



Source: www.wallpaperflare.com licensed under creative commons zero (CC0)

MLOPS: A RETAIL USE CASE

Building auditable and reproducible ML infrastructure

TRUSTEQ GmbH
Hansaring 20
50670 Köln

contact@trusteq.de
www.trusteq.de

Ensuring reproducibility and versioning of ML models increases both: model reusability and auditability

Application teams often use multiple Machine Learning (ML) models shared between them and integrated into different pipelines. Still, many companies believe that modelling is a one-time task and sooner or later run into the issue of missing capabilities to reproduce own decisions.

ML modelling is usually part of a larger modelling process and introducing any change often infers that other dependent components can receive non-expected input or produce non-expected results.

While unlocking ML reproducibility is not for free, it helps to improve performance and reusability of models significantly. In a large diverse enterprise or in longer-lasting development streams, it reduces the risk of losing information.

At the same time auditability (in terms of going back in time) and providing the decision background of ML is ensured if reproducibility and versioning are fully implemented.

Cloud providers are constantly adding new features related to ML and Microsoft lists versioning & reproducibility as one of the top challenges to be solved within the MLOps process (<https://github.com/microsoft/MLOps>), but there are still a lot of areas that require integration of 3rd party tools, processes, and standards. One of the most challenging and common is still to ensure that any developed ML model can be reproduced or compared with older versions.

Against this backdrop we see five core requirements that need to be ensured to implement ML reproducibility measures successfully and to unlock their full potential:

- 1 Ensure reproducibility against the backdrop of changing data storage layers.
- 2 Enable model comparison in fast-paced model development.
- 3 Avoid redundant ML storage layer to ensure cost efficiency.
- 4 Analyse requirements for platform-independent model development.
- 5 Define minimal metadata requirements to make ML pipelines fully reproducible and implement ML metadata storage features.

In this paper we will discuss how these different requirements to ML reproducibility and related challenges can be approached.

As with our first paper on *"MLOps in the financial industry: building a cross-unit ML Cockpit to support your ML Governance"*, we will present relevant parts of our approach on how to implement reproducibility within the MLOps lifecycle based on a practical example for easier understanding of the theoretical concept and background. We will unfold our approach based on an exemplary analytics model implementation for retail enterprises on an Azure Machine Learning infrastructure.

What MLOps versioning & reproducibility can learn from DevOps principles and which differences emerge

The concepts of source code versioning, registering artifacts and automation is well known in the DevOps life cycle and can be seen as an Industry standard nowadays for software development.

So, on the first view, it seems obvious to take these principles as starting point when thinking on versioning and reproducibility for MLOps. However, there are **relevant differences between MLOps and DevOps especially** when it comes to complexity and requirements on the lifecycle reproducibility.

On Figure 1 the key differences between DevOps and MLOps related to reproducibility are summarized:

DevOps	MLOps
Code and pipeline versioning is mainly not related to data changes	Code and pipeline versioning is strongly related to data changes
Code and SW component reuse is usually deterministic	Feature engineering and model code usually must be fixed to behave in deterministic way
Component output is less dependent of package version upgrades	Same model can provide different results if used from different packages or versions
Standards for code versioning, commenting, tagging, and using changelog is well established and stable	Standards for ML model versioning and development tracking are still not broadly established and accepted
Reuse of SW components usually requires only configuration and seed data changes	Reuse of ML components might require additional tuning or transfer learning
Customer data is usually not embedded inside SW component	Customer data are often embedded inside ML components which creates additional risks
SW components do not degrade over time in terms of quality and precision	ML components running on same production infrastructure without any code changes can degrade / drift over time in terms of quality and precision (means we cannot easily reproduce historical results)

Figure 1 comparison of classic DevOps requirements on versioning & reproducibility with MLOps requirements

The reproducibility challenges listed above are well understood and cross-industry analysed. Going forward we will provide deep dives in a few selected of those challenges and discuss concepts how to approach them.

ML Data Versioning

ML data versioning is evolving since ML-based decisions depend on snapshots of data that are provided as input to specific model versions. Data versioning in MLOps assumes primarily storing snapshots of training (and optionally scoring) dataset to reproduce comparable results when evolving model parameters.

One core decision to take when implementing ML data versioning is to choose between:

- A. Utilizing versioning capabilities of storage or time-travel features of databases/warehouses
- B. Managing ML data versioning by a ML reproducibility process

Option A transfers a large part of the responsibility to the storage layer, it is easier to implement and provides a sufficient reproducibility level for small companies where all parts of business logic are tightly coupled. An issue with this approach lies in the fact that storage infrastructure is evolving. This means that input sources provided by queries, materialized views, storage partitions, API or links may vary (in field type, size, compression, extraction logic) when migrating to a new infrastructure and may limit reproducibility significantly.

Option B solves the issues of Option A at a cost of higher implementation, maintenance, and storage costs. For data versioning optimizing storage costs can make the biggest impact. By introducing incremental snapshots and defining multiple reproducibility levels (code, configuration, data environment or even evaluation level) enterprises can significantly reduce data versioning cost.

Denormalized Data Snapshot

When versioning ML model input data, usually only training and test dataset are stored. Since training and test dataset often contain small subset of columns which might be scaled and normalized, it is hard to interpret values and "debug" decisions. Interpreting model behaviour based on input data is especially important in early stages of production where model is expected to predict a lot of edge case scenarios. A solution to this can be implemented in two simplified ways:

- A. Versioning of all datasets and steps in feature engineering pipeline
- B. Taking snapshot of denormalized model input data

Option A is simpler to implement since it uses only already existing data formats but has very high cost of storage and requires multi-step inspection.

Option B creates as snapshot containing original model train/test columns, but also stores all other custom columns that are relevant for quick data inspection.

Comparing models and results

Re-running of a ML pipeline is required when there are changes on:





 Training data	 Model version	 Model parameters	 Infrastructure
e.g. wrong rows were removed from input dataset	e.g. model is changed with same input dataset	e.g. number of grid search combinations is expanded	e.g. pipeline input directory or package versions have changed

Figure 2 potential changes leading to a re-run of a ML pipeline

In all these changes we are interested how they impact results and metrics compared to previous runs.

Our solution approach includes a parallel tracking of ML model actions enriched by information like:

- execution environment,
- parameters,
- scoring metrics,
- artifacts (e.g. image of ROC curve),
- input data shape & distribution,

Industry use case: store assortment planning

For category managers planning store assortments for upcoming periods, it is a common challenge, that they are facing a higher number of available Stock Keeping Units (SKU) than shelve space in the store. Hence an important question is, if some of the SKUs can be replaced to increase margin and sales amounts by introducing new products or by unlocking upselling potentials.

An optimization algorithm might first try to create product groups and identify the substitutability of SKUs before trying to predict potential earnings by performing assortment changes.

The assortment optimization pipeline is composed of data storage, databases, ETL components, ML models, non-ML optimization algorithms, parameters that are manually tuned, artifact storage and much more components for productive provisioning.

In the example on figure 3 and 4 we show ML pipeline components relevant for demonstrating challenges related to MLOps reproducibility. The discussed ML pipeline consists of:

- 2 input datasets: a global product substitutability score and the product sales history
- 2 ML models:
 - Model 1: product substitutability clustering (agglomerative hierarchical clustering)
 - Model 2: product group sales prediction model (recursive autoregressive forecaster)
- 3 ML model execution components:
 - Training and Tuning of Model 1
 - Training, Tuning and Scoring of Model 2
 - Batch Prediction using Model 2
- 2 pre- or post-processing components:
 - Hierarchy denormalization
 - Dataset enrichment

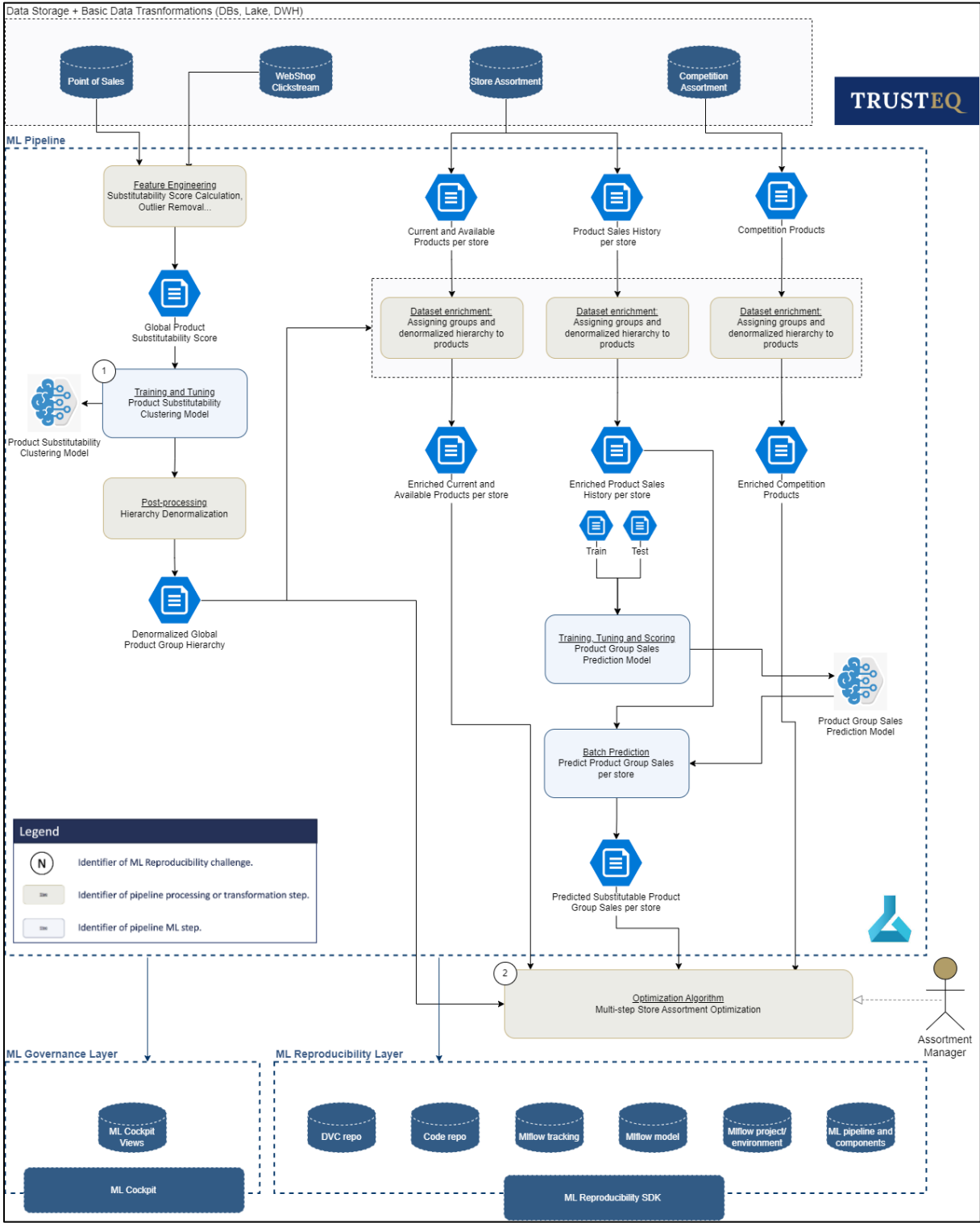


Figure 3 Data flow diagram showing ML-related excerpt of store assortment planning optimization pipeline.

To elaborate the benefit of reproducibility we bring in two sample challenges that might happen during the operation of the ML pipeline (marked by numbers on Figure 3).

The first challenge is related to categories of **ML data versioning** and **comparing models and results**.

For different stores in a retail enterprise different version of ML pipelines were executed to provide optimal assortments. After analysing feedback from production, it was noticed that the „Minimum number of transactions per product“ filter for the global product substitutability score dataset was too low. Since different stores used different ML pipeline versions to produce current assortment optimization, regional manager would like to make sure that current assortment plans are reproducible, would like to create new assortment plans by increasing filters in different model versions and compare optimization results. Also, in future, regional manager would like to share models between stores and compare outcomes.

The second challenge is related to **ML data versioning** and **denormalized data snapshot**.

In 2021 the assortment optimization algorithm suggested to replace product A with product B for summer season in all stores. In 2022 the same assortment optimization algorithm suggested to replace product A with product B for summer season only in 30% of the stores. In between the DWH layer has been changed and the „Product Sales History“ materialized view on which 2021 sales prediction model was trained is no longer available. It is now impossible to reproduce the old results and hence it becomes much more difficult to analyse why suddenly the same product should not be replaced everywhere (having in mind that parameters for none of the 2 ML models have changed).

To solve both mentioned challenges engineering teams need to invest a lot of effort and still must accept some previous decisions as a block box. A solution would be to implement full ML reproducibility (on top of the ML cockpit solution proposed in previous paper).

Here is how to integrate existing open-source ML tools and cloud native services to do exactly that.

Implementing ML reproducibility

Current cloud ML services already provide a lot of features related to reproducibility. Depending on the existing enterprise infrastructure, a stack (e. g. on Microsoft Azure) could consist of:

- Azure Storage Account
- Azure Machine Learning utilizing following services:
 - Notebook, scikit learn package, Python Azure ML SDK v2
 - Data Asset and Data Store
 - Mlflow integration (tracking, model, project)
 - Model registry
 - Pipeline and components
 - Endpoints
- DVC
- GitHub

Even Azure Storage Accounts and Data Assets provide versioning support, but we opt for **DVC** integration due to a git-like low-UI data management option and a git log feature that provides a more transparent data changelog tracking. DVC (Data Version Control) is an open-source git-based data science tool for versioning of machine learning development. It allows tracking of ML experiments, code, and data changes in a standard git repository. In our sample case we use only the features related to dataset change tracking.

The key idea behind this sample ML reproducibility implementation is to build it via the integration of different tools with a central **Mlflow Tracking** service in place. Mlflow is an open-source platform for managing the end-to-end machine learning lifecycle. As well as DVC, it has many different concepts and components, but here we integrate only with Mlflow Tracking, Mlflow Model and Mlflow Projects components. On image 3 it is visible that Mlflow tracks only the pipeline components related to ML model training, tuning, scoring or prediction in our case. Also, Mlflow enables the storing of standardised model and execution environment artifacts which can run on different infrastructure (e.g. on AWS). Other components are tracked only with Git. Exceptions are input and output datasets of ML components utilizing Mlflow Tracking service. These datasets are versioned by DVC and tracked by Mlflow to allow full ML component reproducibility.

The **Mlflow Tracking** component is an API and UI for logging parameters, code versions, metrics, and output files when running your machine learning code and for later visualizing the results. Mlflow Tracking lets you log and query experiments using Python, REST, R API, and Java API APIs. By default `mlflow.autoLog()` allows rich logging of metrics, parameters and models into artifact objects (for all main ML libraries). On top of that, Mlflow Tracking organizes everything into an experiment-run workflow and lets you add different custom artifacts (e.g., grid search, image, ROC curve, ...). Extended artifact details are shown on figure 4 in "Object 1: Mlflow Tracking" section.

GitHub can be used for versioning of code and as DVC repository, while **Azure Storage Account** can provide storage infrastructure for files and tabular datasets.

All described open-source tools can be integrated together within the **Azure Machine Learning** platform. The platform integrates Mlflow standard in all aspects which makes the whole architecture easily reusable in a cross-cloud architecture.

But you need to keep in mind, there are components that prevent seamless cloud migration (e. g. Azure ML pipeline and Azure ML components). If "cross-cloud" is an enterprise requirement, we would suggest using Prefect or some other not cloud native framework for building end to end ML pipelines. A very good article that highlights this approach was published by Keshav Singh (<https://medium.com/@masterkeshav/scalable-ml-training-workflow-management-with-prefect-cloud-mlflow-azure-b6abf1491c8f>).

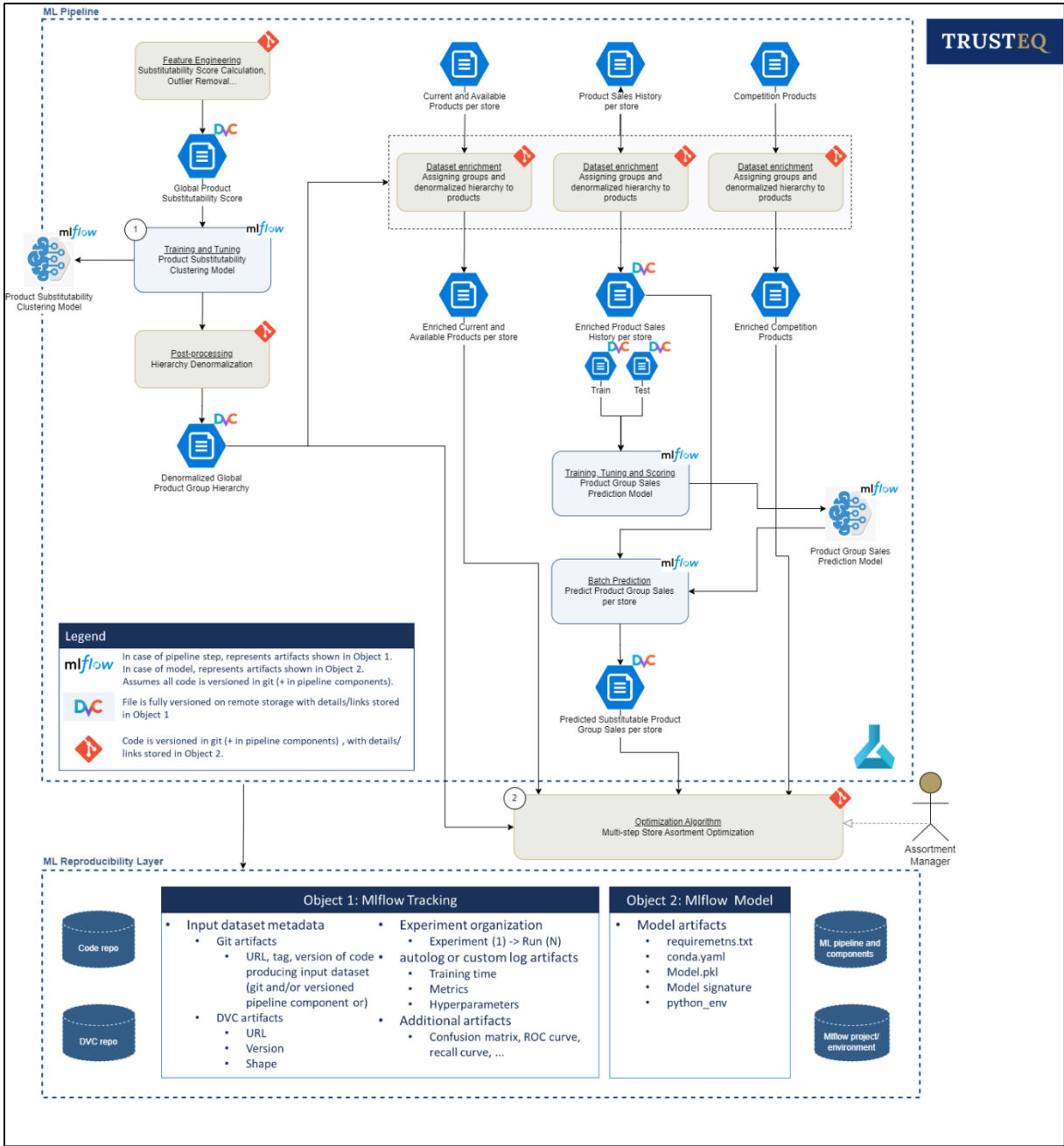


Figure 4 Data flow diagram showing integration with Mlflow and DVC

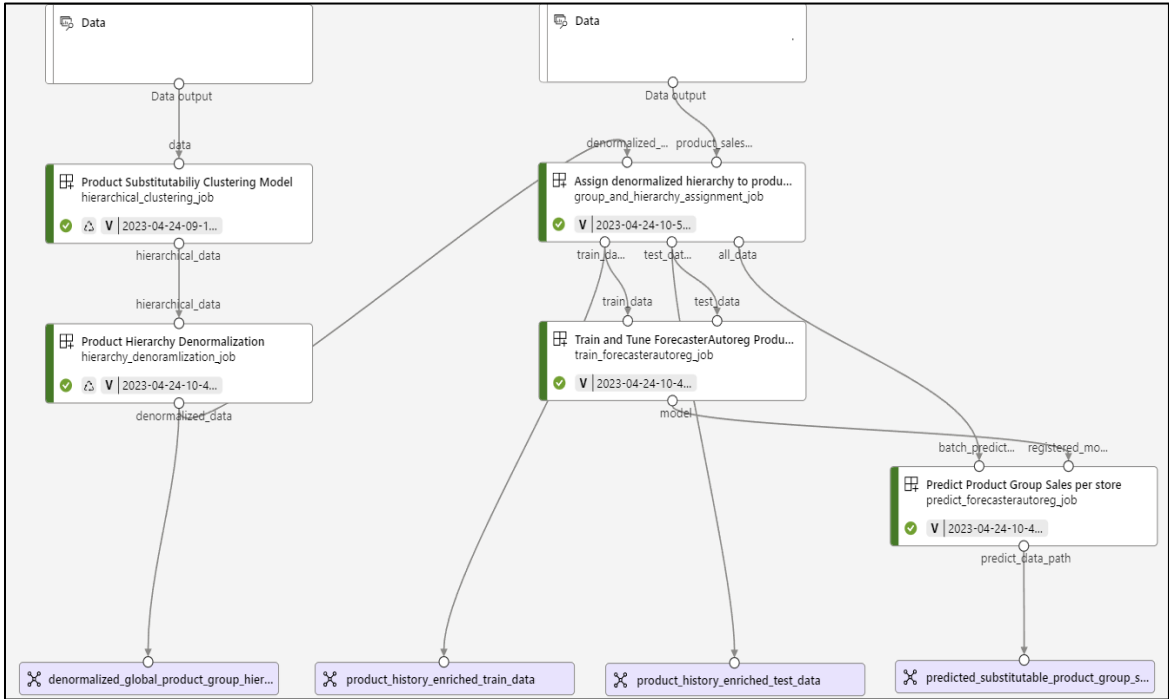


Figure 5 Azure ML pipeline diagram showing core part related to ML model training and batch scoring.

Figure 5 displays the visualization of the discussed Azure Machine Learning pipeline. Custom components defined by python script (logic) and .yaml (interface) files can be used and for use cases like the described retail case, it could be sufficient to enable manual triggering. Output datasets produced by this pipeline can be utilized in a later assortment optimization step which is not necessarily ML-driven.

All cloud providers are offering similar ML platforms but to really enable cross-cloud ML reproducibility, there are established 3-rd party frameworks covering the same ML lifecycle workflows as cloud native services. Some of them are defining reproducibility level based on objective KPIs for its assessment. From business perspective the decision on the level of reproducibility is dependent on the trade-off between implementation & operating costs on the one side and the risk of the impact if information is lost during development or production of the operation of the ML model on the other.

One sample framework that implements a similar ML reproducibility workflow and rates ML reproducibility levels is DagsHub. DagsHub ML reproducibility levels are shown on image 6.

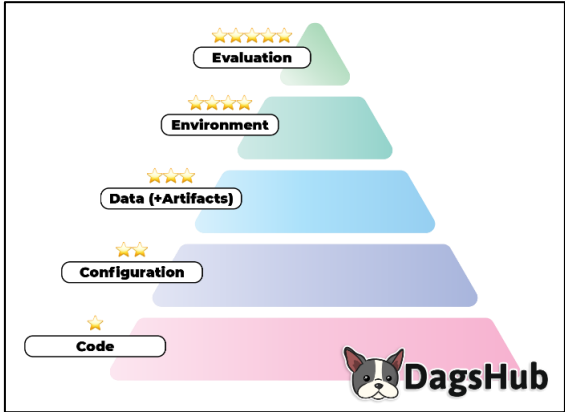


Figure 6 ML reproducibility levels by DagsHub

Conclusion

In this whitepaper we walked through some main challenges in the enterprise ML model reproducibility process. Every ML lifecycle framework describes and breaks down reproducibility challenges in different ways, so we tried to present a small subset of challenges based on a concrete store assortment optimization use case.

A central aspect for ensuring reproducibility is the integration of ML components using Mlflow standards (artifacts, APIs, experiment structure). We presented how Mlflow Tracking (main Mlflow component) can be customized and integrated with DVC and GitHub to provide full reproducibility on any infrastructure.

We presented experiences and suggestions how a concrete Mlflow integration can be set up as a part of a ML pipeline on top of the Azure Machine Learning platform. The aim was to implement full ML reproducibility within the ML pipeline and to solve two sample challenges to reduce the risk of losing any valuable ML-related information.

Higher ML reproducibility levels are still considered by most enterprises as a "luxury" feature and are often identified as low priority. Our strong suggestion is to assess the business risk of not having the capability to back up or not to be able to interpret historical data-driven decisions. This becomes even more important when your business is not located in retail, but in stronger regulated industries like banking, insurance or health care.

All in all, ML reproducibility levels and tools like Mlflow should be at least well understood and considered in every phase of ML model development and when you define your overall strategy of ML governance. Combined with a ML Cockpit these two levers are already big leaps to bring transparency and reusability into the realm of your company's machine learning models.

For more details get in contact with us



Ante Gojsalic
Lead Data Scientist
ante.gojsalic@trusteq.de



Nils Gilles
Head of Data & Analytics
nils.gilles@trusteq.de