Quality Assurance of Predictive Models in Intelligent Systems

Grega Vrbančič*, Zala Lahovnik and Vili Podgorelec

Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška cesta 46, 2000 Maribor, Slovenia

Abstract

Predictive models are becoming an integral part of modern information systems, transforming them into intelligent systems, featuring automated, data-driven decision-making across various domains. However, the reliance of intelligent systems on the evolving data streams and advanced machine learning algorithms introduces unique quality assurance (QA) challenges. This paper provides insight into systematic approaches for ensuring the reliability and robustness of predictive models, emphasizing the role of Machine Learning Operations practices. It addresses critical QA aspects such as data validation, drift detection, and model testing methodologies. A detailed case study of the Airly platform, a solution for air quality prediction, demonstrates practical implementations of these principles, including automated data pipelines, rigorous statistical validation, data version control, and continuous model monitoring. The study demonstrates the necessity of structured, automated, and integrated QA pipelines to maintain model performance throughout the intelligent system life cycle.

Keywords

Quality assurance, Predictive models, Intelligent systems, Machine Learning Operations (MLOps)

1. Introduction

Predictive models are becoming fundamental components of intelligent systems across diverse domains, including healthcare, finance, environmental monitoring, and industrial automation. These models facilitate automated, data-driven decision-making, thereby significantly enhancing the capabilities and efficiencies of modern systems. Nevertheless, integrating predictive models within intelligent systems brings forth distinct quality assurance challenges due to their inherent reliance on data-driven processes rather than traditional software engineering practices. Specifically, any modification within these systems, whether to the data, pre-processing methods, or model parameters, has the potential to affect overall predictive performance and system stability, captured succinctly by the phrase ”changing anything changes everything” [1].

Traditional software quality assurance methods, centered predominantly around static code testing and deterministic outcomes, fall short in addressing the nuanced demands posed by predictive models. Unlike conventional software components, predictive models continuously evolve as they consume new and potentially shifting data distributions. Consequently, model performance can deteriorate silently due to unobserved data drift, concept drift, or changes in underlying relationships between inputs and outputs. Such subtle yet critical changes can lead to erroneous predictions without rigorous and systematic quality assurance practices, undermining user trust and compromising system integrity [2].

Addressing these challenges necessitates a systematic approach to managing the life cycle of predictive models. Recent advancements in Machine Learning Operations (MLOps) propose comprehensive strategies to ensure the quality, reproducibility, and robustness [3]. Central to MLOps is the use of structured and automated pipelines that facilitate continuous data validation, systematic testing, and rigorous performance monitoring. These pipelines are essential to detect and prevent model degradation and manage changes in data, model architectures, and environmental configurations methodically.

SQAMIA 2025: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 20–12, 2025, Maribor, Slovenia

*Corresponding author.

grega.vrbancic@um.si (G. Vrbančič); zala.lahovnik1@um.si (Z. Lahovnik); vili.podgorelec@um.si (V. Podgorelec)

0000-0003-0723-3889 (G. Vrbančič); 0009-0003-9807-4242 (Z. Lahovnik); 0000-0001-6955-7868 (V. Podgorelec)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-4077/paper19.pdf

Within this context, data validation and testing play crucial roles. Effective validation pipelines help ensure the integrity, consistency, and relevance of training and inference data, reducing risks associated with data drift and anomalies. Testing strategies, including schema validation, statistical distribution checks, and performance evaluations, provide early warnings of potential issues, thus enabling proactive maintenance of predictive models.

To illustrate these principles and methodologies in practice, this paper presents a detailed case study based on the Airly platform, an MLOps-based solution developed for air quality prediction in Slovenia. The Airly platform demonstrates a comprehensive application of systematic quality assurance practices, including automated data pipelines, rigorous validation and statistical testing of datasets, structured version control using DVC (Data Version Control), and continuous monitoring through model registries and dashboards integrated via MLflow. This case study underscores the critical importance and practical benefits of systematic, pipeline-oriented quality assurance approaches for predictive models embedded within intelligent systems.

The remainder of the paper is structured as follows. Section 2 presents the most common existing methodologies and frameworks for quality assurance of predictive models, together with data validation and data testing. In section 3, the Airly platform is presented, outlining its features and incorporated quality assurance practices. Finally, the conclusion synthesizes key findings, discusses implications for broader practice.

2. Quality assurance of predictive models

Quality assurance for machine learning predictive models spans the entire life cycle of intelligent system development, from acquiring the data to model deployment. Ensuring high-quality data and rigorously testing models are critical, as poor data or unexpected model behavior can nullify the benefits of machine learning algorithms [4]. This section reviews four key components of QA in ML systems: data validation, data testing (drift detection), testing of predictive models, and their integration into MLOps pipelines.

2.1. Data validation

Reliable predictive modeling requires robust data validation processes. As emphasized by Polyzotis et al. [4], data validation is crucial to detect schema mismatches, outliers, and anomalies before they affect model training or predictions. The errors or inconsistencies in input data can compromise the predictive performance of the model [5, 4]. Additionally, feedback loops in intelligent systems, where model predictions are fed back as training data, can amplify even small data errors, causing gradual model performance degradation over time. Therefore, catching data issues early, before they propagate through training or inference, is crucial for reliable intelligent systems.

In general, there are two approaches to data validation. The first relies on manually defined validation rules or constraints – essentially "unit tests" for data. Researchers have proposed declarative frameworks that allow engineers to write assertions about data (e.g., value ranges, missing value thresholds, uniqueness), which are then automatically checked on datasets updated with new data. The second approach is schema-driven validation. Here, a data schema defines the expected structure, types, and sometimes statistical properties of the data. The schema acts as a contract for what "correct" data should look like in terms of features, types, allowed domains, ranges, etc. Frameworks like TensorFlow Data Validation (TFDV) and Great Expectations offer automated validation of data against inferred or user-defined schemes [6]. These tools validate feature distributions, missing value rates, and type consistency, and flag issues through validation reports. Such validations are typically performed before training and prior to model inference. With the utilization of such tools, organizations can continuously enforce data quality constraints, preventing bad data from ever reaching the model training phase. This not only catches catastrophic errors (such as scrambled feature values or corrupted files) but also subtle issues that could bias or degrade the model if left unchecked. In summary, rigorous data validation –

whether via manually crafted rules or schema-driven automation – is the first line of defense in ML quality assurance, ensuring that models train on clean, trustworthy data.

2.2. Data testing

Regardless of data validation, the data can and most probably will change over time, leading to the effect of data drift or concept drift. Data drift can occur in different ways. The most common are covariate drift, prior probability drift, and conceptual drift. Data drift refers to shifts in the input feature distributions over time, without changes in the underlying relationship between inputs and outputs [7]. Typical examples include shifts in demographic data, sensor calibration changes, or varying data collection methods. Data drift can lead models trained on historical data to perform poorly due to the mismatch between training and production environments [8]. Statistical tests like the Kolmogorov–Smirnov (K–S) test [9] or Population Stability Index (PSI) [10] are commonly employed to detect such shifts, comparing the empirical distributions of new and historical data [11]. Additionally, machine-learning-based detectors, such as classifier-based two-sample tests and autoencoders, have demonstrated effectiveness in identifying complex, high-dimensional distributional changes [12].

In contrast to the covariate drift, the concept drift occurs when there are changes in the conditional distribution, reflecting alterations in the underlying relationships or the decision boundary that the predictive model attempts to learn [13]. Real-world examples include evolving customer preferences, changes in fraud patterns, or medical diagnosis criteria evolving over time. Concept drift basically implies that the previously learned model concept becomes obsolete, thus requiring model adaptation or retraining. Common detection techniques include monitoring prediction error rates with methods such as ADaptive WINdowing (ADWIN) [14], and Early Drift Detection Method (EDDM) [15].

Prior probability drift refers to shifts in the distribution of the output variable, without affecting the relationship between features and targets directly [7]. For example, in medical diagnosis tasks, the prevalence of disease may shift, affecting the baseline probabilities. Prior probability drift primarily impacts the calibration and decision thresholds of the predictive models [16]. Detection of prior probability drift typically involves tracking the class distribution changes over time through straightforward statistical measures such as chi-squared tests or Fisher's exact tests. Recalibration techniques like Platt scaling, isotonic regression, or threshold adjustment are also commonly used to mitigate the effect of prior probability drift [17].

Identifying drift is critical, but it is equally important to establish strategies to mitigate its effects. For covariate and concept drift, retraining or incremental learning algorithms are essential [8]. Prior probability drift typically requires recalibration of model probabilities rather than retraining the entire model from scratch [17, 16]. All these drift types should be integrated into automated pipelines within MLOps frameworks, which continuously check for distributional changes, performance degradation, and calibration drift.

2.3. Testing the predictive models

Aside from data validation and testing, it is also important to focus on the testing of the predictive models. Machine learning models pose specific challenges to traditional software testing practices, as they are derived from data rather than being programmed in the classical sense. Therefore, it is difficult to construct traditional test oracles [18]. ML systems are also prone to silent failures, influenced not just by the model but by dependencies across the data pipeline, pre-processing logic, and external libraries.

Despite these challenges, practitioners have adapted software testing best practices to ML systems. At a high level, the most common are unit testing, integration testing, and system testing. Unit testing involves testing the smallest components of the ML pipeline in isolation. For an intelligent system, that would be testing the data pre-processing functions, feature extraction logic, or model interface code. Polyzotis et al. [4] also advocated for unit tests on the model's learned parameters or predictions for synthetic inputs to verify the training code isn't flawed.

Since the intelligent systems involve multiple pipelines, each with numerous components, the

integration tests ensure that these components work together as intended. A common practice is to run a small end-to-end pipeline on a smaller sub-sample of a dataset and verify that the pipeline produces outputs of the correct format and within the expected ranges. From the academic standpoint, the researchers have proposed frameworks for metamorphic integration testing, where the entire ML pipeline is exercised under transformations of input data to see if the end-to-end output changes in expected ways.

Testing the models' prediction logic is probably the most challenging task. Addressing this challenge, the metamorphic testing is a technique where one leverages known relationships (metamorphic relations) between different inputs and outputs to test the program without a priori oracles. In metamorphic testing, we generate new test cases from existing ones by applying transformations, like adding noise, scaling features, shuffling input order, and checking for consistency or predefined changes in the model output [19].

Another important aspect of model testing is robustness testing, which overlaps with metamorphic and adversarial testing. Techniques like adversarial example generation slightly perturb inputs to see if the model output changes dramatically, and can be seen as testing the model's stability [20]. Additionally, mutation testing has also been adapted, introducing small modifications (mutations) to a trained model (or to its inputs) to see if the model's outputs change, which can help identify brittle logic[21].

Testing ML systems is not only technically challenging but also not yet fully standardized in industry. Empirical studies highlight that software teams often struggle to apply traditional testing to ML projects and have had to develop new practices. For example, a recent industry survey by Li et al. [22] found that testing activities for ML systems focus on three major areas: test data collection, test execution, and monitoring model performance. In particular, they identified entanglement among components as a key problem in test execution. Another challenge is that after conducting tests, the interpretation of the results can be non-trivial since metric thresholds (accuracy, F1 score, etc.) are commonly used as proxies for correctness, and when these metrics drop, it's akin to a test failure. However, understanding why the metric dropped requires human insight and debugging, often beyond the scope of automated testing.

2.4. Integration into MLOps workflows

Modern practices in machine learning operations emphasize that the QA techniques should be automated and integrated as part of the ML pipeline. In a robust MLOps pipeline, data validation, model testing, and monitoring are incorporated into the continuous integration and deployment process for ML models. This ensures that quality checks are consistently applied every time data or code changes, much like continuous testing in DevOps.

In production workflows, data validation is executed prior to model retraining, ensuring that only validated data propagates through the pipeline. Therefore, the potential problems can be caught early, the pipeline can halt or alert when problems arise, similar to failing a unit test in continuous integration. In the same manner, schema validation can also be set up to detect distribution shifts relative to training data, feeding into data drift detection mechanisms [23]. Furthermore, after the data validation and data testing, is it common to automate training or retraining of the model and automatically evaluate it on the test set. If the evaluated model meets defined performance criteria, the pipeline proceeds to push the model to the production stage.

Another aspect of integration is also continuous monitoring in production. After the deployment of the predictive model, an MLOps pipeline does not end. It enters the monitoring phase, where data and predictions are continually logged and analyzed. Here, the earlier-mentioned drift detection methods run on production data streams. When significant issues like data drift are detected, it is common to trigger automated workflows, raising an alert, rolling back to a previous model, or scheduling a new training job with fresh data. When such MLOps loop is closed, ideally, the pipeline itself becomes self-correcting, which means no new model goes out, and no new data influences a model without passing quality assurance checks.

3. Case Study: Quality Assurance in the Airly application

Airly platform, presented on Figure 1, is an MLOps solution developed for monitoring and predicting air pollution levels (specifically PM10 and PM2.5 particulate matter) across 21 air measurement stations in Slovenia. The platform implements comprehensive quality assurance mechanisms throughout the intelligent system life cycle, from data acquisition, validation, and testing to model training and deployment.

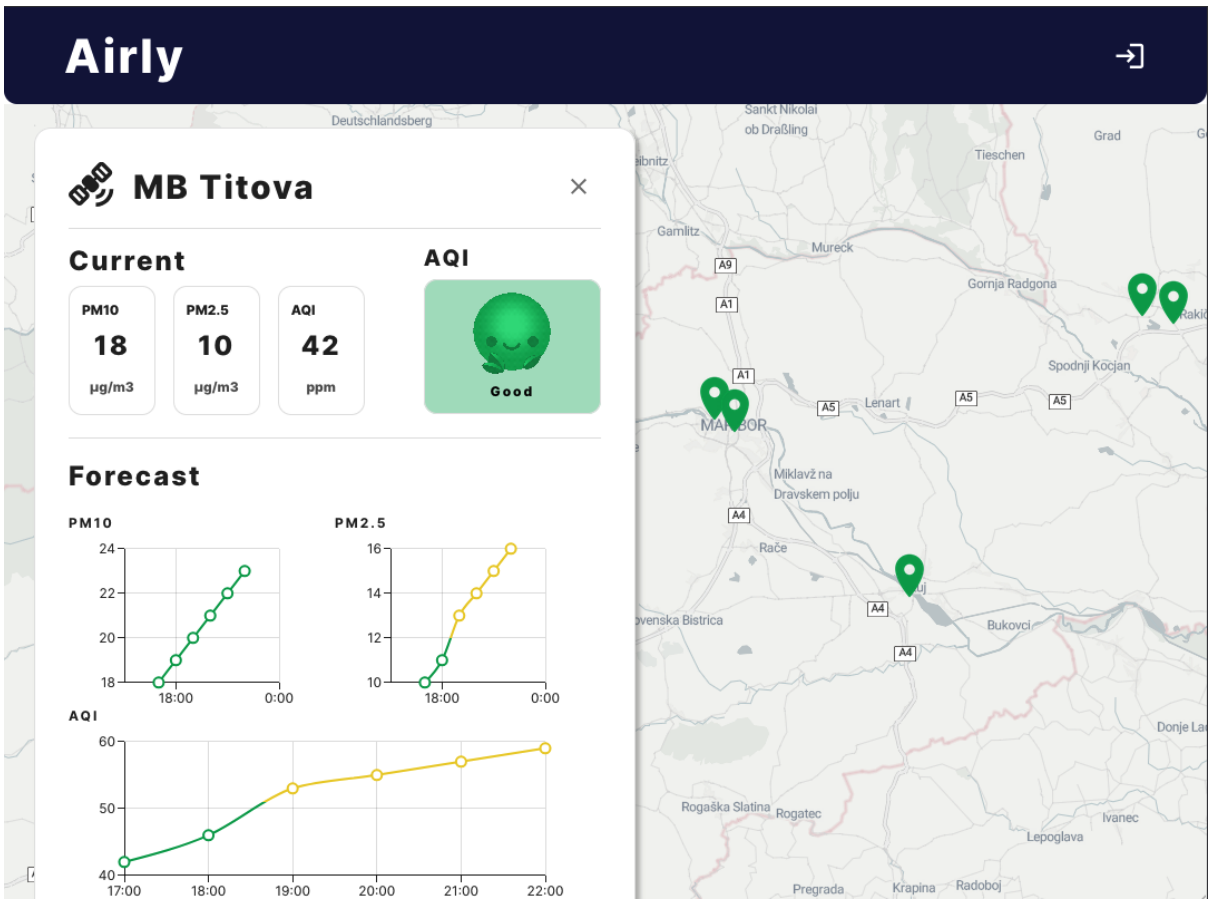

Figure 1: Demonstration of the Airly platform for predicting the air quality parameters in Slovenia.

3.1. Technology stack

The project utilizes a technology stack designed for scalability, maintainability, and efficient development. The user interface is developed using React.js, a declarative JavaScript library for building interactive web applications. The backend is implemented in Python, using the Flask web framework to provide RESTful APIs for data access and prediction services (together with ONNX runtime library). Data acquisition is implemented using BeautifulSoup and requests libraries for scraping and parsing the pollution data from external web sources, while Pandas, NumPy libraries are utilized for data manipulation and numerical operations. MongoDB is used for persistent storage of predictions accessed via the pymongo library. For the machine learning tasks, scikit-learn and TensorFlow libraries are used, while for the experiment tracking and model management, the MLFlow and ONNX libraries are utilized.

3.2. Data Pipeline

The data pipeline, presented in Figure 2, was developed with a strong emphasis on data quality assurance, reproducibility, and robust automated operation.

Figure 2: Automated data pipeline stages.

Raw air-quality sensor, together with weather data, is obtained and transformed using established data processing approaches and techniques. After the initial pre-processing phase, the pipeline merges the data and enforces automated validation checks at multiple stages to ensure the integrity and consistency of the input data. Data quality is of great importance in such pipelines, as inconsistencies or anomalies in the input can significantly degrade model performance [4]. Therefore, we adopt a schema-driven validation approach using the Great Expectations library, which can be observed in Figure 3. The data must satisfy a suite of predefined quality criteria (schema compliance, valid ranges, distributional properties) before it is passed to the next pipeline steps. Such an approach enables the automatic detection of aberrant or out-of-bound data (for example, sensor malfunctions or extreme outliers) early in the process, preventing corrupted data from propagating downstream and thereby protecting the process of training the model.

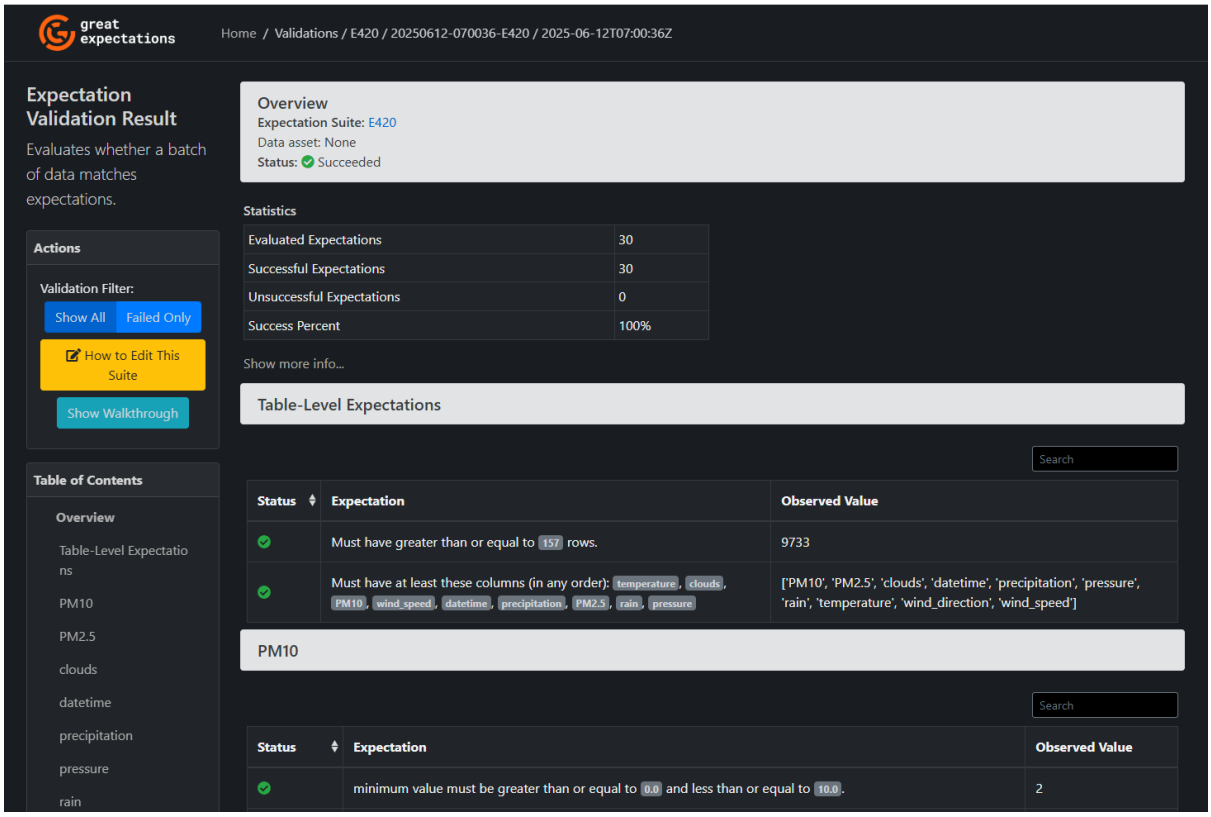

Figure 3: Data validation report for data of one measurement station on the Airly platform.

To support experimental reproducibility and traceability, the pipeline incorporates strict data versioning. Every dataset snapshot is tracked under data version control alongside the code, producing a detailed record of how data and code evolve over time and which versions are used for each training cycle. This practice follows recommended MLOps principles to apply version control to ML assets (datasets, features, trained models) in addition to source code [24]. In our implementation, we leverage a Git-integrated data versioning tool (Data Version Control, DVC) to achieve this linkage. Such an approach has become a popular standard in MLOps practice, as DVC and similar tools allow the seamless integration of data versioning into ML workflows [24]. By using such a versioned data repository, any result in the Airly case study can be reproduced exactly by fetching the corresponding dataset version

and model state, which strengthens the scientific rigor and auditability of the pipeline.

In addition to ensuring data integrity and reproducibility, the pipeline includes mechanisms for continuous monitoring of data. Over time, the statistical properties of environmental sensor data may shift due to seasonality or changes in sensor behavior, and the model's real-world performance can be affected by these shifts. Such data drift or concept drift is a well-known challenge in the machine learning life cycle, as it can quietly erode a model's accuracy and reliability if left unchecked [25]. To address this, our pipeline automatically tracks the distribution of incoming data and the model's predictions, and it triggers alerts or mitigation steps when significant deviations from the training baseline are detected. We implement this stability testing via an open-source monitoring suite (e.g., Evidently), which can be observed in Figure 4 that continuously evaluates whether the live data falls within expected bounds. By integrating such automated drift detection into the pipeline, any emerging anomaly or degradation in model behavior can be identified promptly. This enables a proactive response (such as triggering data quality investigations or model retraining) before the issue materially impacts decision-making, thereby maintaining the overall robustness of the Airly platform.

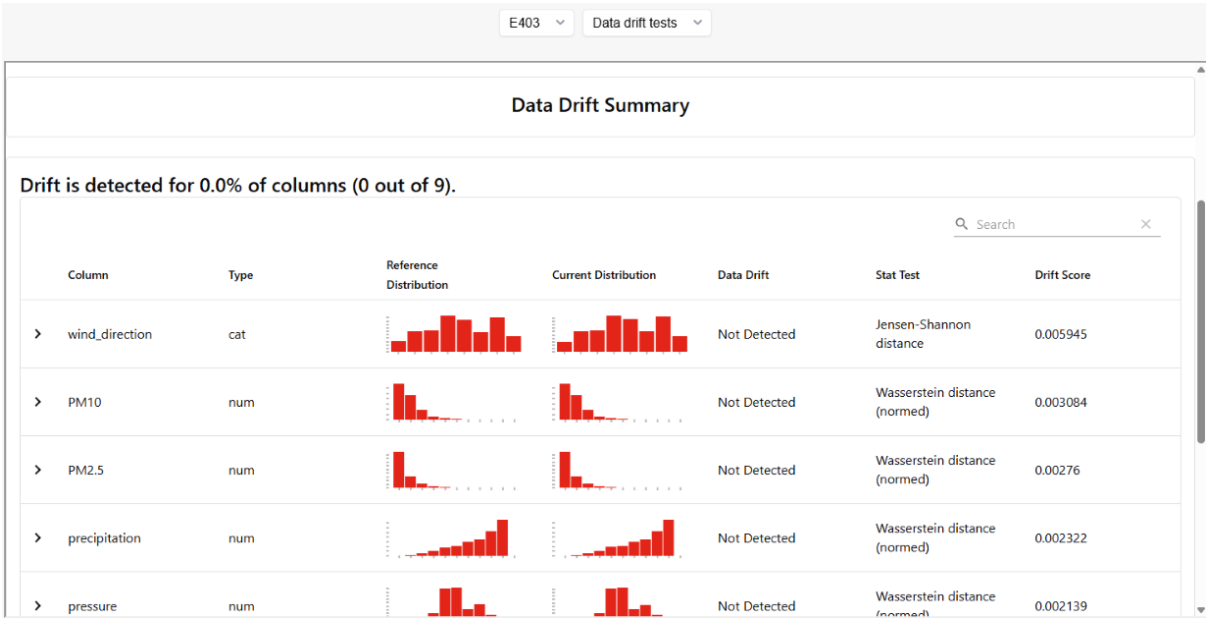

Figure 4: Data test report for data of one measurement station on the Airly platform.

3.3. Model Training Pipeline

The model training and model registration/versioning pipeline in the Airly platform is designed to ensure that predictive models are developed, evaluated, and deployed in a manner consistent with the principles of MLOps: reproducibility, automation, and traceability. This pipeline is triggered upon changes in code or data. The use of automated orchestration tools for machine learning life cycle tasks has been widely advocated in recent MLOps literature to minimize manual errors and enforce reproducible workflows [24, 23].

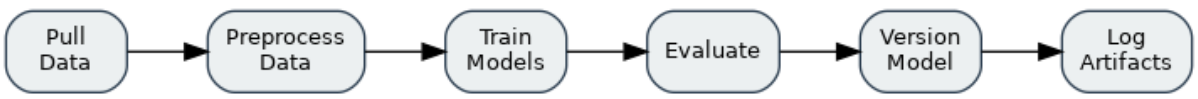

Figure 5: Automated training pipeline stages.

The pipeline begins by setting up a consistent Python environment using Poetry, ensuring reproducibility in dependency resolution and package management. The most recent validated datasets are

retrieved using DVC, linking model training directly to the corresponding versioned data. This practice enables rigorous traceability and reproducibility, aligning with best practices in version-controlled ML development.

Model training is conducted using the TensorFlow and scikit-learn frameworks, both of which support modular development and integration with pipeline components. Scikit-learn pipelines are utilized to manage feature scaling, encoding, and imputation, while the training routine, which exploits the knowledge distillation techniques, is implemented using TensorFlow. To evaluate model performance, the pipeline calculates standard regression metrics such as mean absolute error (MAE), mean squared error (MSE), and explained variance score (EVS). These metrics are logged into MLFlow, an open-source platform for managing the end-to-end machine learning life cycle. MLFlow's experiment tracking and model registry capabilities provide centralized visibility into model performance and facilitate comparison across different runs and versions. The example of executed Airly experiments, logged to MLFlow, can be observed in Figure 6.

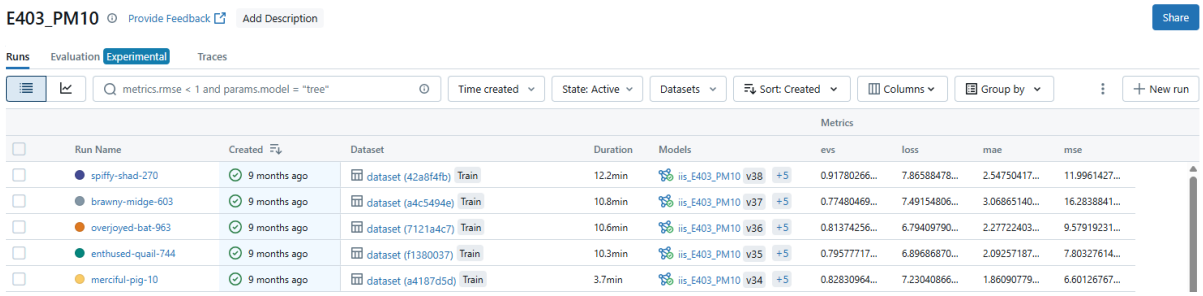

Figure 6: The list of executed experiments for a specific air quality measuring station logged to MLFlow.

Following evaluation, trained models are converted to the ONNX format, which provides a platform-agnostic representation that facilitates efficient model inference and integration into diverse runtime environments. Both the pre-processing pipeline and the trained models are registered/versioned in MLFlow, which allows for storage of metadata, tagging of model versions, and management of aliases (e.g., "production", "champion") to support controlled deployment.

The pipeline also includes rigorous testing infrastructure. Unit tests validate individual components, such as data transformation routines and model structure, while integration tests verify the full execution path from data loading to model registration. By incorporating automated environment setup, data versioning, robust model training, reproducibility-focused evaluation, and model life cycle management, the Airly training pipeline adheres to modern MLOps practices and provides a scalable framework for continuous model improvement.

3.4. Production Monitoring Pipeline

To maintain the reliability of deployed predictive models over time, the Airly platform integrates a dedicated model monitoring pipeline presented in Figure 7. This component executes on a scheduled basis (daily) to assess both prediction performance and data quality. Continuous monitoring of deployed models is a critical aspect of operational machine learning systems, allowing early detection of performance degradation or data drift.

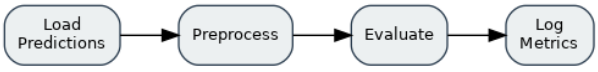

Figure 7: Automated monitoring pipeline stages.

The monitoring process begins by establishing a consistent Python environment via Poetry, followed by automated retrieval of recent predictions and ground truth observations from MongoDB and external APIs. The monitoring script pre-processes and aligns the data to enable model evaluation. Performance

metrics, including MAE, MSE, and EVS, are computed using scikit-learn and are evaluated separately for each deployed model version. The results of these evaluations are stored in MongoDB and optionally logged into MLflow. This integration provides long-term observability and supports centralized analysis of model behavior over time. Such historical tracking, presented in Figure 8, is essential for detecting trends, diagnosing issues, and managing model life cycle decisions in an evidence-based manner. Additionally, predictive model aliases and active model versions can be managed directly through the platform interface. If newer models demonstrate better performance, they can be promoted to production by reassigning the alias to the corresponding model version.

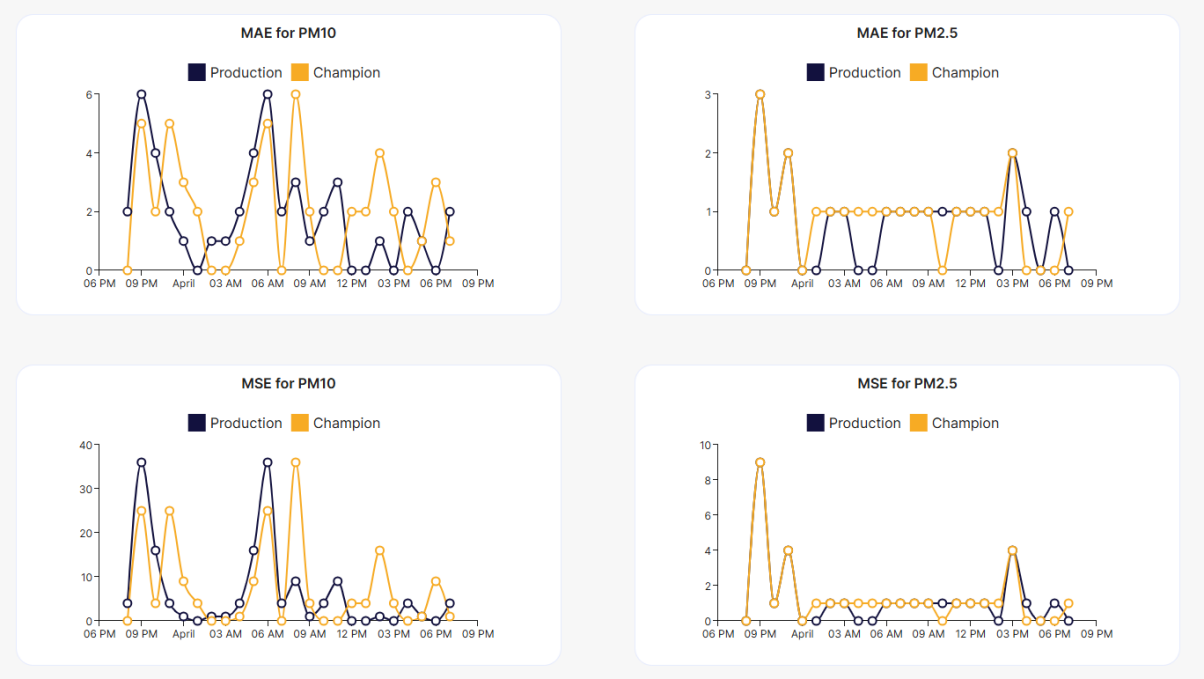

Figure 8: Evaluation metrics of the deployed predictive models integrated on the Airly platform.

3.5. Automation and Infrastructure Support

The quality assurance infrastructure is supported by automated workflows, executed via GitHub Workflows. Each pipeline stage is implemented as a separate job within the CI/CD framework, as can be observed in Figure 9, ensuring that any modification to data, code, or configuration is rigorously validated before integration. The pipeline is defined as a sequence of interdependent stages managed by an orchestration framework, ensuring that each stage executes in the correct order under consistent conditions. Automating the end-to-end workflow in this manner aligns with best-practice methodologies in MLOps, which call for the use of automated pipelines to reliably manage ML life cycle activities from data processing to model deployment [23]. In our case study, this means that new data can be ingested and processed on a defined schedule (or continuously), validated against quality expectations, and fed through to model retraining or updates with minimal human intervention. The combination of pipeline automation with rigorous data validation, version control, and drift monitoring yields a robust data infrastructure. This design not only ensures that the Airly models are trained and updated on high-quality, reproducible data but also that the system can automatically detect and adapt to changes in data or performance over time. Such a formalized pipeline approach is well-suited for any intelligent system, as it provides the reliability, transparency, and scientific rigor required for deployment in a real-world scenario.

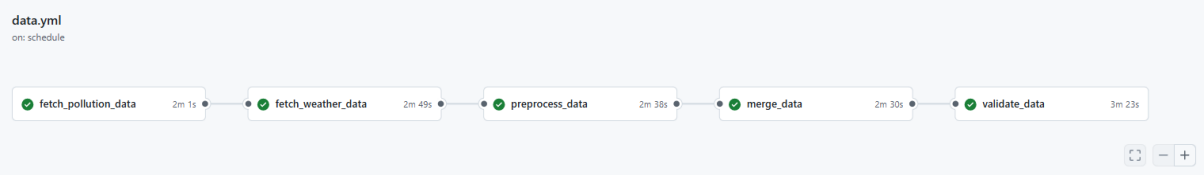

Figure 9: Hourly scheduled data pipeline for the Airly platform on GitHub.

4. Conclusions

The integration of predictive models into intelligent systems necessitates comprehensive quality assurance practices to manage the inherent complexities and dynamics associated with such systems. This paper emphasizes that effective QA must include rigorous and automated data validation procedures, robust drift detection mechanisms for identification of data and concept shifts, and systematic testing methodologies specifically tailored for predictive models. The case study of the Airly platform highlights the practical benefits and effectiveness of embedding these practices into automated MLOps pipelines, demonstrating improved reproducibility, reliability, and maintainability. Key findings support a life cycle-centric view of QA, where continuous monitoring and automated interventions, such as retraining or recalibration triggered by drift detection, are crucial. By systematically integrating validation, testing, and monitoring practices, organizations can ensure sustained predictive accuracy, mitigate the impact of data and concept drift, and maintain high-quality intelligent systems over extended operational periods.

In future work, we aim to expand our investigation by systematically evaluating the effectiveness of the proposed quality assurance approach for predictive models. This will involve simulating a range of data-related scenarios—such as shifts in data distribution, introduction of noisy inputs, or missing values—to assess the system's resilience. We also plan to perform a comparative analysis of model performance under QA-enabled versus non-QA (baseline) conditions. Such an experimental setup will help quantify the concrete benefits of integrated QA mechanisms in terms of predictive accuracy, robustness, and reliability over time.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research and Innovation Agency (Research Core Funding No. P2-0057).

Declaration on Generative AI

During the preparation of this work, the author(s) used Grammarly in order to: Grammar and spelling check. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] I. Bojinov, Keep your ai projects on track, 2023. URL: https://hbr.org/2023/11/keep-your-ai-projects-on-track.

[2] H. Wang, S. Yu, C. Chen, B. Turhan, X. Zhu, Beyond accuracy: An empirical study on unit testing in open-source deep learning projects, ACM Trans. Softw. Eng. Methodol. 33 (2024). URL: https://doi.org/10.1145/3638245. doi:10.1145/3638245.

[3] G. Vrbančič, V. Podgorelec, OTS 2019 Sodobne informacijske tehnologije in storitve: Zbornik štiriindvajsete konference, Maribor, 18. in 19. junij 2019, Maribor: Univerzitetna založba Univerze, 2019. URL: http://press.um.si/index.php/ump/catalog/book/420. doi:10.18690/978-961-286-282-4.

[4] N. Polyzotis, M. Zinkevich, S. Roy, E. Breck, S. Whang, Data validation for machine learning, Proceedings of machine learning and systems 1 (2019) 334–347.
[5] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, A. Grafberger, Automating large-scale data quality verification, Proceedings of the VLDB Endowment 11 (2018) 1781–1794.
[6] E. Caveness, P. S. GC, Z. Peng, N. Polyzotis, S. Roy, M. Zinkevich, Tensorflow data validation: Data analysis and validation in continuous ml pipelines, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 2793–2796.
[7] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, F. Herrera, A unifying view on dataset shift in classification, Pattern recognition 45 (2012) 521–530.
[8] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with drift detection, in: Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17, Springer, 2004, pp. 286–295.
[9] F. J. Massey Jr, The kolmogorov-smirnov test for goodness of fit, Journal of the American statistical Association 46 (1951) 68–78.
[10] B. Yurdakul, Statistical properties of population stability index, Western Michigan University, 2018.
[11] J. Quiñonero-Candela, Dataset shift in machine learning, Mit Press, 2009.
[12] V. Yarabolu, G. Waghmare, S. Gupta, S. Asthana, A scalable approach to covariate and concept drift management via adaptive data segmentation, arXiv preprint arXiv:2411.15616 (2024).
[13] F. Bayram, B. S. Ahmed, A. Kassler, From concept drift to model degradation: An overview on performance-aware drift detectors, Knowledge-Based Systems 245 (2022) 108632.
[14] A. Bifet, R. Gavalda, Learning from time-changing data with adaptive windowing, in: Proceedings of the 2007 SIAM international conference on data mining, SIAM, 2007, pp. 443–448.
[15] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, R. Morales-Bueno, Early drift detection method, in: Fourth international workshop on knowledge discovery from data streams, volume 6, Citeseer, 2006, pp. 77–86.
[16] T. Silva Filho, H. Song, M. Perello-Nieto, R. Santos-Rodriguez, M. Kull, P. Flach, Classifier calibration: a survey on how to assess and improve predicted class probabilities, Machine Learning 112 (2023) 3211–3260.
[17] C. Guo, G. Pleiss, Y. Sun, K. Q. Weinberger, On calibration of modern neural networks, in: International conference on machine learning, PMLR, 2017, pp. 1321–1330.
[18] H. B. Braiek, F. Khomh, On testing machine learning programs, Journal of Systems and Software 164 (2020) 110542.
[19] C. Murphy, G. E. Kaiser, L. Hu, L. Wu, Properties of machine learning applications for use in metamorphic testing., in: SEKE, volume 8, 2008, pp. 867–872.
[20] X. Yuan, P. He, Q. Zhu, X. Li, Adversarial examples: Attacks and defenses for deep learning, IEEE transactions on neural networks and learning systems 30 (2019) 2805–2824.
[21] J. Wang, G. Dong, J. Sun, X. Wang, P. Zhang, Adversarial sample detection for deep neural network through model mutation testing, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 1245–1256.
[22] S. Li, J. Guo, J.-G. Lou, M. Fan, T. Liu, D. Zhang, Testing machine learning systems in industry: an empirical study, in: Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice, 2022, pp. 263–272.
[23] M. M. John, H. H. Olsson, J. Bosch, An empirical guide to mlops adoption: Framework, maturity model and taxonomy, Information and Software Technology 183 (2025) 107725.
[24] L. Berberi, V. Kozlov, G. Nguyen, J. Sáinz-Pardo Díaz, A. Calatrava, G. Moltó, V. Tran, Á. López García, Machine learning operations landscape: platforms and tools, Artificial Intelligence Review 58 (2025) 167.
[25] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under concept drift: A review, IEEE transactions on knowledge and data engineering 31 (2018) 2346–2363.