engineering proceedings

MDPI

Proceeding Paper

A Self-Hosted MLOps Architecture for Reproducible Analytics with Data Quality Management and Artificial Intelligence Integration †

Geovanny Brito-Casanova [1,*], Valeria Torres-Lindao [2], Eduardo Benavides-Astudillo [1] and Orlando Erazo [2]

[1] Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE, Sangolquí 171103, Ecuador; debenavides@espe.edu.ec
[2] Faculty of Computer Science, Quevedo State Technical University (UTEQ), Quevedo 120501, Ecuador; vtorresl@uteq.edu.ec (V.T.-L.); oerazo@uteq.edu.ec (O.E.)
* Correspondence: gjbrito@espe.edu.ec
† Presented at the Sixth International Conference on Information Systems and Software Technologies (ICI2ST 2026), Quito, Ecuador, 23–25 September 2026.

Abstract

Reproducibility, traceability, and data quality remain persistent challenges in institutional analytics, particularly when structured datasets are processed through fragmented and weakly auditable workflows. This study proposes a self-hosted machine learning operations (MLOps) architecture for reproducibility-oriented data-quality analytics based on audit-derived programmatic supervision and local artificial intelligence interpretation. The architecture integrates data ingestion, extract-transform-load (ETL) processing, rule-based quality auditing, supervised quality-risk classification, model artifact management, and natural-language interpretation within a containerized environment orchestrated through Docker Compose. A PostgreSQL-based persistence layer, implemented with Supabase, supports structured storage, experiment traceability, and model-registry functions, while a local large language model (LLM) runtime generates human-readable decision-support outputs without relying on external cloud services. The system was evaluated using 20,045 structured administrative records from higher education. The primary leakage-controlled analysis selected Random Forest among the three evaluated classifiers. Under group-disjoint evaluation, the model achieved an accuracy of 0.806, precision of 0.564, recall of 0.716, F1-score of 0.631, area under the receiver operating characteristic curve (ROC AUC) of 0.849, and Average Precision of 0.655 on the sealed test set, with a grouped cross-validation Average Precision of 0.631 $\pm$ 0.014. Because the target was derived from deterministic audit rules, these metrics quantify internal reproduction of an aggregated operational quality-risk label from leakage-controlled contextual predictors; they do not demonstrate independent discovery of previously unknown defects. The evaluation was limited to one institutional domain and one execution environment, and no formal LLM explanation study, architecture-level benchmark, or external-domain validation was conducted. The main contribution is therefore an integrated self-hosted architectural pattern connecting data-quality governance, experiment traceability, predictive risk prioritization, and privacy-preserving local interpretation.

Keywords: MLOps; reproducible analytics; data quality management; self-hosted architecture; model governance; quality-risk classification; local artificial intelligence; institutional data analytics

Check for updates

Academic Editor: Nelson Herrera

Published: 18 September 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

The growing complexity of institutional data workflows has shifted modern analytics from isolated scripts and manual reports toward integrated execution environments where data, computation, and governance operate as a unified system. In this context, reproducible analytics is not only a property of individual models, but an architectural requirement that depends on infrastructure, data processing, model lifecycle management, and traceability mechanisms [1,2].

Many institutional analytics workflows still rely on fragmented components, including ad-hoc ingestion scripts, disconnected model training notebooks, and externally hosted services. Although these components may work independently, their separation limits reproducibility and weakens auditability [3–5]. This problem is particularly relevant in structured administrative datasets, where data quality issues directly affect downstream decision-making.

Containerization provides the foundation for unified environments, allowing complete analytical stacks to be instantiated from a single entry point [6]. Modern machine learning operations (MLOps) practices extend it by integrating data engineering, model lifecycle management, and deployment into continuous pipelines [6,7]. Nevertheless, data quality assessment remains separate from model training, usually treated as a diagnostic step rather than a computational input [8]. This separation limits the ability to transform governance signals into learning signals.

To address this gap, this study proposes an audit-to-supervision learning loop within a self-hosted MLOps architecture. In the proposed design, rule-based data quality validation generates structured audit events that are transformed into audit-derived programmatic supervision signals for model training [9]. These signals support supervised classification while maintaining traceability between raw data, quality rules, and model artifacts. The architecture is deployed through Docker Compose [4,5], integrating Supabase/PostgreSQL for persistence and experiment traceability [10–12], Scikit-learn for classification, and a local language model for privacy-preserving interpretation [13].

RQ1. How can a self-hosted MLOps architecture support reproducibility-oriented and traceable analytics for structured institutional datasets?

RQ2. To what extent can data quality auditing be transformed into supervision signals for quality-risk classification?

RQ3. How can a local artificial intelligence (AI) layer be integrated to provide privacy-preserving interpretation without relying on external cloud services?

The main contributions of this work extend beyond the engineering integration of existing open-source components. First, we propose a potentially reusable self-hosted MLOps architecture that unifies data auditing, audit-derived supervision, machine learning, model traceability, and local large language model (LLM)-based interpretation within a single reproducibility-oriented and traceable workflow for institutional analytics, whereas existing frameworks typically address these capabilities as separate lifecycle components [6,14–17]. Second, the proposed architecture introduces an audit-derived programmatic supervision mechanism that transforms deterministic data-quality validation outcomes into traceable operational labels for quality-risk classification. This mechanism enables probabilistic risk prioritization while preserving the audit engine as the authoritative source of explicit rule violations [9,18]. Third, the architecture integrates a local AI interpretation layer that converts structured analytical outputs into human-readable decision-support narratives while preserving institutional data sovereignty through fully self-hosted deployment and avoiding mandatory dependence on external cloud-based AI services [13]. The empirical claims are intentionally bounded to the evaluated dataset, grouped internal protocol, and execution environment; the study does not treat internal label reproduction, functional

LLM integration, or a single containerized execution as evidence of external defect discovery, explanation validity, or cross-environment reproducibility.

2. Related Work

The evolution of machine learning (ML) systems toward production-grade infrastructures has redefined how data-driven applications are designed, executed, and maintained. Early approaches treated model development as an isolated analytical activity. However, contemporary systems position machine learning as a full-stack engineering discipline that integrates data pipelines, computation environments, deployment layers, and continuous monitoring into a unified operational workflow [6,14]. This shift has been driven by the increasing complexity of data ecosystems, where reproducibility, traceability, and lifecycle control have become fundamental requirements rather than optional enhancements [2,8].

2.1. MLOps, Reproducibility, and Model Governance

Machine learning engineering has shifted from isolated notebooks toward infrastructures that manage ingestion, preprocessing, training, deployment, and monitoring [6,7]. MLOps combines DevOps practices with machine learning workflows [19–21], including automation, version control, experiment tracking with MLflow [22], and registries that strengthen traceability [14]. Docker Compose supports consistent execution through versioned environments [4,5].

Beyond deployment, model governance covers versioning, experiment tracking, auditability, monitoring, and artifact lifecycle management [7,15,22]. These practices link outcomes to datasets, preprocessing, configurations, and environments [14]. Containerization packages databases, application programming interface (API) layers, orchestration, and training services into versioned units [4,5], while Kubernetes adds distributed scaling and fault tolerance [23].

On the other hand, ML systems accumulate structural dependencies that extend beyond source code into datasets, feature engineering logic, hyperparameter configurations, and runtime environments [1]. These dependencies generate a form of technical debt that is not always visible at the code level but manifests in degraded reproducibility and inconsistent model behavior. Small modifications in extract-transform-load (ETL) pipelines implemented in Python or SQL, or changes in PostgreSQL schemas, can significantly alter model outputs without explicit traceability [8]. Frameworks such as Kubeflow and MLflow attempt to mitigate these issues through experiment tracking and pipeline orchestration [22]. However, they often assume stable upstream data conditions and do not fully incorporate data quality signals as part of the computational lifecycle [16,17]. Most existing solutions address deployment and governance capabilities independently, resulting in fragmented analytical workflows and limited interoperability between governance processes and ML operations [8,16,17].

2.2. Data Quality Management and Audit-Derived Supervision

Data quality management constitutes a fundamental component of modern analytical systems, as the reliability of downstream analytics depends directly on the integrity of the underlying data assets [18]. Contemporary data management frameworks define data quality through dimensions such as completeness, consistency, validity, accuracy, uniqueness, and timeliness [18,24]. Within analytical environments, quality management extends beyond isolated validation procedures and encompasses continuous mechanisms for monitoring, detecting, and correcting data anomalies throughout the data lifecycle [25,26].

Recent DataOps and MLOps practices have incorporated automated quality controls into ETL and extract-load-transform (ELT) workflows through rule-based validation en-

gines, schema verification procedures, integrity constraints, and anomaly detection mechanisms [8]. These controls allow organizations to identify structural and semantic inconsistencies before data is consumed by analytical or machine learning processes [24,25]. Furthermore, systematic recording of quality events enables the creation of audit trails that support governance, accountability, and continuous improvement initiatives [27]. The integration of data quality management into analytical pipelines is particularly relevant for machine learning applications, where data inconsistencies may propagate through training processes and negatively affect predictive performance [26].

Data lineage and provenance mechanisms provide visibility into the origin, transformation history, and dependencies of data assets throughout analytical workflows [27]. By documenting how datasets are acquired, transformed, and consumed, lineage systems contribute to transparency, reproducibility, and accountability in data-driven environments. Frameworks such as OpenLineage and DataHub have introduced standardized approaches for capturing metadata associated with ingestion processes, transformation pipelines, and analytical outputs [27]. The integration of provenance metadata with quality management mechanisms enables organizations to establish end-to-end visibility between data generation, quality assurance, and operational deployment [15,28].

Weak supervision frameworks introduce scalable alternatives to manual annotation by generating labels through heuristics, rules, and probabilistic labeling functions [9]. Systems such as Snorkel demonstrate that noisy labeling sources can be combined to produce usable training datasets for supervised learning tasks [9]. In Python-based implementations, rule engines and constraint validators can generate pseudo-labels that approximate ground truth distributions. However, weak supervision is rarely connected to data quality auditing systems, which already encode structured information about anomalies, inconsistencies, and violations that could be directly translated into learning signals [8].

2.3. Local AI Interpretation and Research Gap

Recent advances in large language models (LLMs) have expanded the possibilities for interpreting analytical outputs and supporting decision-making processes [29]. While cloud-based AI services provide powerful inference capabilities, their adoption in institutional environments frequently raises concerns related to data privacy, confidentiality, operational costs, and dependence on external providers. Local AI frameworks such as Ollama enable the execution of large language models within self-hosted infrastructures, allowing organizations to retain complete control over their data and computational resources [13]. By running models locally, sensitive analytical information remains within organizational boundaries, reducing privacy risks associated with external API-based services [13].

Beyond privacy considerations, local language models can transform structured analytical outputs into human-readable narratives [13]. Such narratives may facilitate interpretation by non-technical stakeholders, although their correctness, faithfulness, and practical usefulness require separate evaluation [30].

Although existing MLOps frameworks provide mechanisms for automation, deployment, tracking, and model lifecycle management, most approaches still treat data quality auditing, supervision generation, and analytical interpretation as separate layers. Similarly, weak-supervision methods demonstrate that programmatic labels can reduce the cost of manual annotation, but they are rarely connected to operational data-quality audit logs. These limitations motivate a systematic comparison of the capabilities provided by existing approaches and the integrated workflow proposed in this study.

To synthesize the positioning of the proposed architecture, Table 1 compares representative MLOps, DataOps and rule-based data-quality tools, weak-supervision frameworks,

and local AI approaches across five dimensions relevant to small and medium-sized institutional environments.

Table 1. Capability comparison of representative frameworks and the proposed architecture.

Approach	Quality-Derived Supervision	Fully Local Operation	Reproducibility Support	Deployment Complexity	Primary Scope
MLflow [22]	Not provided natively	Supported through self-hosting	Partial; requires integration	Low–Medium	Experiment tracking and model registry
Kubeflow [16]	Pipeline-dependent	Supported through self-hosting	Supported; configuration-dependent	High	Scalable ML orchestration
Rule-based data-quality and DataOps tools [8]	Rule outputs; no integrated predictive supervision layer	Tool-dependent	Partial	Low–Medium	Validation and audit-event generation
Snorkel [9]	Yes, through labeling functions	Supported	Partial	Medium	Programmatic label generation
Local LLM deployment, e.g., Ollama [13]	No	Yes	Not its primary purpose	Low–Medium	Privacy-preserving local inference
Proposed architecture	Yes, from audit events	Yes	Partial; evaluated environment only	Low–Medium	Institutional quality-risk analytics

Abbreviations: MLOps, machine learning operations; DataOps, data operations; LLM, large language model.

The comparison highlights two main integration gaps. First, existing data-quality and weak-supervision approaches rarely transform operational data-quality audit logs directly into traceable supervision signals for quality-risk classification. Second, existing MLOps, weak-supervision, and local LLM solutions generally address separate stages of the analytical lifecycle rather than providing a unified, fully self-hosted, and containerized end-to-end workflow without mandatory external cloud dependencies.

The comparison is capability-based rather than an empirical benchmark, because the evaluated frameworks address different portions of the analytical lifecycle and are not directly interchangeable end-to-end systems.

Therefore, the contribution of this study is not a new validation algorithm or a replacement for MLflow, Kubeflow, Snorkel, or local LLM runtimes. Its incremental value lies in their methodological integration: rule-based auditing, audit-derived programmatic supervision, predictive modeling, artifact traceability, and local AI interpretation are connected within a self-hosted, reproducibility-oriented institutional analytics workflow.

3. Materials and Methods

3.1. Deployment and Reproducibility Scope

The proposed architecture was evaluated as a self-hosted and containerized analytical system orchestrated through Docker Compose. Core services, including data ingestion, data-quality validation, model training, registry persistence, and the local LLM runtime, were defined through versioned container configurations. The reported execution demonstrates consistent operation in the evaluated environment; however, it does not establish reproducibility or portability across heterogeneous hardware and software environments.

Figure 1 illustrates the complete system design integrating data ingestion, ETL pipelines, data quality auditing, risk labeling, model training, and local LLM interpretation.

3.2. Containerized Execution Environment

The architecture is deployed through Docker Compose, with each core service isolated in an independent container.

The containerized services are built and started through the following command:

```
docker compose up -d --build
```

This configuration provides a consistent execution specification for the evaluated environment; cross-environment portability was not empirically assessed. The containerized components and their respective functions are summarized in Table 2.

Table 2. Containerized components of the proposed architecture.

Component	Technology	Function
Database Layer	PostgreSQL (Supabase)	Persistent storage and relational management
ETL Engine	Python (Pandas, SQLAlchemy)	Data ingestion and transformation
API Layer	Kong/PostgREST	Data access and service communication
ML Training	Scikit-learn	Model training and evaluation
LLM Engine	Ollama	Local natural language interpretation
Orchestration	Docker Compose	Service coordination

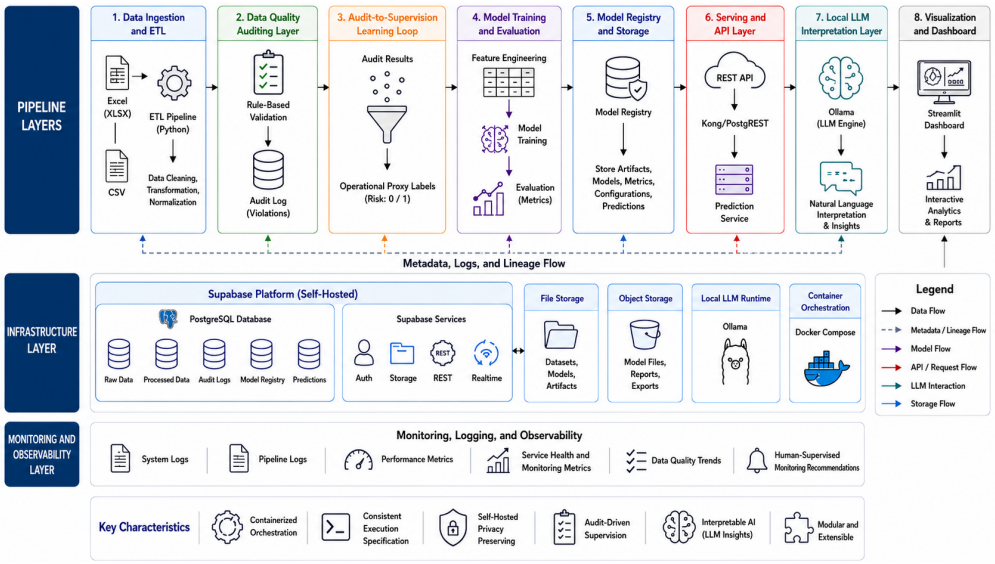


Figure 1. Evaluated self-hosted MLOps workflow integrating ingestion, deterministic auditing, audit-derived supervision, grouped training, traceability, monitoring, and local LLM interpretation. Monitoring is human-supervised; automatic drift-triggered retraining is not implemented.

3.3. Data Ingestion and ETL Pipeline

The ingestion pipeline converts Excel or CSV files into a relational format through extraction, Pandas-based cleaning and standardization, and loading into PostgreSQL schemas managed by Supabase. Although the infrastructure can be reused for other structured tabular datasets, schema mappings, transformation rules, audit rules, predictor configurations, and trained models remain domain-specific and require independent validation.

3.4. Audit-Derived Supervision for Quality-Risk Classification

The data quality layer evaluates structural and semantic integrity through a structured catalog of deterministic validation rules. Each rule specifies: (i) a unique rule identifier; (ii) the associated data quality dimension; (iii) the input attributes required for its execution; (iv) the logical violation condition; (v) the generated audit-event category; and (vi) its contribution to the operational quality-risk label. This explicit specification allows each derived label to be traced back to the rule and data attributes that produced it.

The implemented rules cover completeness, uniqueness, consistency, and referential integrity. They detect missing required values, duplicate business keys, contradictory categorical states, missing normalized territorial values, and invalid normalized province–canton pairs against a versioned reference catalog. Statistical outlier detection and numerical-

range anomaly rules were not implemented in the current study. Therefore, unusual but otherwise valid observations were not automatically classified as audit events.

The rules were implemented as part of the Python-based ETL and auditing workflow. For each processed record, the validation module evaluates the applicable rules and generates a rule-level binary indicator. Each detected violation is stored as an audit event in the PostgreSQL database, producing a structured quality log that supports monitoring, traceability, and downstream learning.

For each record x_i and audit rule r_k, a binary violation indicator z_{ik} is defined as follows:

$$z_{ik} = \begin{cases} 1, & \text{if record } x_i \text{ violates rule } r_k, \\ 0, & \text{otherwise.} \end{cases}$$

The set of rules violated by record x_i is defined as:

$$Q(x_i) = \{r_k \mid z_{ik} = 1\}.$$

A binary operational quality-risk label y_i is then assigned as follows:

$$y_i = \begin{cases} 1, & \text{if } |Q(x_i)| > 0, \\ 0, & \text{if } |Q(x_i)| = 0. \end{cases}$$

The resulting labels are interpreted as operational proxy indicators of quality risk rather than independently verified measures of record correctness. Accordingly, the labels indicate whether at least one predefined audit rule was violated. This mechanism transforms data quality auditing into traceable and repeatable supervision signals for machine learning while preserving traceability between the evaluated record, the violated rule, and the derived operational label. Figure 2 illustrates the feedback loop between auditing and model training.

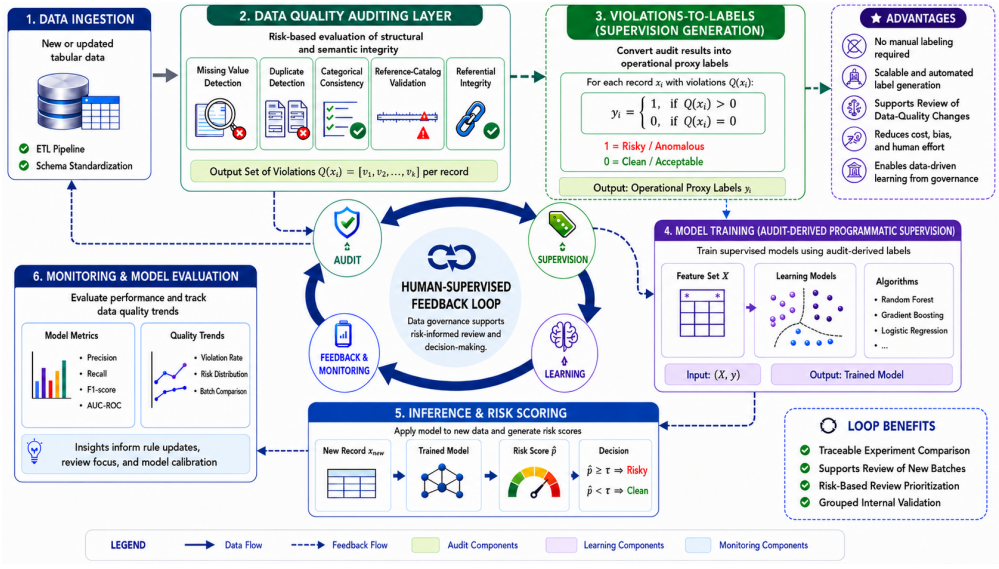


Figure 2. Human-supervised audit-to-prioritization loop. Audit rules generate operational proxy labels, and grouped models estimate risk scores for prioritization. The workflow neither retrains autonomously nor claims independent defect discovery.

3.5. Model Training and Evaluation

The primary leakage-controlled analysis used five contextual categorical predictors: institution type (`tipo_ies`), funding type (`tipo_financiamiento`), broad field of study

(`campo_amplio`), education level (`nivel_formacion`), and modality (`modalidad`). Variables that directly reproduced audit-rule conditions or represented outputs of the target-construction process were excluded. A separate full-feature scenario was retained only as a sensitivity analysis of rule-reproduction capacity and was not interpreted as independent detection of previously unknown data-quality problems.

The dataset was divided into an approximately 80/20 group-disjoint training–test partition using `natural_key`. Records sharing the same natural key were assigned entirely to either the training or test partition, resulting in 16,036 training records from 14,544 groups and 4009 test records from 3635 groups, with no group overlap. The test set remained sealed throughout feature selection, preprocessing, hyperparameter optimization, model selection, and threshold analysis.

Hyperparameter selection was performed exclusively on the training data using `RandomizedSearchCV` with five-fold `StratifiedGroupKFold` cross-validation (`shuffle=True`, `random_state=42`) and `natural_key` as the grouping variable. Forty configurations were evaluated for each algorithm using Average Precision as the optimization metric. The selected configurations were Logistic Regression with $C = 0.3$, `class_weight=None`, L1 penalty, the `saga` solver, and `max_iter=500`; Gradient Boosting with `n_estimators=200`, `learning_rate=0.05`, `max_depth=4`, `subsample=0.6`, `min_samples_split=2`, and `min_samples_leaf=10`; and Random Forest with `n_estimators=300`, `max_depth=10`, `max_features=0.5`, `min_samples_split=2`, `min_samples_leaf=2`, and `class_weight=balanced_subsample`. All imputation, categorical encoding, scaling, hyperparameter-selection, and model-fitting operations were encapsulated within Scikit-learn `Pipeline` and `ColumnTransformer` objects and fitted independently within each cross-validation fold. Complete search spaces and reproducibility resources are available in the public repository under run ID `8c464366-c5ab-433a-abb0-380bad37683a`.

3.6. Validation Strategy and Artifact Management

The rule-based audit engine and the predictive model perform different functions and were not treated as interchangeable detection mechanisms. The rule engine remains the authoritative mechanism for identifying explicit violations and assigning interpretable audit-event categories. In contrast, the learning component estimates the probability that a record belongs to the aggregated rule-defined quality-risk class. Its purpose is to provide a probabilistic ranking and support risk-based prioritization, rather than to replace deterministic validation.

A direct classifier-versus-rule baseline would be tautological because the target label is generated from those rules. The classifier is therefore evaluated only on its ability to reproduce the aggregated rule-derived outcome from predictor variables under held-out conditions; it is not claimed to discover new audit-event categories.

Operationally, the two components answer different questions. The audit engine determines whether a record violates a specific known rule after the required attributes are evaluated, whereas the classifier estimates an aggregate probability used to rank records for review from leakage-controlled contextual predictors. The classifier therefore supports triage and prioritization but does not replace direct rule execution or identify violations outside the predefined audit catalog.

3.7. Closed-Loop Module Coupling

ETL and auditing act as sensing layers; the classifier estimates operational risk; the registry preserves data, configurations, predictions, metrics, and hashes; monitoring is designed to compare subsequent audit outcomes with prior risk scores, while any rule

revision or retraining decision remains human-supervised; and the local LLM converts structured summaries into decision-support text. The loop is informational and human-supervised: outputs may motivate rule revision, experiment comparison, or a manually initiated training run, but never autonomous retraining.

Each module supports a different property. Without auditing, no operational target exists; without the registry, provenance is lost; without the LLM, predictive metrics remain unchanged but natural-language interpretation disappears. Thus, the study verifies functional integration and artifact flow rather than claiming quantitative component ablation.

4. Results

4.1. Verified Computational Execution and Operational Monitoring

During the verified Docker execution, the model search, grouped out-of-fold evaluation, encoding ablation, persistence, and report generation required 1839.65 s on 20 logical CPUs. Peak trainer memory during the heaviest parallel search was approximately 4.53 GiB. These measurements correspond to the complete model-development experiment and should not be interpreted as isolated ETL-stage performance. The verified computational characteristics of the experiment are summarized in Table 3.

Table 3. Verified computational characteristics of the reported experiment.

Metric	Value
Records analyzed	20,045
Model search, grouped OOF evaluation, ablation, persistence, and reporting	1839.65 s
Logical CPUs	20
Peak trainer memory	~4.53 GiB

4.2. Data Quality Outcomes and Label Distribution

The data quality auditing layer identified multiple structural and semantic inconsistencies in the dataset. Additionally, the final dataset characteristics and the distribution of quality-risk labels derived from auditing are summarized in Table 4.

Table 4. Data-quality outcomes and quality-risk label distribution.

Indicator	Value	Interpretation
Duplicate natural-key rows	3706	DQ-001; 1840 affected groups
Missing normalized territorial values	1176	DQ-002; 1050 affected groups
Invalid normalized province–canton pairs	0	DQ-003
Conflicting-state group events	1801	DQ-004; affecting 3617 rows
Missing institution-name rows	1	DQ-005
Missing program-name rows	0	DQ-006
Skipped missing territorial dimensions	1051	ETL diagnostic; not a mutually exclusive audit category
No quality issue detected	15,392 (76.8%)	Operational-label distribution
At least one quality issue detected	4653 (23.2%)	Operational-label distribution

Audit-event counts are not mutually exclusive because a record may violate more than one rule. Group-level events and row-level affected counts are reported separately where applicable. In addition, some event categories were sparse or absent in the evaluated dataset; specifically, DQ-003 and DQ-006 produced no positive events. Therefore, statistically meaningful event-specific classifiers could not be estimated consistently for every audit category. The current predictive task consequently uses an aggregated operational label indicating whether a record is associated with at least one predefined audit violation. Event-specific, multilabel, and cost-sensitive evaluation is reserved for datasets containing sufficient positive examples for each audit-event category.

4.3. Model Comparison and Selection

The model selection process compared three conventional supervised learning algorithms: Logistic Regression, Gradient Boosting, and Random Forest. All algorithms were evaluated using the same five-fold `StratifiedGroupKFold` procedure, and model and hyperparameter selection were based on mean grouped cross-validation Average Precision. The sealed-test metrics were reported only after model selection and did not influence the choice of algorithm or configuration. The comparison was intended to select a reference classifier within the implemented pipeline rather than to establish superiority over all available learning algorithms. The comparative results are summarized in Table 5.

Table 5. Comparative performance of quality-risk classification models under grouped validation.

Model	Grouped-CV AP	SD	Test Accuracy	Test Precision	Test Recall	Test F1	Test ROC AUC	Test AP
Logistic Regression	0.599	0.012	0.836	0.722	0.478	0.575	0.830	0.630
Gradient Boosting	0.628	0.015	0.836	0.718	0.481	0.576	0.847	0.656
Random Forest	**0.631**	0.014	0.806	0.564	0.716	0.631	0.849	0.655

Note: Bold indicates the highest mean grouped cross-validation Average Precision, the prespecified model-selection criterion.

Random Forest was selected because it achieved the highest mean Average Precision under grouped cross-validation. Sealed-test results were reported only after model selection and did not influence algorithm or hyperparameter selection.

The grouped cross-validation Average Precision difference between Random Forest (0.631) and Gradient Boosting (0.628) was 0.003. Because no paired significance test or feature-importance analysis was performed, Random Forest is reported only as the model selected by the prespecified mean-Average-Precision criterion; statistical superiority and feature-level interpretation are not claimed.

4.4. Model Registry and Artifact Traceability

The Supabase/PostgreSQL experiment registry stored model configurations, hyperparameters, evaluation metrics, audit-derived labels, predictions, dataset identifiers, grouped partitions, and prediction provenance. Trained pipelines and selected-model artifacts were generated and preserved locally with run-specific hashes. The database registry persistence completed successfully; however, the complete upload of all candidate binaries to Supabase Storage did not complete during the reported run because one artifact exceeded the active object-size limit.

4.5. Functional Verification of Local AI Interpretation

The local LLM transformed structured model and data-quality summaries into decision-support text without transmitting institutional data externally or modifying audit outcomes, labels, predictions, thresholds, or model parameters. This verifies functional integration

only: correctness, consistency, faithfulness, hallucination risk, latency, and end-user usefulness were not formally evaluated.

5. Discussion

Deterministic auditing produced traceable operational labels within a consistently specified MLOps workflow [9,15]. Rules remain authoritative for event-specific violations, whereas the classifier ranks records by aggregated risk; because the target originates from the audit catalog, the experiment does not demonstrate independent defect discovery.

A recent complementary preprint describes a cloud-oriented MLOps architecture combining deterministic ingestion, Kubernetes- and Terraform-based infrastructure provisioning, and LLM-supported interpretation, providing a contrasting deployment perspective to the self-hosted Docker Compose and audit-derived supervision workflow evaluated in the present study [31].

Under leakage control, Random Forest achieved 0.849 ROC AUC, 0.655 Average Precision, 0.716 recall, and 0.631 ± 0.014 grouped-CV Average Precision. Its lower performance than the full-feature sensitivity scenario confirms that direct audit proxies inflate rule-reproduction performance. It is therefore the selected model among the three evaluated classifiers, not a universally or statistically superior method.

The results characterize only this implementation. Feature importance, paired significance testing, external validation, controlled perturbation, module ablation, architecture benchmarking, cross-environment replication, and formal LLM evaluation were not conducted. The contribution is the integration of auditing, leakage-controlled supervision, grouped validation, traceability, and local interpretation in a self-hosted workflow [9,14–18,27].

6. Limitations

Evaluation used one higher-education dataset and an operational proxy target derived from deterministic rules. Performance therefore depends on the rule catalog, and grouped validation prevents natural-key leakage but does not establish temporal, institutional, or external generalizability. The full-feature scenario measures rule-reproduction capacity only.

No feature-importance study, paired statistical test, component ablation, controlled data perturbation, equivalent-framework benchmark, cross-environment replication, or external-domain validation was conducted. The local LLM was verified functionally but not assessed for correctness, consistency, faithfulness, hallucinations, latency, or user usefulness; monitoring remains human-supervised.

Future work will use frozen specifications on separated datasets, controlled missingness, duplication and noise, equivalent reference pipelines, heterogeneous environments, event-specific models, and expert evaluation of local explanations.

7. Conclusions

The study presented a self-hosted MLOps architecture combining ETL, deterministic auditing, audit-derived supervision, grouped modeling, experiment traceability, monitoring, and local AI interpretation. Random Forest obtained the highest grouped-CV Average Precision among the three classifiers (0.631 ± 0.014) and achieved 0.849 ROC AUC, 0.655 Average Precision, and 0.716 recall on the sealed test set.

These results characterize internal predictive performance under the evaluated grouped distribution; they do not establish external generalizability, independent defect discovery, or superiority over other classifiers or MLOps frameworks. The contribution is a reproducibility-oriented pattern connecting data-quality governance, traceable machine learning, and privacy-preserving local interpretation within the tested dataset and environment.

https://doi.org/10.3390/engproc2026153015

Author Contributions: Conceptualization, G.B.-C. and O.E.; methodology, G.B.-C., V.T.-L. and E.B.-A.; software, V.T.-L. and E.B.-A.; validation, V.T.-L., E.B.-A. and O.E.; formal analysis, G.B.-C., O.E. and E.B.-A.; investigation, V.T.-L.; resources, G.B.-C.; data curation, V.T.-L.; writing—original draft preparation, V.T.-L.; writing—review and editing, G.B.-C., E.B.-A. and O.E.; visualization, V.T.-L.; supervision, G.B.-C. and O.E.; project administration, G.B.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding. The APC was funded by Universidad de las Fuerzas Armadas ESPE.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The source workbook used in this study, the complete source code, Docker Compose configurations, deterministic audit rules, hyperparameter search spaces, and run-specific publication evidence are publicly available at https://github.com/geo-bricex/mlops-supabase-university-offers (accessed on 29 May 2026). Derived database tables and large binary model artifacts are not redistributed as part of the manuscript package but can be regenerated by executing the documented pipeline. The definitive results reported in this article correspond to run `8c464366-c5ab-433a-abb0-380bad37683a`.

Acknowledgments: The authors acknowledge the open-source communities that maintain Python 3.14, Scikit-learn 1.9.0, Pandas 3.0.5, the Supabase Python client 2.31.0, the Supabase PostgreSQL image 15.8.1.085, Docker 29.5.2, and Ollama 0.32.5, whose tools supported the implementation and evaluation of the architecture presented in this study.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kosmas, I.; Papadopoulos, T.; Michalakelis, C. Machine Learning Lifecycle: A Survey. *AppliedMath* **2026**, *6*, 113. [CrossRef]
2. Beaulieu-Jones, B.K.; Greene, C.S. Reproducibility of Computational Workflows Is Automated Using Continuous Analysis. *Nat. Biotechnol.* **2017**, *35*, 342–346. [CrossRef] [PubMed]
3. Boettiger, C. An Introduction to Docker for Reproducible Research. *ACM SIGOPS Oper. Syst. Rev.* **2015**, *49*, 71–79. [CrossRef]
4. Docker Inc. Docker Compose Documentation. Available online: https://docs.docker.com/compose/ (accessed on 29 May 2026).
5. Docker Inc. Docker Compose up CLI Reference. Available online: https://docs.docker.com/reference/cli/docker/compose/up/ (accessed on 29 May 2026).
6. Kreuzberger, D.; Kühl, N.; Hirschl, S. Machine Learning Operations (MLOps): Overview, definition, and architecture. *IEEE Access* **2023**, *11*, 31866–31879. [CrossRef]
7. Woźniak, A.P.; Milczarek, M.; Woźniak, J. MLOps Components, Tools, Process, and Metrics: A Systematic Literature Review. *IEEE Access* **2025**, *13*, 22166–22175. [CrossRef]
8. Foidl, H.; Golendukhina, V.; Ramler, R.; Felderer, M. Data pipeline quality: Influencing factors, root causes of data-related issues, and processing problem areas for developers. *J. Syst. Softw.* **2024**, *207*, 111855. [CrossRef]
9. Ratner, A.; Bach, S.H.; Ehrenberg, H.; Fries, J.; Wu, S.; Ré, C. Snorkel: Rapid Training Data Creation with Weak Supervision. *VLDB J.* **2020**, *29*, 709–730. [CrossRef] [PubMed]
10. Supabase. Self-Hosting Documentation. Available online: https://supabase.com/docs/guides/self-hosting (accessed on 29 May 2026).
11. Supabase. Data REST API Documentation. Available online: https://supabase.com/docs/guides/api (accessed on 29 May 2026).
12. Supabase. Self-Hosting Storage Documentation. Available online: https://supabase.com/docs/reference/self-hosting-storage/introduction (accessed on 29 May 2026).
13. Ollama. API Introduction. Available online: https://docs.ollama.com/api/introduction (accessed on 29 May 2026).
14. Mateo-Casalí, M.Á.; Boza, A.; Fraile, F. Towards a Reference Architecture for Machine Learning Operations. *Computers* **2026**, *15*, 218. [CrossRef]
15. Bayram, F.; Ahmed, B.S. Towards Trustworthy Machine Learning in Production: An Overview of the Robustness in MLOps Approach. *ACM Comput. Surv.* **2025**, *57*, 111. [CrossRef] [PubMed]
16. Zhao, X.; Ma, Z.G.; Jørgensen, B.N. A Systematic Lifecycle-Referenced Capability Mapping of MLOps Platforms for Energy Forecasting. *Information* **2026**, *17*, 328. [CrossRef]

17. Zhao, X.; Ma, Z.G.; Jørgensen, B.N. Operationalising an End-to-End MLOps Lifecycle for Energy Forecasting: Implementation and Controlled Evaluation on ClearML. *Information* **2026**, *17*, 576. [CrossRef]
18. Batini, C.; Cappiello, C.; Francalanci, C.; Maurino, A. Methodologies for Data Quality Assessment and Improvement. *ACM Comput. Surv.* **2009**, *41*, 16. [CrossRef]
19. Grande, R.; Vizcaíno, A.; García, F.O. Is it worth adopting DevOps practices in Global Software Engineering? Possible challenges and benefits. *Comput. Stand. Interfaces* **2024**, *87*, 103767. [CrossRef]
20. Akbar, M.A.; Yousuf, M.A.; Mukadam, S.; Shaikh, R. Toward Successful DevOps: A Decision-Making Framework. *IEEE Access* **2022**, *10*, 29901–29918. [CrossRef]
21. Arvanitou, E.-M.; Ampatzoglou, A.; Chatzigeorgiou, A.; Avgeriou, P. Applying and Researching DevOps: A Tertiary Study. *IEEE Access* **2022**, *10*, 29805–29834. [CrossRef]
22. Chen, A.; Chow, A.; Davidson, A.; DCunha, A.; Ghodsi, A.; Hong, S.A.; Konwinski, A.; Mewald, C.; Murching, S.; Nykodym, T.; et al. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning (DEEM '20), Portland, OR, USA, 14 June 2020*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1–4. [CrossRef]
23. Petrakis, K.; Kokkalas, G.; Tsolkas, A. Enhancing DevOps Practices in the IoT–Edge–Cloud Continuum: Architecture, Integration, and Software Orchestration Demonstrated in the COGNIFOG Framework. *Software* **2025**, *4*, 10. [CrossRef]
24. Jarmakovica, A. Machine learning-based strategies for improving healthcare data quality: An evaluation of accuracy, completeness, and reusability. *Front. Artif. Intell.* **2025**, *8*, 1621514. [CrossRef] [PubMed]
25. Rahal, M.; Ahmed, B.S.; Szabados, G.; Fornstedt, T.; Samuelsson, J. Enhancing machine learning performance through intelligent data quality assessment: An unsupervised data-centric framework. *Heliyon* **2025**, *11*, e42777. [CrossRef] [PubMed]
26. Xie, J.; Sun, L.; Zhao, Y.F. On the Data Quality and Imbalance in Machine Learning-based Design and Manufacturing—A Systematic Review. *Engineering* **2025**, *45*, 105–131. [CrossRef]
27. Moreau, L.; Missier, P. (Eds.) PROV-DM: The PROV Data Model. W3C Recommendation. 30 April 2013. Available online: https://www.w3.org/TR/prov-dm/ (accessed on 29 May 2026).
28. Zarour, M.; Alzabut, H.; Al-Sarayreh, K.T. MLOps best practices, challenges and maturity models: A systematic literature review. *Inf. Softw. Technol.* **2025**, *183*, 107733. [CrossRef]
29. Manzoor, M.A.; Albarri, S.; Xian, Z.; Meng, Z.; Nakov, P.; Liang, S. Multimodality Representation Learning: A Survey on Evolution, Pretraining and Its Applications. *ACM Trans. Multimed. Comput. Commun. Appl.* **2024**, *20*, 74. [CrossRef]
30. Brito Casanova, G.J. Servicio Web Basado en Analítica de datos en la nube para Instituciones de Educación Superior. Master's Thesis, Quevedo State Technical University, Quevedo, Ecuador, 2025.
31. Guerrero-Ulloa, G.; Brito-Casanova, G.; Torres-Lindao, V.; Erazo, O.; Ordóñez-Guerrero, K. Cloud-Based MLOps Architecture for Automating the Predictive Model Lifecycle: A Case Study Using Academic Publication Data from Ecuador. *Preprints* **2026**, 2026060493. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.