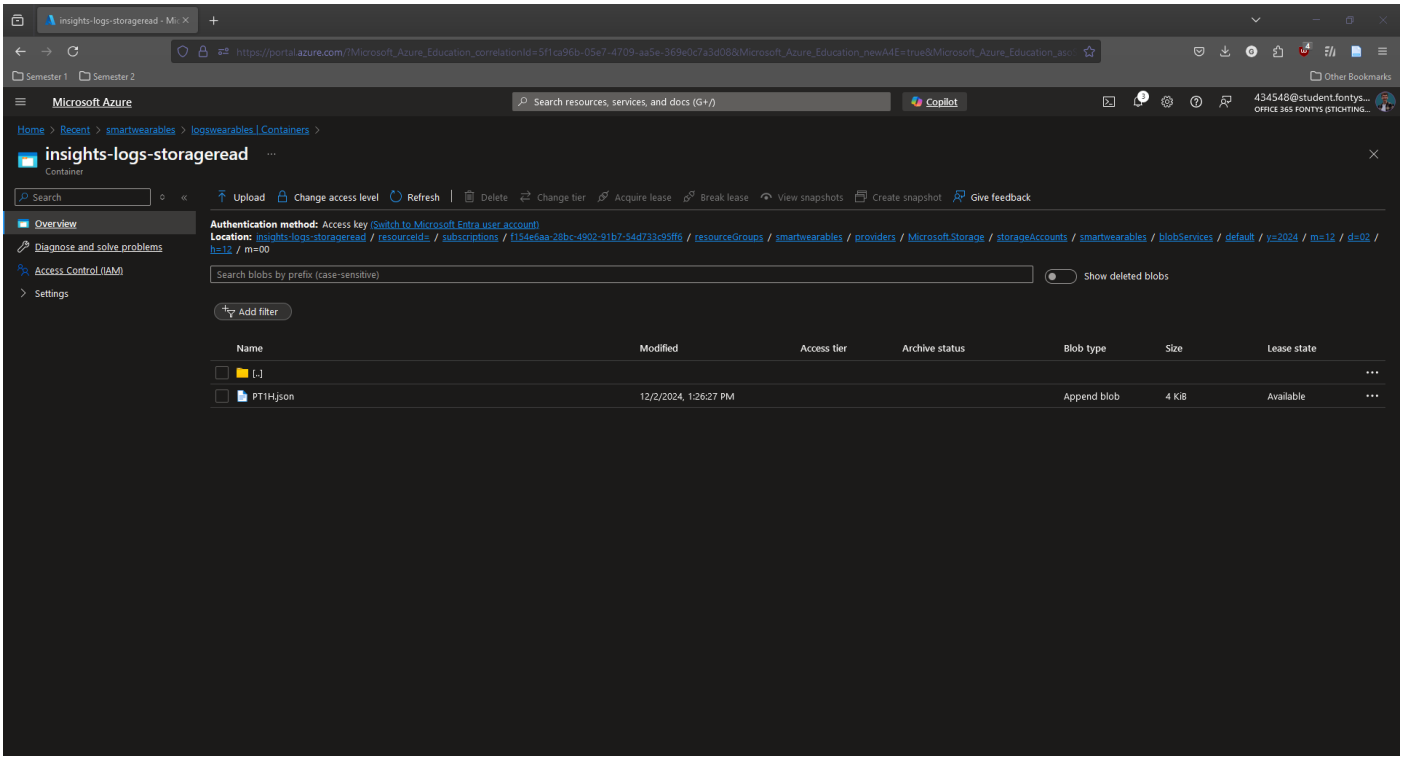
Logging Access in Azure Blob Storage Using Diagnostic Settings

Azure Blob Storage provides tools to monitor and log access to a storage account. By configuring Diagnostic Settings, we can capture detailed logs of all access and activities within your Blob Storage. These logs help with performance monitoring, security analysis, and compliance requirements.

For Blob Storage, access logs include details such as:

- Read, write, and delete operations
- Authentication methods
- Client IP addresses





Inspect log file shows:

Inspect log details such as:

- **Timestamp:** When the operation occurred.
- **Operation Name:** Type of access (e.g., GetBlob, PutBlob).
- **Client IP Address:** Source of the access.
- **Authentication Type:** Method used for authentication.
- **HTTP Status Code:** Outcome of the operation (e.g., 200, 404).


```
{
    "time": "2024-12-02T12:12:29.3668323Z",
    "resourceId": "/subscriptions/f154e6aa-28bc-4902-91b7-
54d733c95ff6/resourceGroups/smartwearables/providers/Microsoft.Storage/storageAccounts/sma
rtwearables/blobServices/default",
    "category": "StorageRead",
    "operationName": "ListBlobs",
    "operationVersion": "2024-11-04",
    "schemaVersion": "1.0",
    "statusCode": 200,
    "statusText": "Success",
    "durationMs": 2,
    "callerIpAddress": "This is my Home IP-adress",
    "correlationId": "670ac368-c01e-0040-4cb3-44c141000000",
    "identity":     {
        "type":"SAS",
        "tokenHash":"key1(),SasSignature()"
    }, "location": "westeurope",
    "properties": {
        "accountName":"smartwearables",
        "userAgentHeader":"azsdk-python-storage-blob/12.23.1 Python/3.9.19 (Windows-10-
10.0.22631-SP0)",
        "clientRequestId":"b2f34507-b0a6-11ef-9c1b-b42e9985d288",
        "serviceType":"blob",
        "objectKey":"/smartwearables/dvcremote",
        "metricResponseType":"Success",
        "serverLatencyMs":2,
        "requestHeaderSize":575,
        "responseHeaderSize":210,
        "responseBodySize":1116,
        "tlsVersion":"TLS 1.3",
        "sourceAccessTier":"Invalid"
    },
    "uri":
"https://smartwearables.blob.core.windows.net:443/dvcremote?restype=container&comp=list&pr
efix=data/files/md5/&maxresults=1&sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupiytfx&se=2025-
04-16T02:48:32Z&st=2024-11-19T19:48:32Z&spr=https&sig=XXXXX",
    "protocol": "HTTPS",
    "resourceType": "Microsoft.Storage/storageAccounts/blobServices"
}
```

Establishment of DVC Studio for Collaboration and Visualization

A. Use DVC Studio to manage and visualize experiments, including parameters, metrics, and outcomes.

DVC Studio is a web-based tool that helps you visualize, organize, and compare machine learning experiments, including parameters, metrics, and outcomes. This guide explains how to integrate DVC Studio into your workflow.

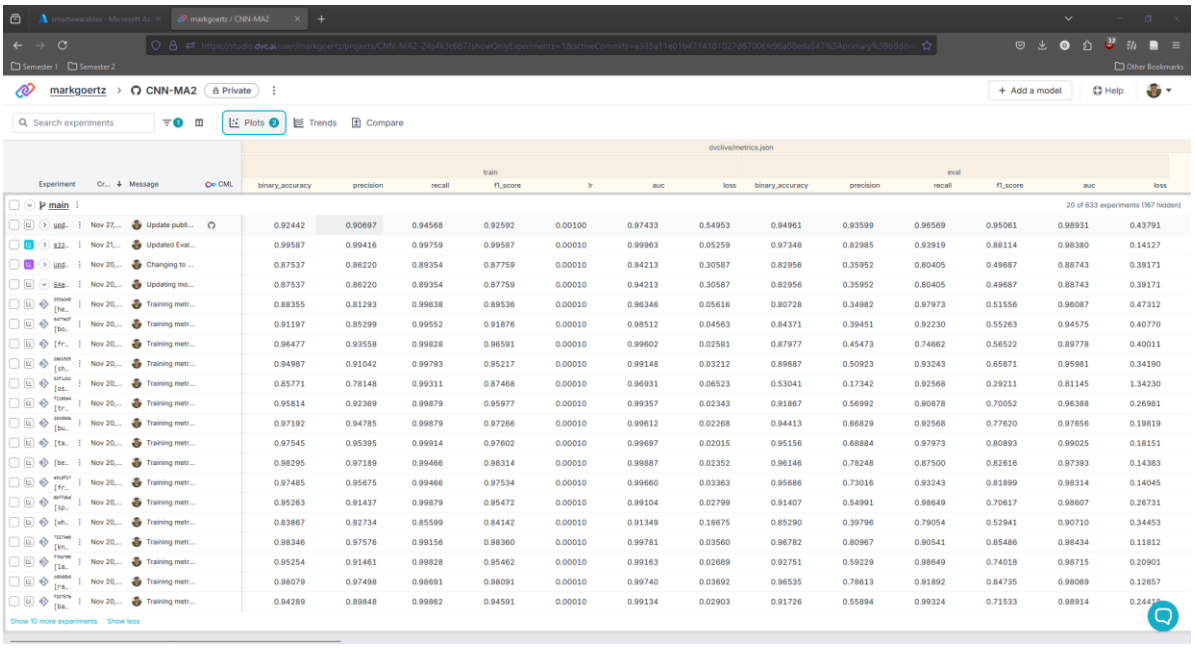


Figure 15: This showcases the experiments with performance metrics of training and evaluation

We can compare experiments, by clicking on the plot icon (currenty colored as blue and purple). And get to see plots of the training trajectory:

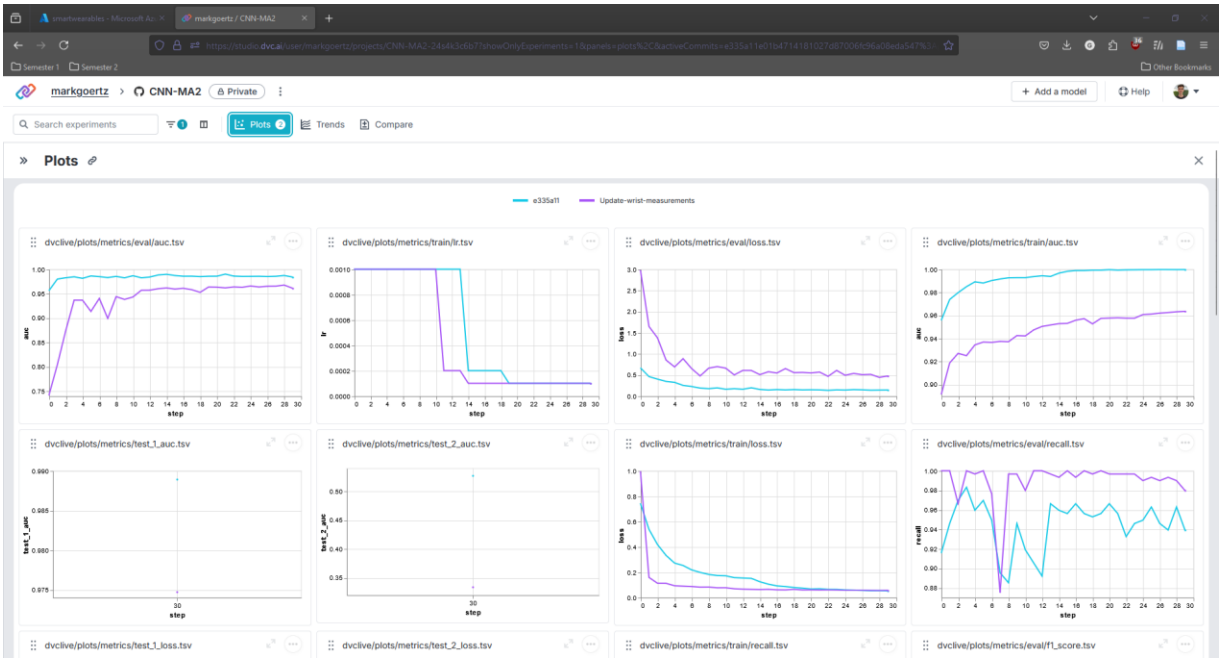


Figure 16: Comparing the two experiments

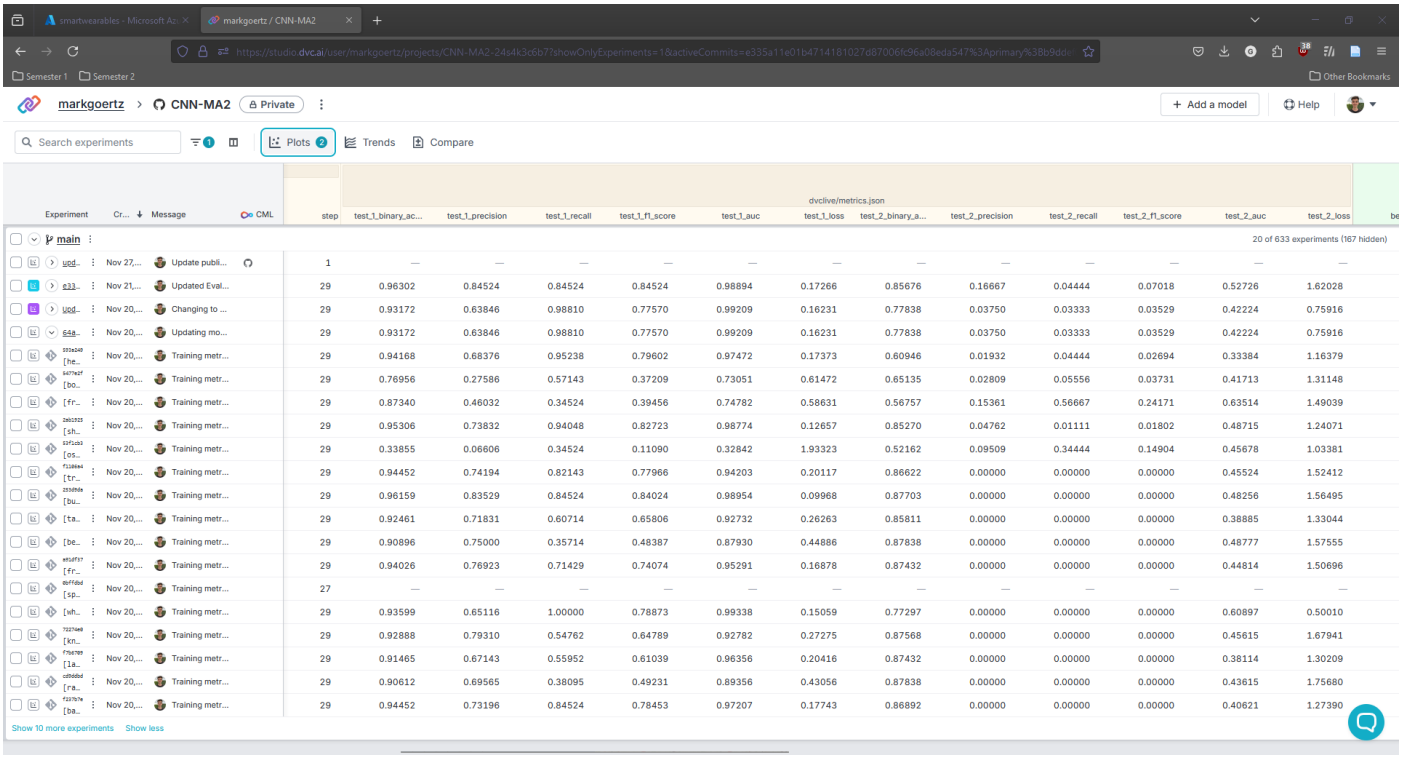


Figure 17: Performance of Test Subjects

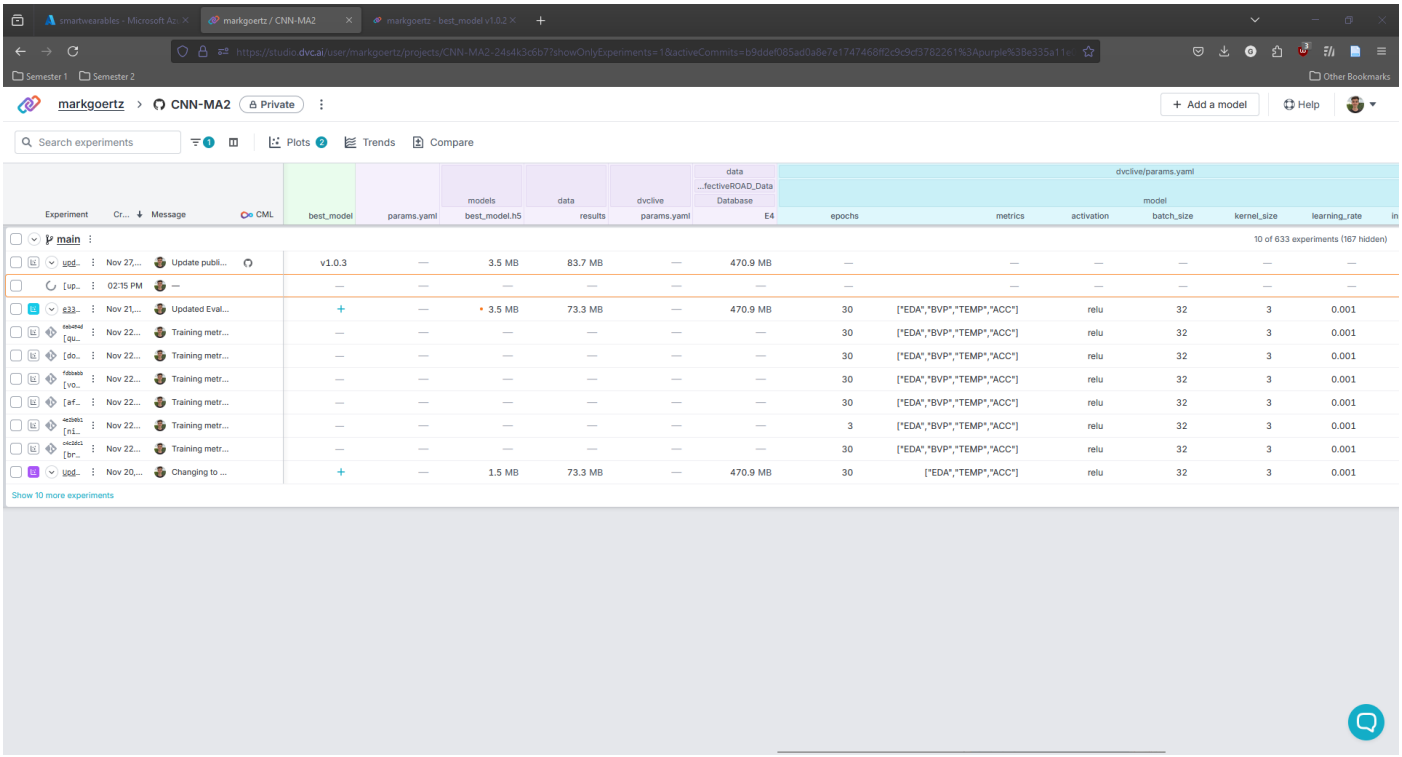


Figure 18: Model Registry, Used datasets and model parameters

Implementation of Continuous Integration/Continuous Delivery (CI/CD) with CML

The following outlines the steps taken to implement Continuous Integration and Continuous Delivery (CI/CD) for machine learning workflows, leveraging Continuous Machine Learning (CML). This approach automates processes, efficiency, traceability, and model performance monitoring.

A. Integration of CML into the CI/CD Pipeline

The integration began with configuring a version-controlled repository, using Git to manage all machine learning code, datasets, and configurations. CML was selected as the primary tool to enhance automation within the CI/CD pipeline. **CML (Continuous Machine Learning)** is an open-source tool that enables the integration of machine learning workflows into Continuous Integration/Continuous Delivery (CI/CD) pipelines. CML automates key processes in the machine learning lifecycle, such as model training, validation, and reporting, within a CI/CD framework.

Key configurations included:

- Installing CML to enable seamless integration with existing CI tools such as GitHub Actions.
- Setting up workflows to automate the execution of key processes like dataset pulling, model training, and validation. This ensured that every code or dataset update triggered a complete evaluation pipeline.

B. Automation of Workflows

Automated workflows were established to address three core stages of the machine learning lifecycle:

- **Model Training:**

Scripts for training were integrated into the pipeline, ensuring they utilized the latest datasets and configurations. Datasets were tracked using DVC, and training was automated to initiate upon updates on the main-branch.

- **Validation:**

Model validation processes were incorporated to evaluate performance metrics automatically.

C. Automated Reporting

A key component of this implementation was the automation of reporting, providing transparency into model performance after each run.

- **Generation of Reports:**

After every training and validation cycle, detailed reports were automatically generated. These reports included metrics such as accuracy, precision, recall, and loss curves, as well as visualizations comparing current and previous model performance.

- **Integration with Pull Requests**

Reports were seamlessly integrated into the development workflow by attaching them to pull requests as comments. This provided immediate feedback to stakeholders, improving collaboration and decision-making.

Screenshots of Continous Integration part.

Automatically triggers model training and evaluation when code or data is updated. Integrated with Git-based repositories and works with tools like GitHub Actions, GitLab CI, and Bitbucket Pipelines. Automatically generates and publishes performance reports, such as accuracy, precision, recall, and visualizations, after each run.

In the screenshots below, you can see the CI-part of this development pipeline. It builds, lint and trains the model on a cloud-runner. To ensure that the pushed code and model is working as expected.

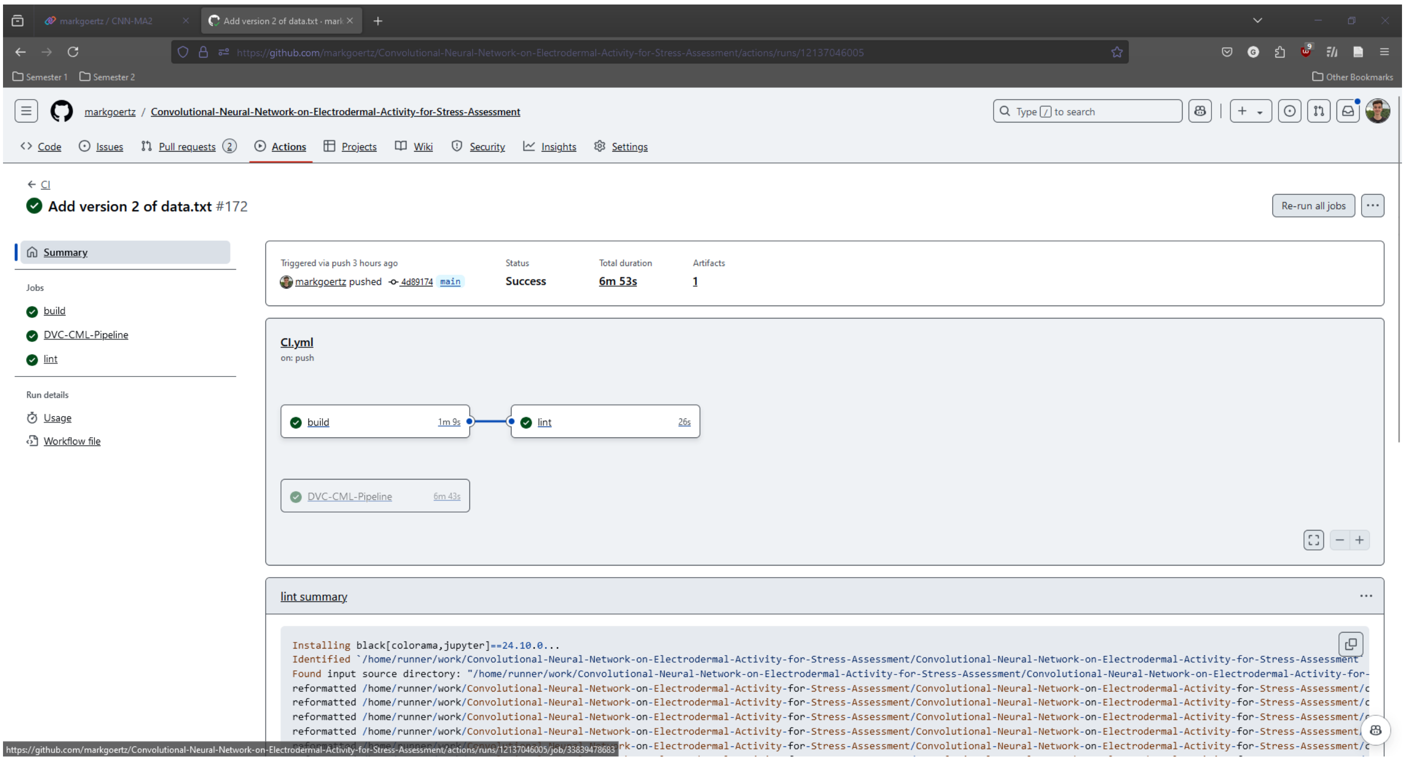


Figure 19: Running a build, linter and DVC-CML model training with the training code.

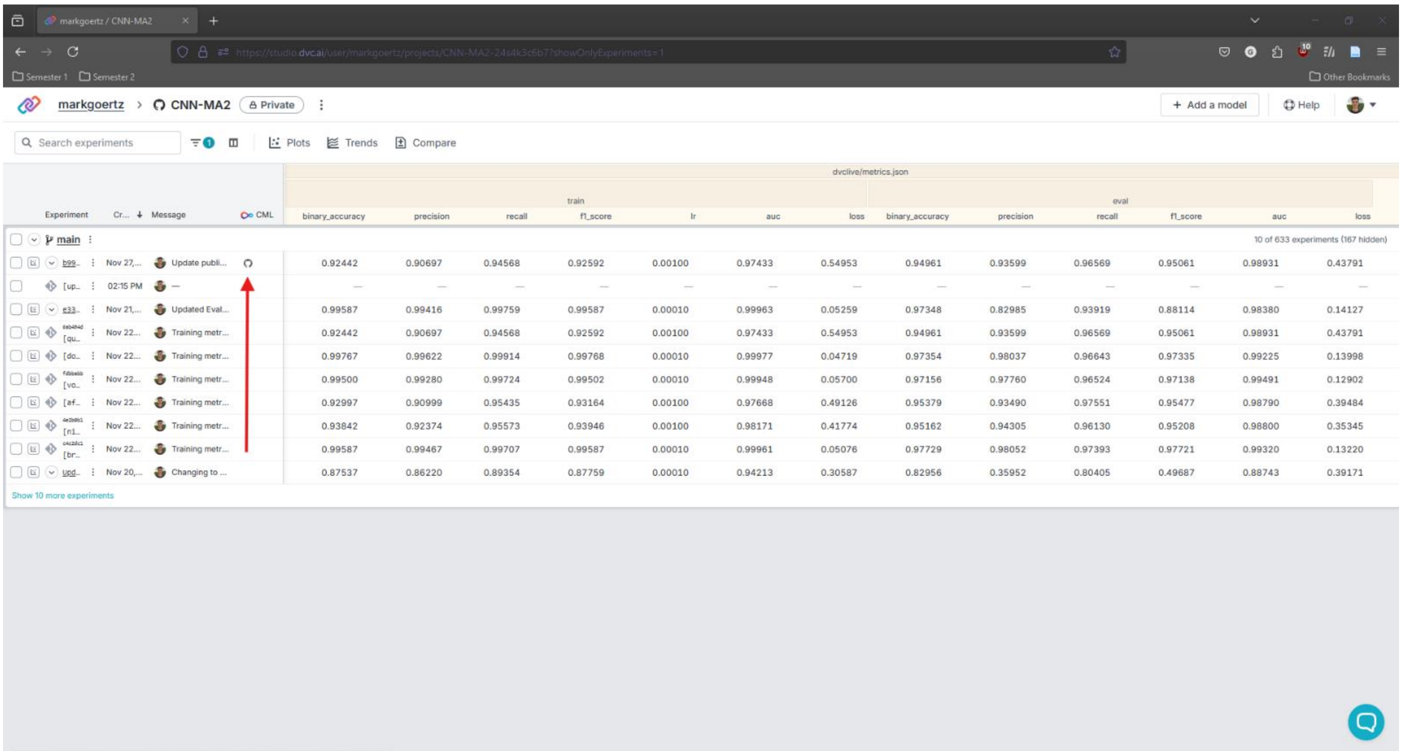
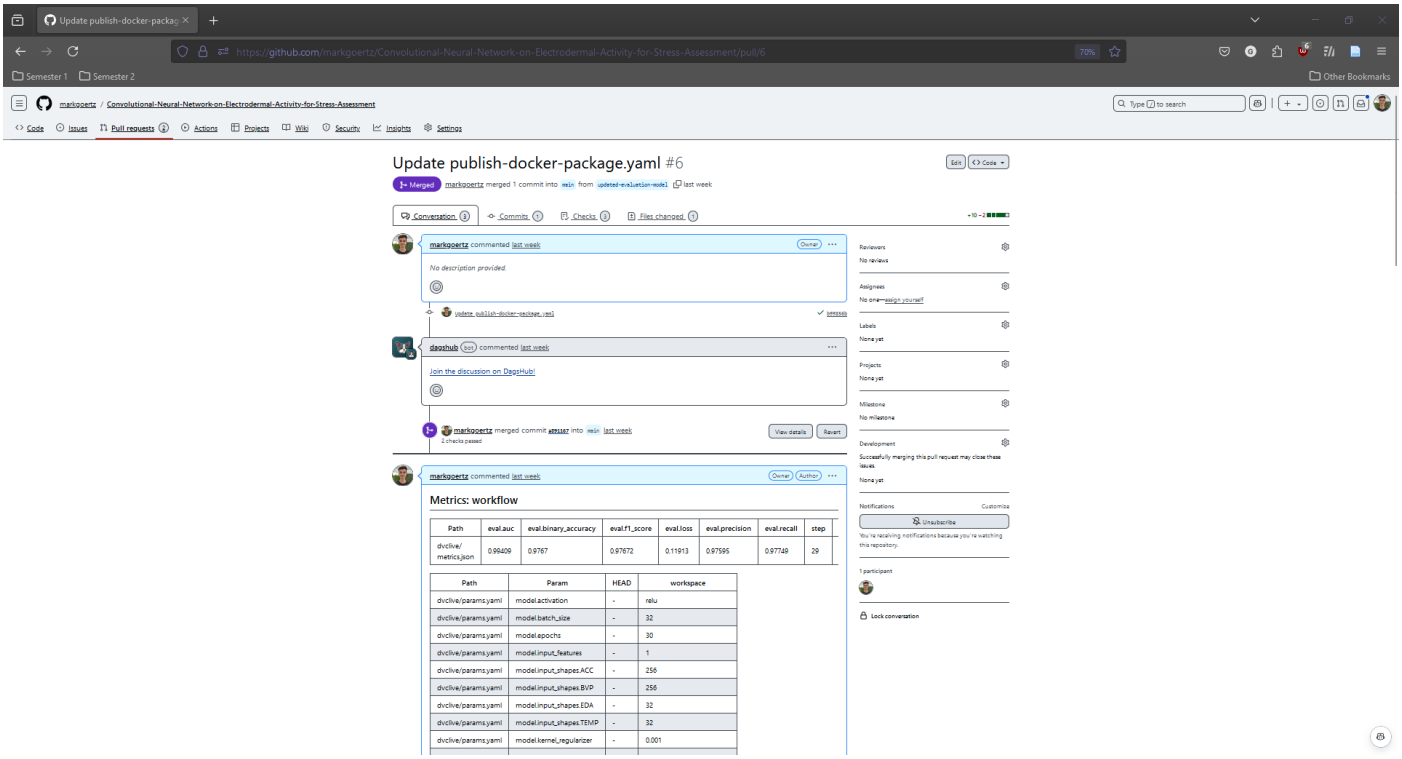


Figure 20: The performance of the training will be reported in DVC studio. You can find these with the GitHub symbol.

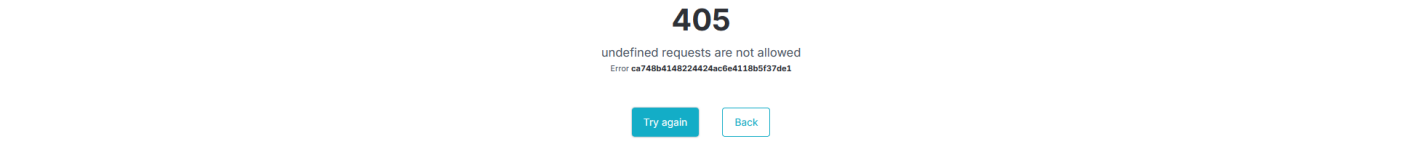
The automated training in the cloud will only be triggered when pull-requests are created towards the main-branch. Meaning that the main ‘production’ branch will get new updates. To ensure that the new updates are fault-proof, the model gets trained, to compare those results with the results from the runner.

The runner will produce a report in the pull-request before it can be merged with the main-branch. Where you can see the hyperparameters, used model configuration settings and additional created plots.

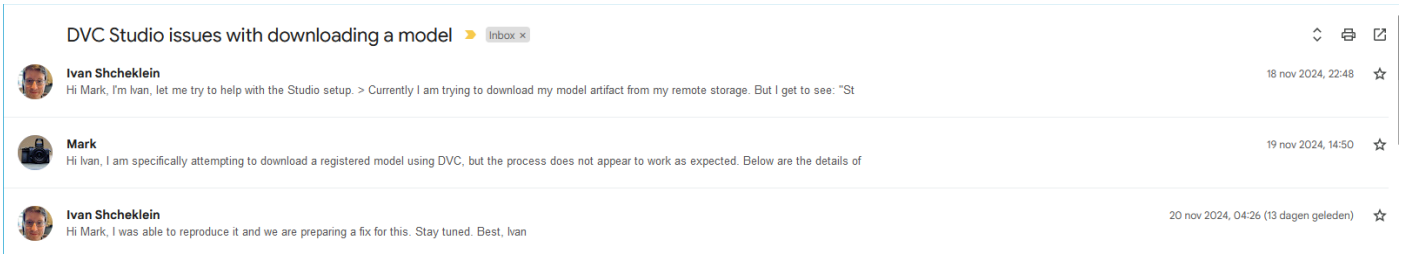


D. Automated Deployment

Due to an issue with DVC Studio's model registry, I was unable to fully integrate model deployment via model tags into our Continuous Integration/Continuous Deployment (CI/CD) pipeline at this stage. Although the pipeline successfully tracks, trains, validates, and reports on models, the deployment integration could not be completed due to the registry issue.



After contacting DVC Studio's support team, I sent a follow-up message for further clarification. I am still awaiting a fix, which has been pending since November.



Given the constraints, I had to find an alternative method to demonstrate the deployment process. Due to a bug in DVC Studio, I was unable to directly retrieve the registered model for our CD pipeline. As a workaround, I proceeded by pulling the latest model from the DVC metafile, which allowed me to showcase the deployment functionality within the pipeline.

While this method enabled me to continue the demonstration, it is important to emphasize that pulling the latest model is not a recommended practice for production environments. Using dvc pull to fetch the latest model directly from the repository bypasses the model versioning and registration process. This introduces several risks, such as:

- **Inconsistent model versions**: The model deployed may not be the exact version intended for production, as it is not versioned or tracked appropriately.
- **Model drift:** The latest model could differ from previous versions due to data changes, leading to unpredictable behavior in production.

In a proper deployment scenario, it is essential to deploy specific, registered model versions to ensure stability, reproducibility, and reliability in production systems.

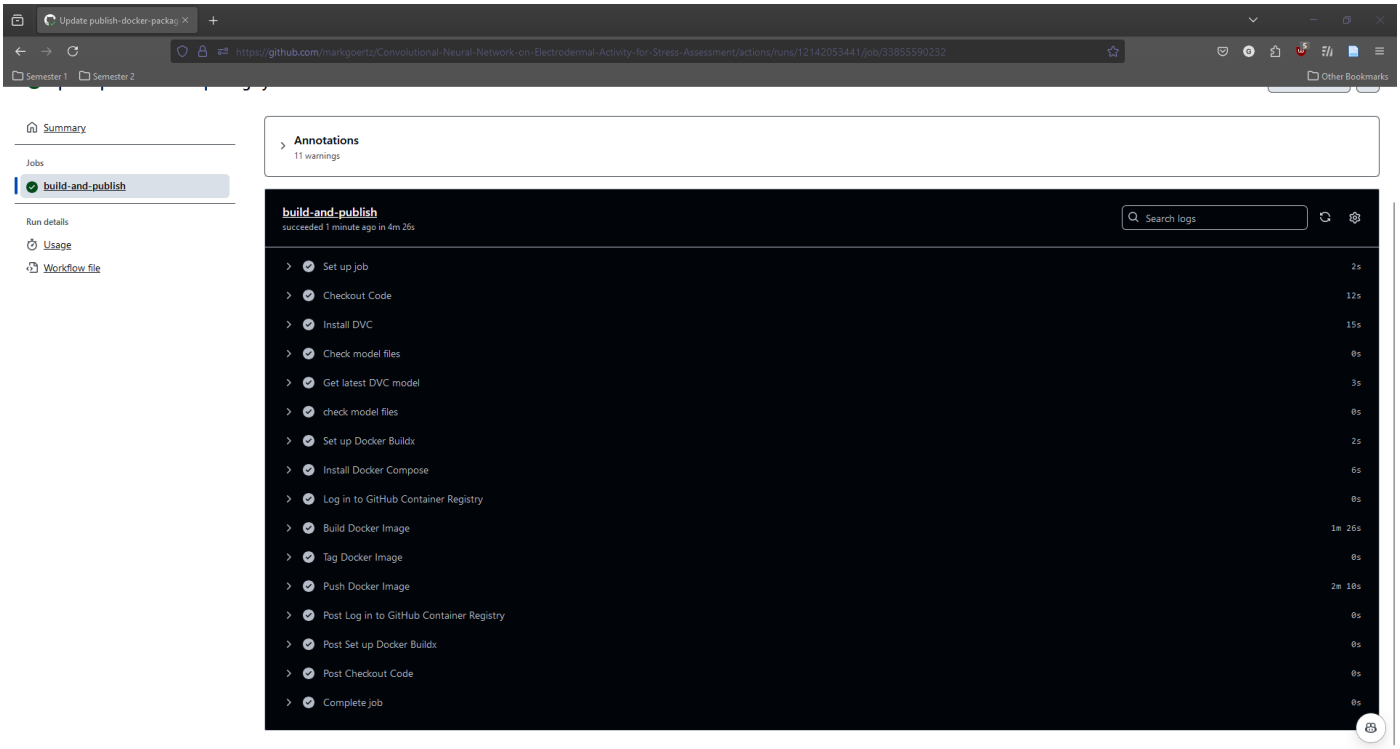


Figure 21: Docker registry after fetching the latest model.

To demonstrate this process, the model was pulled from the repository and deployed via Docker, creating a Docker image with the latest model version:

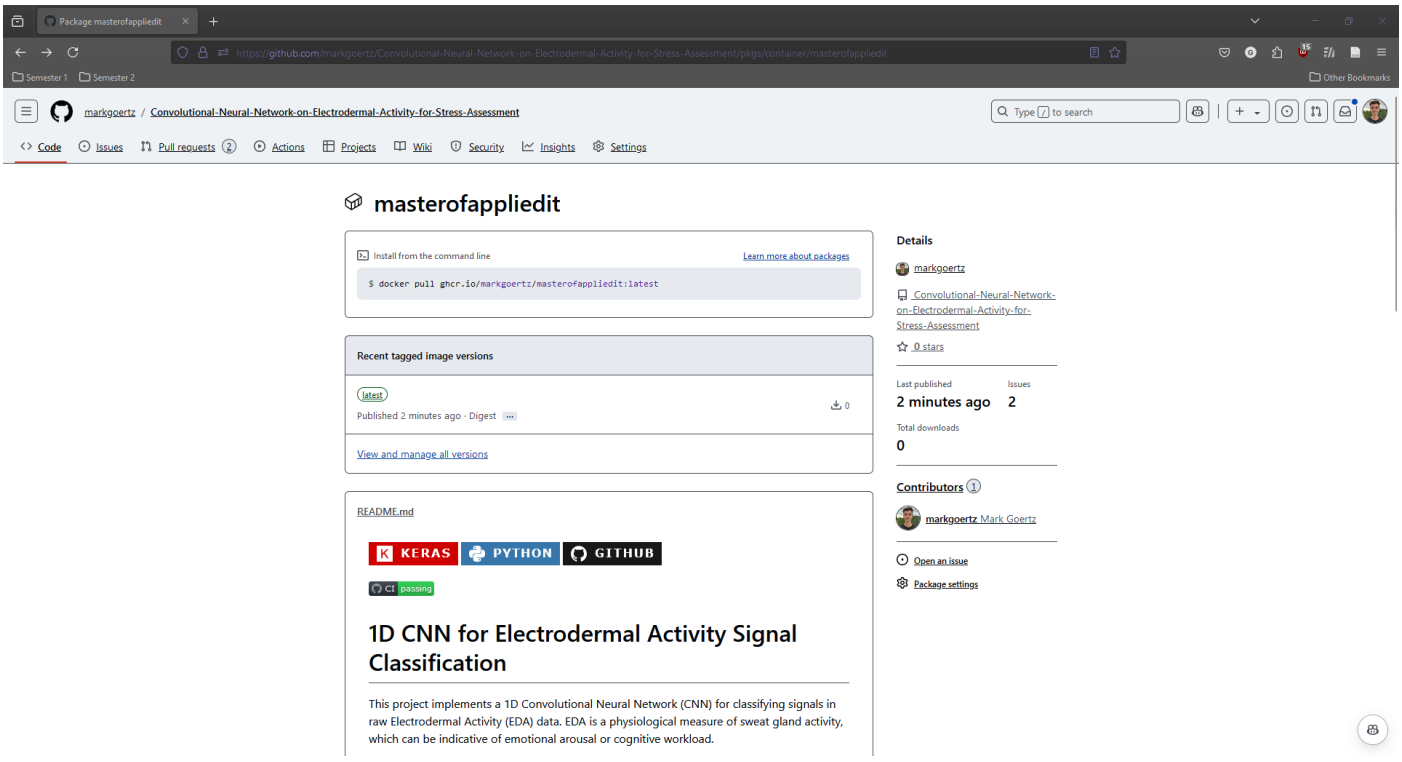
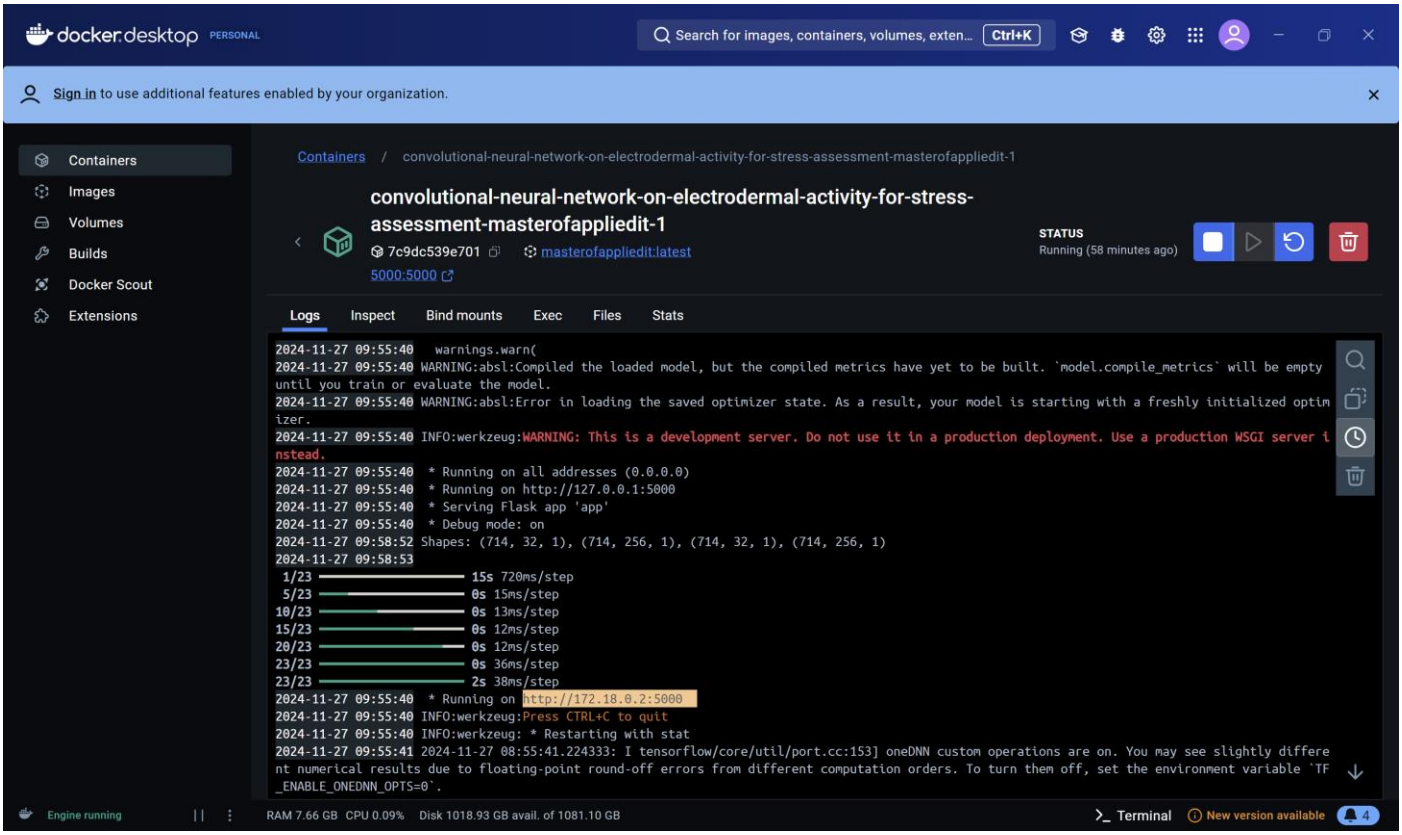
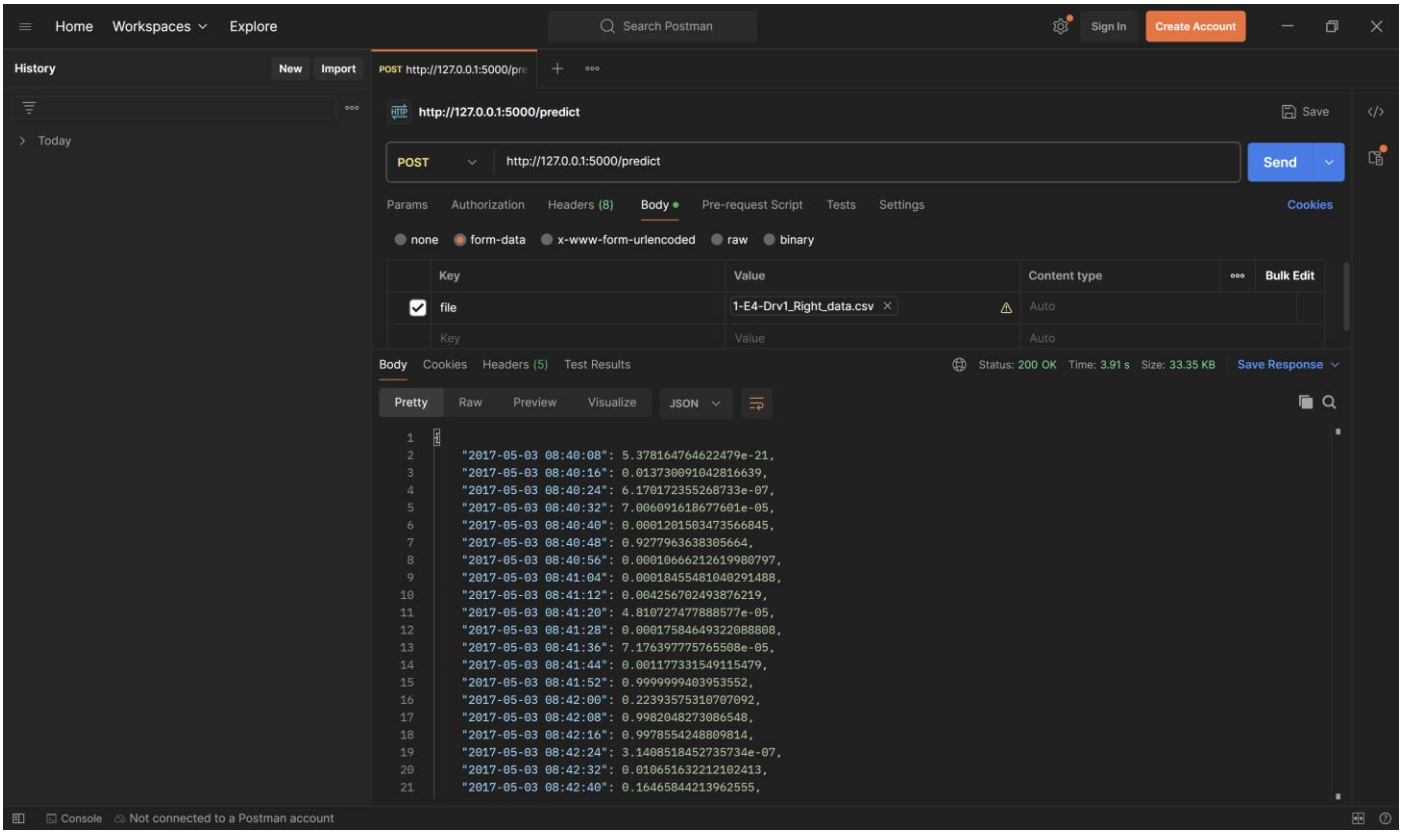


Figure 22: GitHub repository with the Docker package containing the latest model.

This process served as a temporary solution to showcase the deployment, since I was unable to produce a proper prototype of a model deployment.





Conclusion

In summary, the ability to reproduce experiments and effectively track changes over time is a critical requirement for any research project, especially in the context of machine learning. The current infrastructure, which lacks centralized tools for experiment tracking and version control, presents challenges in managing the complexity of workflows and ensuring the reproducibility of results. These challenges are particularly pronounced when working with multiple signal combinations, such as in stress detection models using physiological data like EDA, BVP, TEMP, and ACC. Without a robust system to track the variations in signal configurations, preprocessing steps, and model parameters, it becomes difficult to maintain consistency and validate findings.

By adopting a combination of DVC (Data Version Control) and CML (Continuous Machine Learning), along with Docker for Continuous Integration (CI), Continuous Testing (CT), and Continuous Deployment (CD), we have established a robust framework that directly addresses these challenges. DVC and CML provide an effective way to version data, and experiments, enabling the easy tracking of changes, reproducibility of experiments, and transparency in the research process. Docker's integration ensures that the model development and experimentation can be standardized and deployed across environments, streamlining collaboration and minimizing errors in different stages of the workflow.

This infrastructure significantly improves the ability to reproduce experiments, manage complex dependencies, and maintain consistency across different model configurations and signal combinations. The introduction of automated pipelines also reduces the risk of errors, enhances collaboration, and makes it easier to scale up experiments as the data and computational needs grow.

By implementing these tools, the research process is made more efficient and reliable, ensuring that future machine learning models, such as those for stress detection, are built on a solid foundation of reproducibility, transparency, and collaboration.

Learning Outcomes

1. Analysis

Perform analysis of infrastructure requirements, including manual data versioning, model tracking, and partial automation pipelines.

The analysis of the infrastructure requirements highlighted critical areas such as data versioning, model tracking, and partial automation of experiment tracking. The current setup relies on manual processes for managing dataset versions, parameters tracking and model registry which poses challenges for reproducibility and consistency. The evaluation included an in-depth examination of tools and methodologies for experiment tracking, emphasizing the importance of reproducibility and traceability in machine learning workflows. Additionally, considerations for security and compliance were reviewed to ensure the infrastructure meets necessary standards while supporting scalable and efficient research practices.

The analysis identified several key requirements for improving the infrastructure. **D**ata versioning is essential to ensure consistency and traceability of datasets throughout the machine learning lifecycle. Effective experiment tracking is needed to document configurations, parameters, and results systematically, enabling seamless comparison and reproducibility of experiments. Partial automation of pipelines is also required to reduce manual effort and minimize errors in the workflow. Additionally, **the** security and compliance standards **should be maintained when possible.**

2. Advise

Provide 2-3 recommendations for optimizing cloud infrastructure and implementing automation practices through tools like DVC, CML, and MLflow.

In reflecting on the infrastructure analysis for the machine learning project, I implemented **DVC** to enhance data and model versioning, ensuring reproducibility and traceability of experiments. This allowed for efficient management of data and models, making it easier to track changes and reproduce results. Additionally, I saw an opportunity to further optimize workflows by integrating **CML**, which would automate model training and evaluation pipelines, enabling continuous integration and faster iterations.

Furthermore, to meet **GDPR compliance** and enhance security, I integrated **logging within Azure Blob Storage**. This would ensure that data management practices align with required regulations, providing traceability and transparency for all data processing actions and experiment tracking

3. Design

Design an infrastructure architecture with a focus on model registry, manual data version control and machine learning lifecycle tracking.

The infrastructure architecture for the project was designed with a focus on data version control, model tracking, and experiment lifecycle management, with **DVC (Data Version Control)** as the central tool. Automation practices were considered, particularly through **CML (Continuous Machine Learning)**, to automate the training and evaluation processes, reducing manual intervention. The design was done by practices of DVC in their GitHub and applied to our context. Additionally, this gives an explainer of how DVC and CML interact within the development cycle.

4. Realisation

Implement the designed infrastructure, focusing on experiment tracking, model versioning, and partial automation using CML to deploy one ML model.

I created a guide-like documentation within this document to present my results. And posted images of results from various systems (local development, GitHub actions, Data version control studio, and Docker) to clarify some steps. Additionally, these guidelines can be used to help future work within the stress wearable project.

Unfortunately, as documented within results section, I could not fully implement a deployment of a registered model to docker due to a bug within DVC. I notified my stakeholders and mentors and managed to create a work-around, which is not optimal, but managed to showcase a similar result. And showcased that experiment tracking, data version control, traceability and model registry are met.

5. Manage & Control

Manage the infrastructure, model version control, and experiment tracking. This includes monitoring performance and ensuring the infrastructure supports continuous tracking and model updates.

The management and control of the infrastructure using DVC for version control and experiment tracking has been effective in ensuring traceability and reproducibility. DVC enables clear tracking of data, model versions, and configurations, supporting easy rollbacks and validation of updates. With the integration of CML, automated workflows have streamlined model training and evaluation, improved consistency and reducing manual intervention.

References

[1] T. Mboweni, T. Masombuka, and C. Dongmo, “A Systematic Review of Machine Learning DevOps,” in *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, Jul. 2022, pp. 1–6. doi: 10.1109/ICECET55527.2022.9872968.
[2] A. Knox, “Enterprise-Ready MLOps: Re-thinking the ML maturity model | CtiPath.” Accessed: Dec. 16, 2024. [Online]. Available: https://www.ctipath.com/articles/mlops/emlops-re-thinking-the-ml-maturity-model/
[3] P. Schmidt, A. Reiss, R. Duerichen, C. Marberger, and K. Van Laerhoven, “Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection,” in *Proceedings of the 20th ACM International Conference on Multimodal Interaction*, Boulder CO USA: ACM, Oct. 2018, pp. 400–408. doi: 10.1145/3242969.3242985.
[4] “MLflow | MLflow.” Accessed: Nov. 26, 2024. [Online]. Available: https://your-docusaurus-site.example.com/
[5] “Kubeflow,” Kubeflow. Accessed: Nov. 26, 2024. [Online]. Available: https://www.kubeflow.org/
[6] “Weights & Biases: The AI Developer Platform,” Weights & Biases. Accessed: Nov. 26, 2024. [Online]. Available: https://wandb.ai/site/
[7] “neptune.ai | The experiment tracker for foundation model training,” neptune.ai. Accessed: Nov. 26, 2024. [Online]. Available: https://neptune.ai/
[8] “DataChain | AI Data Management at Scale - Curate, Enrich, and Version Datasets,” DataChain. Accessed: Nov. 26, 2024. [Online]. Available: https://datachain.ai
[9] “Datasets and Dataset Versions | ClearML.” Accessed: Jan. 09, 2025. [Online]. Available: https://clear.ml/docs/latest/docs/hyperdatasets/dataset/
[10] “Open Source LLM Evaluation Platform,” Comet. Accessed: Jan. 09, 2025. [Online]. Available: https://www.comet.com/site/
[11] “Home Page,” Pachyderm. Accessed: Dec. 16, 2024. [Online]. Available: https://www.pachyderm.com/
[12] “Git Large File Storage,” Git Large File Storage. Accessed: Dec. 16, 2024. [Online]. Available: https://git-lfs.com/
[13] “TensorBoard,” TensorFlow. Accessed: Nov. 26, 2024. [Online]. Available: https://www.tensorflow.org/tensorboard
[14] “Installation,” Data Version Control · DVC. Accessed: Nov. 26, 2024. [Online]. Available: https://dvc.org/doc/studio/self-hosting/installation
[15] normesta, “Configure a lifecycle management policy - Azure Blob Storage.” Accessed: Dec. 05, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/lifecycle-management-policy-configure
[16] normesta, “Azure Storage-versleuteling voor inactieve gegevens.” Accessed: Dec. 04, 2024. [Online]. Available: https://learn.microsoft.com/nl-nl/azure/storage/common/storage-service-encryption
[17] pauljewellmsft, “Encrypt and decrypt blobs using Azure Key Vault - Azure Storage.” Accessed: Dec. 04, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-encrypt-decrypt-blobs-key-vault
[18] normesta, “Een minimale vereiste versie van Transport Layer Security (TLS) afdwingen voor binnenkomende aanvragen - Azure Storage.” Accessed: Dec. 04, 2024. [Online]. Available: https://learn.microsoft.com/nl-nl/azure/storage/common/transport-layer-security-configure-minimum-version