PROCEEDINGS OF SCIENCE

# MLOps for Scientific Applications: Building Reliable, Reproducible, and Transparent AI Workflows

**Luca Clissa,**[a,b,*] **Leonardo Plini**[c,d] **and Daniele Bonacorsi**[a,b]

[a] *Department of Physics and Astronomy "Augusto Righi", University of Bologna*

[b] *Istituto Nazionale di Fisica Nucleare, Sezione di Bologna*

[c] *Department of Computer, Control and Management Engineering "Antonio Ruberti" Sapienza University of Rome*

[d] *Istituto Nazionale di Fisica Nucleare, Laboratori Nazionali di Frascati*

*E-mail:* luca.clissa2@unibo.it

Modern scientific workflows rely increasingly on machine-learning (ML) models. To enable reliable results, these models must be trained and deployed with high standards for reproducibility, transparency, and traceability. However, many scientific ML pipelines remain ad hoc, with poorly documented data processing steps, opaque hyperparameter tuning and insufficient provenance. Machine Learning Operations (MLOps) promises to close this gap by extending the automation and continuous integration ideas of DevOps to the full ML lifecycle. This paper reviews the state of the art in MLOps for scientific applications, focusing on high-energy physics (HEP) and related data-intensive sciences. We synthesise a general MLOps pipeline, survey best practices and tooling across each stage, and analyse how these principles apply when deployment consists of batch inference and offline reprocessing rather than online serving. A case study based on the `higgsML-uncertainty-mlops` repository illustrates how Weights & Biases (W&B) can track experiments, version data, capture lineage and enable reproducible comparisons. We conclude by arguing for community-driven benchmark suites and registries of scientific ML pipelines, analogous to GLUE in natural language processing, to accelerate reproducible ML research in HEP and beyond.

*International Symposium on Grids and Clouds (ISGC2026)*
*15-20 March 2026*
*Academia Sinica, Taipei, Taiwan*

[*] Speaker

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0) All rights for text and data mining, AI training, and similar technologies for commercial purposes, are reserved. ISSN 1824-8039 . Published by SISSA Medialab.



PoS(ISGC2026)025

## 1. Introduction

Machine Learning (ML) has become a central tool for extracting insight from modern scientific experiments. This is particularly true for domains like High-Energy Physics (HEP), where smart and efficient analysis solutions are crucial to cope with the sheer quantity of data to be processed [6]. Such data volumes demand scalable analysis pipelines, yet the development, deployment and maintenance of ML models remain labour-intensive and error-prone. Ad hoc scripts and undocumented decision paths hinder reproducibility and impede collaboration. These challenges are symptomatic of a broader reproducibility crisis in ML-based science: subtle mistakes in data processing, evaluation or code can lead to optimistic but irreproducible results [3].

Machine Learning Operations (MLOps) offers a principled solution. Borrowing concepts from DevOps, MLOps advocates for automation, continuous integration/continuous deployment (CI/CD), reproducibility, versioning, monitoring and iterative feedback across the ML lifecycle [2]. In contrast to classical software, ML systems must handle stochastic training, evolving datasets and complex data dependencies. Hidden technical debt arises from entangled models, brittle preprocessing and feedback loops [1]. MLOps aims to mitigate this debt by treating data, code and models as first-class artefacts whose lineage is recorded and managed.

This paper explores MLOps through the lens of scientific workflows. We define the stages of an MLOps pipeline, review best practices and map available tools. To make the discussion concrete, we analyse a representative HEP pipeline implemented in the open source repository `higgsML-uncertainty-mlops`. The pipeline illustrates how experiment tracking and data artefact management, can enable reproducible analysis, transparent comparisons and collaborative development. Our contributions are the following:

- an overview of MLOps and its relevance to scientific research, highlighting differences with DevOps and the scale of HEP data.
- a structured MLOps pipeline covering data processing, modelling, deployment, monitoring and feedback, together with recommended best practices.
- a survey of current MLOps tools and their coverage of the pipeline stages.
- a case study demonstrating how W&B enables experiment tracking, artefact versioning and lineage in a HEP-inspired ML workflow.
- a forward-looking discussion on adapting MLOps to scientific deployments and on building community benchmarks and registries for scientific ML.

## 2. Background: MLOps and Scientific Workflows

### 2.1 From DevOps to MLOps

DevOps emerged to streamline software development by integrating continuous integration, automated testing, infrastructure as code and monitoring [2]. Applying these principles to ML, however, requires additional considerations. Models depend on data whose distribution may drift, training is stochastic, code changes and data updates are intertwined, and evaluation requires



PoS(ISGC2026)025

careful statistical analysis. Kreuzberger *et al.* characterise MLOps as encompassing nine core principles, including automation of the ML lifecycle, reproducibility of data and code, versioning of datasets and models, collaborative experiment tracking, monitoring of models in production, and a continuous feedback loop for model improvement [2]. These principles aim to reduce friction and technical debt throughout the ML lifecycle.

### 2.2 Challenges in Scientific ML

Scientific experiments differ from commercial ML applications in several respects. First, datasets are often orders of magnitude larger or more complex than typical industry benchmarks; HL-LHC experiments will generate hundreds of billions of events [6]. Second, reproducibility is paramount: results must be traceable years after publication. Third, deployment rarely involves real-time online serving; instead, models may be applied in batch to reprocess years of data or to simulate detector conditions. The hidden technical debt of ML systems is particularly acute in this context: unrecorded preprocessing steps, evolving calibration constants, and tangled code lead to "Changing Anything Changes Everything" (CACE) phenomena [1].

The reproducibility crisis in ML-based science further motivates structured practices. Studies have highlighted how data leakage, improper evaluation and opaque code can lead to irreproducible findings [3]. Data-centric frameworks argue that reproducibility must capture not only the final results but also the dynamic development process, including data selection, feature engineering, hyperparameter choices and human decision points [4]. Huang *et al.* propose six structured artefact types – datasets, features, workflows, executions, assets and controlled vocabularies–to preserve provenance across the ML lifecycle [4]. These ideas resonate with the need to manage large, evolving datasets and complex analysis chains in HEP.

### 2.3 Examples of MLOps in Scientific Domains

Recent efforts demonstrate the applicability of MLOps beyond commercial contexts. Testi *et al.* introduce the Cell Counting MLOps (CC-MLOps) framework for automated quantification of neuronal cells [5]. Their pipeline integrates data access, preprocessing, model training, monitoring, explainability and sustainability to deliver reliable biomedical analyses. The success of such domain-specific pipelines underscores the need for general MLOps practices tailored to scientific requirements.

## 3. Structured MLOps Pipeline

Figure 1 sketches a generic MLOps pipeline suitable for scientific workflows. The pipeline is divided into seven stages: data acquisition and curation; data processing and splitting; feature engineering; model training; model evaluation and calibration; deployment; and monitoring with iterative feedback. Each stage must be orchestrated to ensure reproducibility, provenance and scalability.



PoS(ISGC2026)025

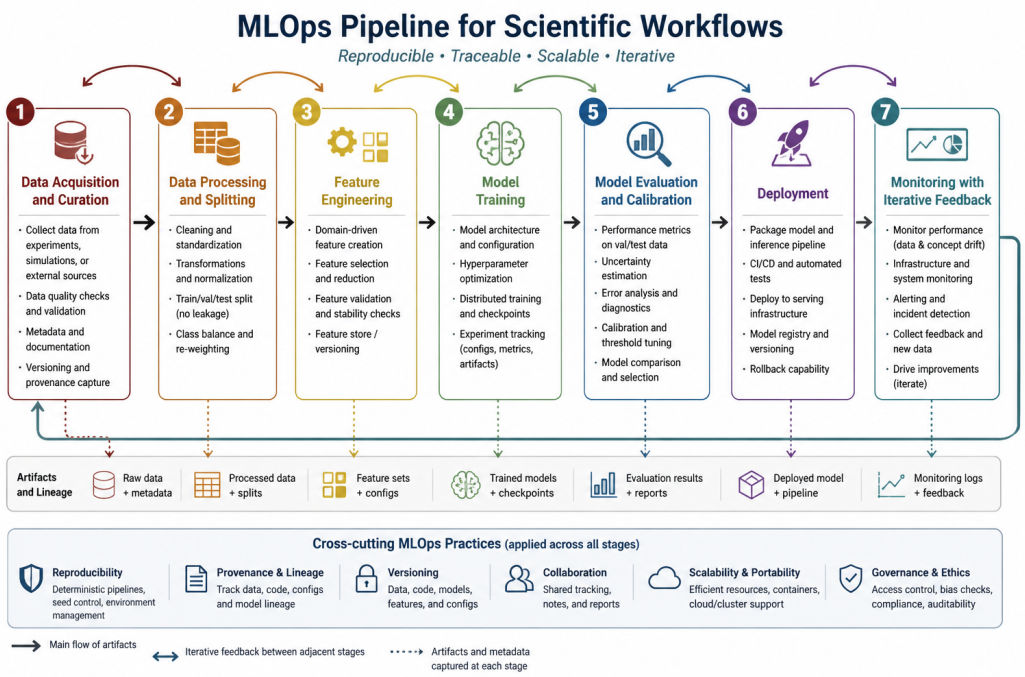


**Figure 1:** Conceptual MLOps pipeline illustrating the stages from data ingestion through processing, modelling, deployment and monitoring. The arrows indicate the flow of artefacts and feedback.

### 3.1 Data Acquisition and Curation

Raw data collected from detectors or instruments must be ingested into a managed environment. Best practices include capturing raw datasets as immutable artefacts, recording metadata such as acquisition time, detector configuration and calibration constants, and ensuring data quality through validation tests. In the HL-LHC context, the sheer volume of data necessitates scalable storage and parallel ingestion. Tools like W&B Artefacts allow datasets to be versioned and shared across runs [8].

### 3.2 Data Processing and Splitting

Preprocessing transforms raw measurements into analysis-ready features. This stage often includes filtering, cleaning, normalisation and feature construction. To preserve reproducibility, preprocessing code should be version controlled and parameterised; data splits (train/validation/test) should be deterministic and logged as artefacts. In our case study, separate short-lived runs are used to log raw and split data, establishing a lineage chain.

### 3.3 Feature Engineering

Scientific ML frequently relies on physics-motivated features or complex simulations. Feature engineering should be automated and accompanied by exploratory data analysis (EDA) to validate distributions. Visualisations of feature distributions can be logged as images or tables in experiment trackers such as W&B.



PoS(ISGC2026)025

### 3.4 Model Training

The core stage involves selecting an algorithm, defining loss functions and optimisation strategies, and training the model on labelled data. Reproducibility requires fixing random seeds, logging hyperparameters and code versions, and tracking training metrics. Automated hyperparameter search (e.g., Bayesian optimisation, grid or random search) can be orchestrated through sweeps [9]. W&B Sweeps support defining search spaces, launching agents and visualising results across multiple runs.

### 3.5 Evaluation and Calibration

Model evaluation should be performed on independent test sets with metrics appropriate for the task (e.g., area under the ROC curve, precision-recall, calibration error). For uncertainty-aware models, calibration methods such as Platt scaling or conformal predictions are employed. Log metrics and confusion matrices to the experiment tracker for comparison across runs. Calibration plots and reliability diagrams can be logged as images or interactive tables.

### 3.6 Deployment

In scientific settings, deployment often means exporting models for offline batch inference or integrating them into analysis frameworks rather than serving online predictions. Nonetheless, deployment requires packaging models and preprocessing code in reproducible containers, registering model artefacts, and documenting dependencies. Tools like BentoML, Seldon or KServe provide serving capabilities; for offline usage, storing model artefacts in a registry and providing reproducible notebooks suffices. W&B Artefacts can store models along with metadata [8].

### 3.7 Monitoring and Iterative Feedback

Even without real-time serving, monitoring remains important. Models should be periodically re-evaluated as new data become available or as detector conditions change. Drift detection tools (e.g., Evidently, WhyLogs) can flag shifts in feature distributions. Feedback loops allow analysts to retrain models with updated data and hyperparameters. Logging and alerting frameworks should capture metrics over time and notify stakeholders when performance degrades.

## 4. Best Practices Across Pipeline Stages

Drawing on the literature and industry practices, we summarise recommended best practices for each pipeline stage.

**Data management.** Treat raw data and processed datasets as immutable artefacts. Version datasets using data registries (e.g., W&B Artefacts, DVC, Pachyderm). Record provenance metadata, including acquisition conditions and preprocessing parameters. Use schema validation tools such as Great Expectations to enforce data quality.

**Code and environment.** Store all code under version control. Use containerisation (Docker/Singularity) and environment specifications (e.g., Conda) to capture dependencies. Record the commit hash and environment details in every run [10].



PoS(ISGC2026)025

**Experiment tracking.** Initialise a unique run for each experiment using a tracking tool. W&B recommends using a `with wandb.init()` context to ensure that the run is properly finalised [7]. Log hyperparameters, metrics, losses and artefacts throughout training. Use consistent naming conventions for metrics and group runs with tags for comparability.

**Hyperparameter tuning.** Use sweep frameworks (W&B Sweeps, Optuna, Ray Tune) to explore hyperparameter spaces. Define search strategies, metrics to optimise and early stopping criteria. Ensure that each sweep run logs its configuration and results.

**Model versioning and registry.** Register trained models as artefacts with semantic versioning. Include metadata such as training data version, hyperparameters and performance metrics. Use registries (W&B, MLflow, TensorFlow Hub, Hugging Face Hub) to store and query models. Avoid overwriting existing models; instead create new versions when retraining.

**Evaluation and calibration.** Use held-out test sets; avoid peeking into test data during hyperparameter tuning. When calibrating uncertainties, document the calibration method and log reliability curves. Compare models using consistent metrics across runs.

**Deployment and monitoring.** Package models and preprocessing code into reproducible containers. For batch inference, document the environment and input/output formats. Monitor models for drift or decay; reprocess and retrain as needed. Use alerting mechanisms to notify stakeholders when performance falls below thresholds.

## 5. MLOps Tooling Landscape

Table 1 summarises popular MLOps tools and the pipeline stages they support. We focus on open-source or widely used services.

The table emphasises that no single tool covers all stages. Full pipelines often combine data versioning (DVC or W&B Artefacts), experiment tracking (W&B, MLflow, Neptune), orchestration (Kubeflow, Airflow), deployment (BentoML, KServe) and monitoring (Evidently). Selecting tools depends on requirements, domain constraints, and the scale of data.

## 6. Case Study: A HEP-Inspired Pipeline with W&B

To illustrate the concrete application of MLOps principles, we examine the open-source repository `higgsML-uncertainty-mlops`. The project implements a classification and regression pipeline for HEP data, incorporating conformal prediction and uncertainty quantification. Its structure is modular and designed around MLOps principles.

### 6.1 Repository Overview

The repository is organised into Python packages for data loading, preprocessing, model training, calibration, evaluation and MLOps utilities. A top-level script `scripts/run_train.py` reads a YAML configuration file, instantiates a run context, and orchestrates training, calibration and evaluation [10]. Key MLOps components include:



PoS(ISGC2026)025

**Table 1:** Survey of common MLOps tools and their coverage across pipeline stages. Data: data management and versioning; Tracking: experiment tracking; Registry: model artefact registry; Orchestration: pipeline definition and scheduling; Deployment: serving models; Monitoring: model and data drift monitoring. A filled circle (•) indicates strong support for the corresponding stage; a dash (-) indicates little or no support.

| Tool | Data | Tracking | Registry | Orchestration | Deployment | Monitoring |
|---|---|---|---|---|---|---|
| Weights & Biases | • | • | • | - | - | • |
| MLflow | • | • | • | - | - | - |
| DVC | • | - | - | - | - | - |
| Kubeflow | - | - | - | • | • | - |
| Airflow/Prefect | - | - | - | • | - | - |
| ClearML | • | • | • | - | - | - |
| Neptune | - | • | - | - | - | - |
| Comet | - | • | • | - | - | - |
| BentoML/Seldon/KServe | - | - | • | - | • | - |
| Evidently/WhyLogs | - | - | - | - | - | • |
| Feast (Feature Store) | • | - | - | - | - | - |
| GitHub Actions | - | - | - | • | - | - |
| Hugging Face Hub | - | - | • | - | - | - |

- **Tracker** (`src/conformal_predictions/mlops/tracker.py`). A wrapper around W&B that initialises runs (optionally using `wandb.init`), logs metrics via `log` and `log_dict`, records images and tables, and writes metrics to a local file [10]. The tracker provides methods to start, finish and resume runs; when finishing it registers artefacts and appends run metadata to an index [10].

- **Artifacts** (`mlops/artifacts.py`). Functions for creating or consuming W&B artefacts: `log_or_use_data_artifact` uploads datasets or fetches existing versions; `log_raw_data_run` and `log_split_data_run` create short-lived runs to log raw and split datasets, enabling data lineage [10]. `log_model_artifact` uploads trained model files as artefacts [10].

- **Logging keys** (`mlops/log_keys.py`). Defines constants and helper functions to build hierarchical keys for metrics (e.g., `EDA/Feature/Mean`, `Evaluation/Calibration/Error`), ensuring consistent naming across runs [10].

- **Run context** (`mlops/run_context.py`). A dataclass capturing metadata such as run ID, timestamp, configuration, dataset name, Git commit and output directory [10]. It provides methods to save manifest files and register artefact manifests [10].

- **Trainer** (`src/conformal_predictions/training/trainer.py`). Orchestrates data loading, model fitting, calibration and evaluation. It logs metrics to W&B at each epoch and evaluation step and generates plots using the tracker [10]. It calls `use_data_artifact` to fetch datasets and logs EDA plots and evaluation artefacts [10].



PoS(ISGC2026)025

**Table 2:** Mapping of key repository components to MLOps pipeline stages and W&B features.

| Component | Pipeline stage | MLOps function | W&B usage |
|---|---|---|---|
| `run_train.py` | Orchestration | CLI entrypoint; parses config and triggers runs | Config stored in run context; optional W&B initiation |
| `mlops/tracker.py` | Tracking | Start/finish runs; log metrics, images, tables | `wandb.init`, `run.log`, `log_artifact` [10] |
| `mlops/artifacts.py` | Data & model versioning | Create/use data and model artefacts; lineage | `wandb.Artifact`, `run.log_artifact`, `run.use_artifact` [10] |
| `mlops/log_keys.py` | Tracking | Define metric keys | n/a (naming convention) |
| `mlops/run_context.py` | Metadata management | Capture run metadata and manifest | Write manifest files [10] |
| `training/trainer.py` | Training/Evaluation | Training loop, calibration, evaluation | Logs metrics, images and tables [10] |

### 6.2 Experiment Tracking and Data Lineage

The pipeline uses a *Tracker* class to encapsulate experiment tracking. When W&B is enabled, a call to `wandb.init` starts a run and binds it to a project. Within this context, the tracker logs metrics via `log` or `log_dict`, capturing losses, validation scores and calibration errors at each epoch. Images (e.g., feature histograms) are recorded via `log_image`, while tables summarising event counts or evaluation statistics are logged via `log_table`. At the end of a run the tracker registers data and model artefacts and writes a manifest with run metadata [10]. Runs are also recorded locally, enabling offline comparison when W&B is disabled.

Data lineage is established through helper functions in `mlops/artifacts.py`. The function `log_or_use_data_artifact` checks whether a dataset with a given name already exists in the project; if not, it creates a new artefact of type "dataset", adds the files and logs it [10]. To capture raw data and split data separately, short runs `log_raw_data_run` and `log_split_data_run` create minimal runs that log the raw dataset and the train/validation/test splits [10]. This design yields an explicit lineage graph connecting raw data, derived splits and downstream model artefacts. The model artefact itself is logged with `log_model_artifact`, which packages the trained network weights and accompanying metadata into a versioned artefact [10].

### 6.3 Training, Calibration and Evaluation

The `Trainer` class handles the core ML workflow. Its `fit` method iterates over epochs, trains the model, computes validation metrics, and logs them via the tracker using hierarchical keys (e.g., `Training/Loss`, `Validation/Accuracy`) [10]. Hyperparameters and model configurations are



PoS(ISGC2026)025

included in the run's configuration; random seeds are fixed to ensure reproducibility. After training, the `calibrate` method applies conformal prediction or other calibration methods, logging calibration curves and expected coverage [10]. The `evaluate` method then assesses the calibrated model on a held-out test set, logging metrics and producing plots such as ROC curves or reliability diagrams [10]. These artefacts are stored both locally and as W&B artefacts, ensuring reproducibility and comparability across runs.

### 6.4 Configuration and Reproducibility

Configuration files (YAML) define dataset paths, model hyperparameters, training options, calibration methods and evaluation settings. The `tracking` section specifies whether W&B should be enabled and includes parameters such as project name, entity and tags. When W&B logging is disabled, the pipeline still records metrics and artefacts locally, enabling offline experimentation. The `run_context` stores the configuration snapshot, dataset name and git commit hash [10], ensuring that each run can be reproduced.

### 6.5 Strengths and Limitations

This case study demonstrates that MLOps principles can be applied to complex scientific analyses. The pipeline separates concerns between data management, training logic and experiment tracking; establishes explicit data lineage; and uses W&B to log metrics, artefacts and dashboards. The modular design facilitates reuse and comparison across experiments. However, the pipeline still relies on manual configuration of training and calibration; automatic orchestration and deployment are not addressed. Moreover, integration with external feature stores or monitoring systems is absent. Nevertheless, the example underscores how accessible tools such as W&B can bridge the gap between research and production quality workflows.

## 7. Weights & Biases for Scientific MLOps

Weights & Biases (W&B) is an experiment tracking platform widely adopted in both industry and research. Its core abstraction is the *run*. A run is initialised with `wandb.init`, which starts tracking an experiment within a specified project and entity [7]. W&B recommends wrapping initialisation in a `with` block so that the run is properly finalised when the block exits. Within the run, users log scalar metrics, media (images, audio, video), tables and custom objects via `run.log` or the helper functions available in the W&B SDK.

### 7.1 Artefacts and Versioning

Artefacts provide first-class versioning for files and directories. W&B's documentation explains that artefacts can track datasets, models or other files, and can be used as inputs and outputs of runs [8]. Creating an artefact requires four steps: initialise a run, create a `wandb.Artifact` object with a name and type, add one or more files to the artefact, and log the artefact with `run.log_artifact` [8]. To consume an artefact in a downstream run, call `run.use_artifact` with the artefact name; the returned object can be downloaded to local storage [8]. The pipeline described in Table 2 uses these APIs to version raw datasets, data splits and models.



PoS(ISGC2026)025

### 7.2 Experiment Tracking and Metrics

During training, metrics are logged at regular intervals. A typical pattern is:

```python
with wandb.init(project="my-project", config=config) as run:
    for epoch in range(num_epochs):
        train_loss, val_loss = train_one_epoch(...)
        run.log({
            "Training/Loss": train_loss,
            "Validation/Loss": val_loss
        }, step=epoch)

    # Log a model file as an artefact
    artifact = wandb.Artifact("model", type="model")
    artifact.add_file("model.pt")
    run.log_artifact(artifact)

    # Consume a dataset artefact
    data_artifact = run.use_artifact("my_dataset:latest")
    data_dir = data_artifact.download()
```

**Listing 1:** Example W&B instrumentation for experiment tracking and artefact lineage.

Here the `config` dictionary is stored in the run, making hyperparameters accessible in the UI. The `step` parameter allows metrics to be plotted against epochs. Additional charts (e.g., histograms, scatter plots) can be logged via W&B Tables. Reports and dashboards can summarise results and share them with collaborators.

### 7.3 Hyperparameter Sweeps

W&B Sweeps automate hyperparameter search. A sweep is defined in YAML or as a Python dictionary specifying the search strategy (grid, random, Bayesian), the parameters to explore, and the metric to optimise. The documentation notes that sweeps support popular methods such as Bayesian, grid and random search and can scale across multiple machines [9]. Sweeps can be launched via the CLI with `wandb sweep <config>` and agents started with `wandb agent <sweep-id>`. In scientific workflows, sweeps facilitate systematic exploration of hyperparameters while ensuring that each run's configuration and results are recorded for later analysis.

## 8. Discussion

### 8.1 Scientific Deployment Is Not Always Online Serving

Industrial MLOps literature often assumes deployment into live services with millisecond latency and high throughput. Scientific workflows present a different reality. In HEP, for example, models are applied to large datasets in batch to produce analysis variables; models may be retrained only when detector conditions change. Consequently, deployment focuses on packaging models and code into reproducible containers and documenting the environment for later reprocessing rather than serving predictions via REST endpoints. Monitoring involves periodic evaluation on



PoS(ISGC2026)025

new data rather than real-time alerts. Nevertheless, the same MLOps principles-version control, artefact management, lineage, experiment tracking-are indispensable for ensuring that results can be reproduced and compared across analysis iterations. The case study shows that even offline pipelines benefit from run tracking and artefact versioning.

### 8.2 Towards Community Benchmarks and Registries

Scientific ML currently lacks the shared benchmarks and model hubs that have catalysed progress in natural language processing and computer vision. We propose establishing community-driven benchmark suites for recurring HEP tasks such as signal-background classification, energy regression, anomaly detection, uncertainty quantification, object tagging, fast simulation and domain adaptation. Each benchmark should include standardised datasets or data interfaces, defined train/validation/test splits, agreed metrics, uncertainty reporting conventions and reproducible baseline pipelines. Together with an open registry of models, datasets, artefacts and complete pipelines, such benchmarks would enable transparent comparison and fair competition. Researchers could submit trained models, full MLOps pipelines, configuration files, data processing scripts, evaluation reports and lineage metadata. This vision draws inspiration from benchmark suites like GLUE for language understanding: they did not solve the field but provided a structured ecosystem that accelerated progress. In HEP and other sciences, analogous benchmarks could transform scattered efforts into a cohesive, reproducible landscape.

### 8.3 Limitations and Open Challenges

Despite the promise of MLOps, several challenges remain. Domain heterogeneity means that pipelines must accommodate a variety of data modalities (e.g., images, time series, point clouds) and physics-motivated features. Data privacy and collaboration policies may restrict sharing of raw datasets, complicating cross-experiment benchmarking. Large-scale datasets impose storage and compute costs for versioning. Maintaining reproducible environments over decades requires preserving container images and software dependencies. Integration of MLOps tools with domain-specific frameworks (e.g., ROOT, Geant4) is still immature. Finally, cultural change is necessary: researchers must adopt systematic tracking and versioning practices, which may initially slow down exploration but ultimately improve reliability.

## 9. Conclusion

MLOps extends the principles of DevOps to the unique challenges of machine learning, emphasising automation, reproducibility, versioning, monitoring and collaborative development. For scientific workflows, where data volumes are immense and reproducibility is non-negotiable, these principles are not optional but essential. We have outlined a structured MLOps pipeline, summarised best practices and surveyed the tooling landscape. A detailed case study demonstrated how the `higgsML-uncertainty-mlops` repository leverages W&B to track experiments, version datasets and models, and record lineage. Although scientific deployment seldom involves real-time serving, the same MLOps mechanisms enable robust and transparent offline workflows. Looking forward, community benchmarks and registries could catalyse progress by providing shared tasks, metrics, baseline pipelines and reproducible artefacts. We hope this work encourages the adoption



PoS(ISGC2026)025

of MLOps in scientific domains and stimulates further research on reproducible, transparent and collaborative ML workflows.

## Acknowledgements

This research was partly funded by PNRR - M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 8 "Pervasive AI" and the European Commission under the NextGeneration EU programme. The collection of original images was supported by funding from the University of Bologna and the European Space Agency (Research agreement collaboration 4000123556).

## References

[1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo and D. Dennison, *Hidden Technical Debt in Machine Learning Systems*, Advances in Neural Information Processing Systems 28 (2015).

[2] D. Kreuzberger, N. Kühl and S. Hirschl, *Machine Learning Operations (MLOps): Overview, Definition, and Architecture*, IEEE Access 11 (2023) 31866-31879.

[3] S. Kapoor and A. Narayanan, *Leakage and the Reproducibility Crisis in Machine-Learning-Based Science*, arXiv:2207.07048 (2022).

[4] J. Huang, C. Nyamawe and A. K. Sinha, *Data-centric AI for Reproducible Machine Learning: A Review of Provenance and Workflow Practices*, preprint (2021).

[5] M. Testi, L. Clissa, M. Ballabio, S. Ricciardi, F. Baldo, E. Frontoni and G. Vessio, *Enhancing Cell Counting through MLOps: A Structured Approach for Automated Cell Analysis*, arXiv:2504.20126 (2025).

[6] L. Clissa, *Big Data Challenges for High-Luminosity LHC Workflows*, conference contribution / preprint cited for HL-LHC data-volume estimates.

[7] Weights & Biases, *Create and manage runs*, W&B Documentation, accessed 2026. https://docs.wandb.ai/guides/runs/

[8] Weights & Biases, *Artifacts*, W&B Documentation, accessed 2026. https://docs.wandb.ai/guides/artifacts/

[9] Weights & Biases, *Sweeps*, W&B Documentation, accessed 2026. https://docs.wandb.ai/guides/sweeps/

[10] L. Clissa, *higgsML-uncertainty-mlops*, GitHub repository. https://github.com/clissa/higgsML-uncertainty-mlops



PoS(ISGC2026)025