MLExp: A Tool for Applying Experimentation to Compare the Performance of Supervised ML Model Versions

Iury Rosal
irosal@inf.puc-rio.br
PUC-Rio
Rio de Janeiro, Brazil

Eduardo Almentero
almentero@ufrrj.br
UFRRJ
Seropédica, Brazil

Marcos Kalinowski
kalinowski@inf.puc-rio.br
PUC-Rio
Rio de Janeiro, Brazil

Abstract

The lifecycle of machine learning (ML) models presents specific challenges, particularly when comparing performance across models, configurations, and versions. Although Machine Learning Operations (MLOps) practices and tools have improved the automation and reproducibility of ML workflows, challenges remain in systematically comparing and interpreting model performance across different versions throughout deployment. As a result, data scientists often rely on manual and interactive approaches for statistical testing and model comparison, which may compromise reproducibility, consistency, and reliability. In this work, we developed and evaluated MLExp, an open-source tool designed to support the comparison of versions of supervised ML models using statistical hypothesis testing during deployment. The tool allows professionals to associate trained models with test datasets, automatically compute performance metrics, apply statistical tests to compare model performance, and generate comparative reports highlighting statistically significant differences among versions. To evaluate its effectiveness, we conducted a small-scale study with two data scientists at a large Brazilian digital bank, comparing MLExp with a previously adopted manual workflow based on Jupyter Notebooks and MLflow. Results provided initial evidence that MLExp supports more systematic and reproducible model comparisons while reducing manual effort in statistical evaluation.
Demo video: https://zenodo.org/records/21459901?preview_file=Mlexp+Demo+Cameraready.mp4

Keywords

Experimentation, Machine Learning, MLOps

1 Introduction

Unlike traditional software, Machine Learning (ML) systems change continuously as datasets, configurations, and model versions evolve. Machine Learning Operations (MLOps) automates this lifecycle, enabling more agile and efficient development through continuous integration, delivery, experiment tracking, and model versioning [16, 24, 33].

Once deployed, models lose predictive quality as distributional conditions shift, whether through data drift, a change in the input distribution, or concept drift, a change in how inputs relate to targets [22]. To counter this degradation, modern MLOps architectures rely on continuous retraining, periodically training new model versions on the most recent data [3].

To ensure that newly generated versions can be fully utilized in production, it is necessary to verify whether the candidate model meets prediction quality requirements within the operational context [19, 28]. This makes comparability, traceability, and reproducibility across model versions a central challenge in ML engineering [8].

However, performance metrics alone are often insufficient to clearly interpret and justify the outcomes of model comparisons, particularly when models exhibit similar predictive performance [10]. Continuous experimentation can further support deployment decisions by identifying significant performance differences, increasing confidence in selecting candidate models for production use [9, 37].

Existing tools predominantly support infrastructure management, pipeline automation, experiment tracking, and monitoring aggregate performance metrics, while offering limited support for statistically informed comparison of evolving supervised ML models throughout deployment [8, 15, 31]. As a result, data scientists are often responsible for manually determining whether observed differences between model versions are statistically significant [6, 15]. In practice, statistical validation is often performed using custom scripts or ad hoc analyses, leading to fragmented, non-standardized comparison workflows. Consequently, model evaluation often relies on manual, interactive testing, which may compromise reproducibility, consistency, reliability, and the interpretability of model-comparison outcomes [12, 19]. More robust solutions are therefore needed to support automated experimentation and improve the comparability, reproducibility, and interpretability of model evaluation results within MLOps workflows [30].

In response to these challenges, this work proposes MLExp, an open-source tool that enables comparison of supervised machine learning models through statistical hypothesis testing and performance benchmarking throughout the deployment of new ML model versions.

MLExp allows users to associate trained models with test sets, automatically calculate performance metrics, perform statistical comparisons, and generate comparative reports to aid in interpreting results.

To evaluate its practical usefulness, we conducted a small-scale study at a large Brazilian digital bank, comparing MLExp with the same previously adopted manual workflow, which combined Jupyter Notebooks with MLflow. We also investigated practitioners' perceptions regarding the tool's reliability and usefulness. Results indicate that MLExp supports more systematic and reproducible model comparisons while reducing manual effort in statistical evaluation. Furthermore, practitioners reported positive perceptions regarding the tool's usefulness for supporting model comparison activities.

2 Background and Related Work

This section aims to provide key definitions foundational to the construction of the MLExp tool, including Statistical testing, as well as technologies and frameworks that can support the experimentation process, to help understand how MLExp can contribute to the ecosystem.

2.1 Experimentation and Statistical Testing

An experiment aims to collect data in a controlled environment to determine whether modifying independent variables has a causal effect on a particular outcome, typically to optimize that result [2]. Hypotheses must define independent variables, representing the process inputs expected to drive changes, and dependent variables, representing the measurable process outputs. The hypothesis evaluates whether variations in the independent variables produce statistically measurable effects on the dependent variables.

In the context of Machine Learning (ML), this practice involves comparing different scenarios based on hypotheses regarding model characteristics, such as predictive performance, availability, or computational cost. Hypothesis testing in this domain is not always straightforward; it typically requires a rigorous statistical workflow applied to groups of performance data collected during the experiment. MLExp aims to automate this statistical process, including the generation of the performance data groups required for testing.

Two central concepts guide the tool's operation: **experiment** and **context**. A **context** is a trained machine learning model that predicts a target variable from a specific dataset. Each context produces a set of performance metrics, which serve as the dependent variables and form the input groups for hypothesis testing. An **experiment** is a collection of contexts executed to generate these groups for statistical comparison. Hypothesis testing determines whether significant differences exist among contexts; when they do, measures of central tendency indicate the direction of the effect, identifying which context performed best.

The appropriate statistical test depends primarily on the distributional properties of the data, such as normality and homoscedasticity, whether the observations are independent or paired (same dataset vs. distinct datasets), and the number of contexts being compared.

Normality is commonly assessed using the Shapiro-Wilk test [34], which is suitable for continuous data and effectively detects departures from normality at moderate sample sizes. Homoscedasticity, in turn, is typically assessed with Levene's test [18], which evaluates whether the variances across groups can be considered equal.

Distinct models applied to the same dataset (or paired datasets). Because each instance receives a prediction score from both model versions, the observations share a 1:1 relationship, allowing for the application of paired tests (paired t-test [36] or Wilcoxon signed-rank test [39] for two contexts; repeated-measures ANOVA [14] or Friedman test [11] for more than two). Pairing eliminates the natural variance among individual instances, thereby isolating and increasing sensitivity to the model's actual effect. This scenario is ideal for detecting degradation due to concept drift or establishing model superiority.

The same model applied to distinct datasets. In this scenario, the instances differ across groups, meaning there is no direct correspondence. Consequently, independent sample tests are required (Student's t-test [1], Welch's t-test, or Mann-Whitney U test [21] for two contexts; ANOVA [5] followed by Tukey's HSD [25], or Kruskal-Wallis [27] followed by a Dunn test with Benjamini-Hochberg correction [4] for more than two). This configuration supports operational monitoring and detects degradation caused by data drift (shifts in data distribution) rather than a decline in the model's intrinsic capability.

Distinct models applied to distinct datasets. MLExp permits this configuration via an unpaired statistical workflow, but the results demand cautious interpretation. Here, two sources of variation change simultaneously (the model and the data), and the experimental design provides no mechanism to isolate them. A significant difference could stem from a superior model, an "easier" dataset, or an interaction between the two. The statistical test cannot distinguish among these causes. Therefore, the comparison is strictly descriptive, not causal, regarding model capability. To attribute an effect directly to the model, one factor must remain fixed, reverting the design to one of the two preceding scenarios.

A significant statistical result establishes only that a difference exists. The direction of this difference is obtained from the relevant measure of central tendency. Combined, the significance and direction indicate whether a candidate version performs as expected. MLExp automates this entire rigorous process, as illustrated in Figure 1.

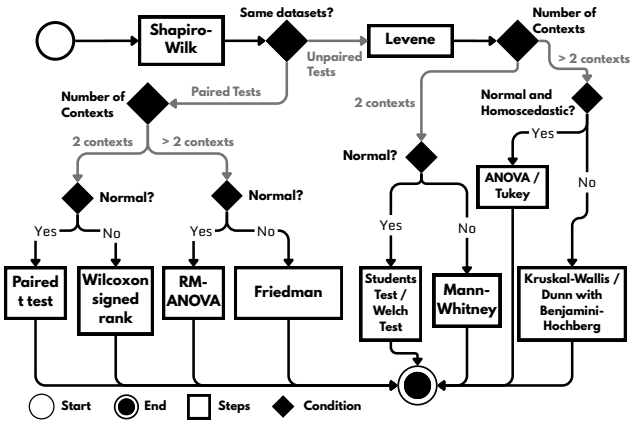

Figure 1: Statistical test selection flowchart.

2.2 Related Work

Some frameworks address how to organize and structure the evaluation of machine learning (ML) models, enabling systematic experimentation, such as the MLTE Framework [20] and ExtremeXP [31]. These frameworks establish practices and processes whose implementation depends on technologies such as ML platforms to automate workflows and integrate different artifacts.

Several MLOps platforms are widely used in industry, including Kubeflow [13], SageMaker [38], and MLflow [6]. These tools provide features supporting CI/CD workflows for ML models, facilitating

development, experiment tracking, and monitoring. However, they do not provide a native approach to continuous experimentation, requiring data scientists to configure their own automation using platform capabilities, which may still leave room for interactive and manual processes [15].

ClearML [7] and ZenML [41] focus on experiment tracking, orchestration, and metadata management, logging performance metrics and displaying them for visual comparison, yet they leave the interpretation of these metrics to the practitioner and lack a native statistical inference solution. Seldon Core [32] operates at a later stage of the pipeline, routing production traffic among candidate models via percentage-based splits to enable A/B testing and shadow deployments based on operational metric thresholds (such as latency and error rates) rather than hypothesis testing regarding predictive quality.

MLExp fills the gap left by these tools: comparing model versions using statistical inference. Instead of merely displaying metrics or splitting traffic, it verifies distributional assumptions, selects the appropriate test, applies post hoc corrections when necessary, and reports both the significance and direction of the results.

3 Problem Formulation

The problem addressed by our candidate solution follows the guidelines for formulating research problems with practical relevance [29]:

"The practical problem addressed in this research is the lack of automated and systematic support for a robust and reliable experimentation process among professionals working with supervised ML models (classification and regression). This limitation hinders the use of experimental methods to evaluate performance differences between model versions throughout deployment. In this context, MLExp is proposed to support automated statistical hypothesis testing and the systematic comparison of model performance throughout the deployment of new ML model versions."

4 MLExp: The Experimentation Tool

This work presents MLExp, an open-source tool (MIT license) that automates the comparison of the performance of supervised machine learning models, thereby supporting a systematic experimental process. The tool allows data scientists to assess performance changes between versions of machine learning models with greater statistical rigor and automation.

4.1 User Inputs and Design Experiment

The user initially defines a set of parameters that will guide the experimentation process:

- **scores_target**: Defines the performance metrics used as the basis for model comparison. Accepts a single value or a list of values. This parameter is mandatory. Possible values include `accuracy`, `roc_auc`, `f1` and `precision_recall` (for classifiers); and `mae`, `mse` and `r2` (for regressors).
- **n_splits**: Specifies the number of data groups to be generated from the test dataset. This value determines the number of performance metric observations per context. It is important to emphasize that a very high value is not recommended for small test datasets [17]. Default value: 10.
- **random_state**: This parameter is used in algorithms and computational functions involving random processes, such as data splitting. This allows the experiment to be reproduced, ensuring the same result each time the code is executed. Default value: 42
- **report_path**: Defines the folder where all generated reports will be stored. Default value: `/reports`.
- **report_name**: Specifies the name of the folder generated within `report_path` to store the results of a given run, separated by timestamp. Default value: `general_report`.
- **export_json_data**: When enabled, saves `JSON` files in the report directory specified (inside the timestamped folder) containing all performance metric values collected before the application of statistical tests. A separate `JSON` file is generated for each performance metric. Default value: `True`.
- **export_html_report**: Generates an HTML report summarizing the results of all statistical tests for the selected performance metrics, as well as identifying the best-performing model for each metric, if applicable. Default value: `True`.
- **return_best_context**: When the function responsible for executing the pipeline is called, the best-performing context for the selected performance metric is returned through the API. This option only applies when a single performance metric is defined. Default value: `False`.
- **paired**: MLExp automatically selects between an unpaired flow and a paired flow based on how contexts reference test data ("auto"). The user can override it: paired=False forces the unpaired flow, and paired=True forces the paired flow. Default value: `auto`.

The tool allows data scientists to map test data from various file formats into a *Pandas DataFrame* (e.g., CSV, JSON, Parquet). It can also accept an already instantiated *Pandas DataFrame* object. The user specifies the file or object containing the test features (X) and the corresponding target (Y). During this mapping, the user can assign a name as a unique identifier for the test dataset.

After that, the user can independently load each trained supervised machine learning model saved in a serialized format, such as *Pickle* or *ONNX*. The tool also accepts instantiated model objects from the *scikit-learn* ecosystem. Models developed using major Python libraries, such as the latter, can be converted and saved as *Pickle* and *ONNX* files, preserving their original state and facilitating standardized model deployment [26].

During the model mapping process, the user must specify which previously mapped test dataset to use with each model. The pair consisting of a trained model and its corresponding test dataset is referred to as a **context**. As with test data, each context must be assigned a unique identifier, which will be referenced in the reports generated at the end of the experiment.

Testing multiple models on the same dataset is supported, but the flow matters. In the unpaired flow, shared data violates independence and compromises validity, so distinct datasets are recommended per context. To compare versions on the same instances, use the paired flow (paired=True), which treats them as matched rather than independent observations.

All models across different contexts must belong to the same machine learning category, meaning they must all be either classifiers or regressors. Furthermore, the performance metric used in the experiment must correspond to the model category, for example, accuracy for classifiers or MAE for regressors.

4.2 MLExp's Workflow

After all contexts have been mapped, the operation of the tool can be summarized in the following steps to apply and evaluate the experiment results (Figure 2):

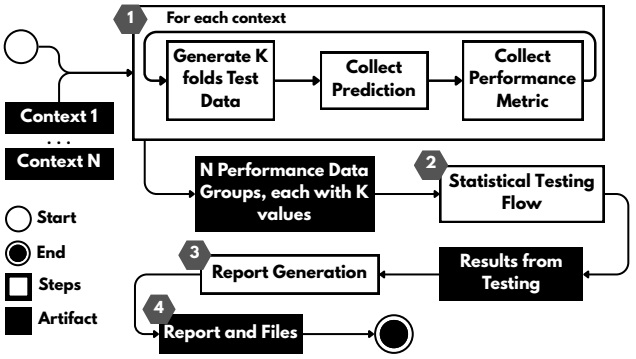

Figure 2: Tool Operation Flow.

(1) For each context, the tool utilizes the corresponding test dataset and applies a K-Fold procedure for regression contexts and Stratified K-Fold for classification contexts, dividing the test data into k folds and generating predictions for each one. It then calculates performance metrics by comparing predicted values against actual values across all folds. The ultimate goal of this step—which justifies the use of K-Fold—is to generate a set of performance data for each context.
(2) The data resulting from the performance metrics of each fold are grouped by context. Subsequently, these groups are processed through the automated statistical testing pipeline (Figure 1) to determine whether significant differences exist between them.
(3) The tool gathers results from the statistical testing pipeline and uses them to identify which contexts (and, consequently, which models) exhibit statistically significant performance differences. These results are used to identify the best performing model. The winner selection respects the metric direction: for performance statistics, where higher values indicate better performance, the model with the higher median is selected; for error-based metrics, where lower values indicate better performance, the model with the lower median is selected. In case of a tie, either model may be nominated. A winner is declared only when a statistically significant difference between contexts is observed.
(4) Finally, the tool generates output files containing the statistical test results (as JSON files for each metric) and an HTML report summarizing the analysis conclusions.

The output is a JSON file containing results for each performance metric, along with an HTML report (if applicable). If enabled, the name of the context that obtained the best result will be returned in the run() call. The user will be able to view the results in the report, thereby assisting with decision-making.

4.3 Implementation and Architecture

The library was implemented in *Python* due to its widespread adoption in data science and artificial intelligence applications, as well as its extensive ecosystem integration.

The *Pandas* library was employed for data manipulation, while *scikit-learn* was utilized for data processing, including the K-Fold logic, and for computing performance metrics. The *Pathlib*, *Pickle*, and *ONNX* libraries were used for file management and loading trained models from serialized formats. Additionally, *SciPy* and *StatsModels* were used to perform the statistical tests required for the hypothesis testing procedures.

The project adopted the layered architectural pattern to separate concerns into three main layers: user interface, internal processing, and an interaction layer that manages different data sources. This modular structure improves code organization, promotes reuse, and facilitates unit testing [23].

Figure 3 presents a high-level architectural overview of the implemented solution.

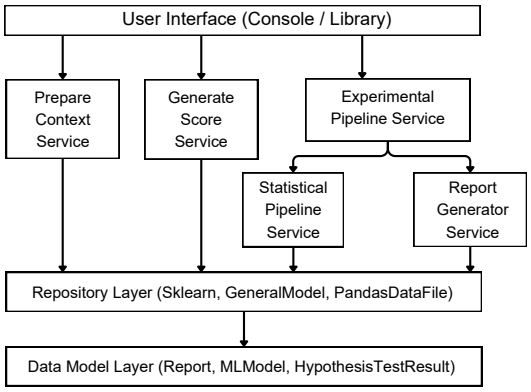

Figure 3: High-level architecture of MLExp.

- **Prepare Context Service:** It acts as a configuration registry where each *context* associates a trained model with a reference to its corresponding test dataset.
- **Generate Score Service:** Evaluates model performance by applying K-Fold Partition over the test data in each context. For each fold, it collects predictions and computes the specified metrics.
- **Statistical Pipeline Service:** Implements an adaptive statistical testing methodology that selects appropriate hypothesis tests based on data characteristics, following the workflow illustrated in Figure 1.
- **Experimental Pipeline Service:** Orchestrates the end-to-end decision process. For each performance metric, it delegates statistical testing to the Statistical Pipeline Service,

interprets the significance results, and supports identifying the best-performing model via median comparison when statistical significance is detected. It aggregates all findings into a structured, general report that includes significance messages, the best model per metric, and detailed test statistics.

4.4 Example of Use

This subsection presents an example of using the MLExp tool. To begin, the user must download the source code from the public GitHub repository [1]. After downloading the source code, the package can be installed with pip, as described in the repository instructions. The package is also available through PyPI. Once the user has the package installed, they can use it by importing the library into their source code or Jupyter Notebooks, or by using it from the command line.

Figure 4 illustrates an example of using the tool declaratively in source code. Initially, the library is instantiated, passing parameters related to the experimentation process. The `add_test_data()` method allows adding a mapped test dataset, and the `add_context()` method adds contexts representing model-test database pairs, as described earlier in Subsection 4.1.

```
from ml_exp import MLExp
ml_exp = MLExp(
    scores_target=["accuracy"],
    report_path="tests/reports/class_ml_experiment",
    report_name="library_with_different_test_data")
ml_exp.add_test_data(
    test_data_name="test_data",
    X_test="tests/data/x_test.csv",
    y_test="tests/data/y_test.csv")
ml_exp.add_context(
    context_name="model_3_sklearn",
    model_trained="tests/models/model_3.pkl",
    ref_test_data="test_data")
ml_exp.add_context(
    context_name="model_0_v2_sklearn",
    model_trained="tests/models/model_0_v2.pkl",
    ref_test_data="test_data")
ml_exp.run()
```

Figure 4: Example of usage from MLExp library.

Upon completion of execution, the tool generates an HTML report that presents the execution results, as well as a JSON file for each performance metric containing the test results displayed in the HTML report (Figure 5). For each performance metric, the report presents distribution measures for the performance data collected in each context, the sequence of statistical tests performed, and the results of each applied test. At the end of the report, a summary indicates whether a significant difference was detected, which metrics were involved, and which model performed best for each metric. Examples of the tool usage and reports are available in the replication package (Section 6), in the `examples.zip` file.

[1] https://github.com/iuryrosal/ml-exp/tree/main

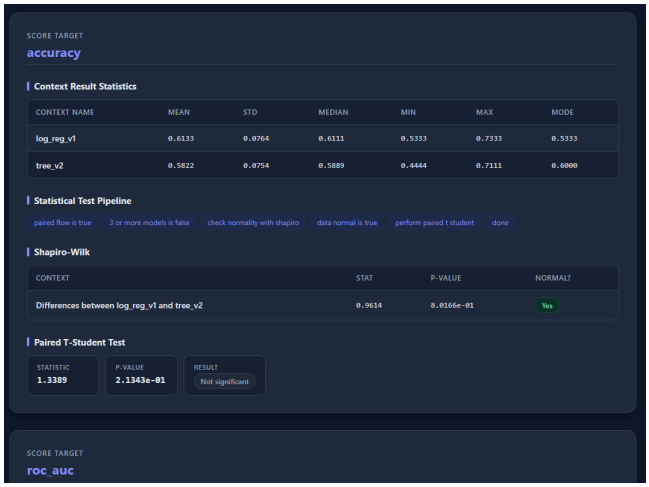

Figure 5: Report Example

The CLI tool enables integrated execution with popular MLOps tools and facilitates its use. Figure 6 shows a summary of this interface.

```
ml_exp [-h] [--n_splits N_SPLITS] [--report_path
REPORT_PATH] [--report_name REPORT_NAME]
[--test_data_paths TEST_DATA_PATHS
[TEST_DATA_PATHS]] [--contexts CONTEXTS
[CONTEXTS]] scores_target
```

Figure 6: Example of usage from MLExp CLI.

5 Comparative Study: Small-Scale Evaluation

A small-scale evaluation typically involves a proof-of-concept application conducted in a controlled environment with limited resources and participation [40]. Since the objective of this study is to evaluate the usage of MLExp in comparison with previously adopted interactive and decentralized procedures, this investigation aligns with the characteristics of a small-scale evaluation scenario.

5.1 Goal and Research Questions

Using the GQM template [35], the goal is defined as follows:

*"Analyze **MLExp** for **characterization** with respect to **its effectiveness in comparing supervised machine learning models and its perceived acceptance** from the viewpoint of **data scientists** in the context of **assessing performance improvement in a real project**".* The following research questions were derived from this goal:

- **RQ1**: When applying the tool to a previously implemented model comparison case, does MLExp lead to conclusions consistent with those originally obtained?
- **RQ2**: What is the perceived acceptance of the tool when used to compare machine learning models?

5.2 Context

A practical case was conducted at a large digital bank to apply the tool and collect feedback from data scientists regarding its usefulness and completeness. The selected case involved four different supervised machine learning model versions (M_1, M_2, M_3, and M_4) developed using *scikit-learn* to support fraud detection in instant transactions. The models differed in their feature sets, consequently affecting how predictions were generated. Before MLExp, two data scientists had manually compared these models using interactive Jupyter Notebooks in conjunction with MLflow and distinct datasets, combined with analyses that explored other scenarios (like using the same dataset). The analysis indicated that versions M_1, M_2, and M_3 showed no statistically significant differences regarding ROC AUC metric, whereas model M_4 exhibited significant differences compared to the others.

As mentioned in Section 2, the comparison between datasets and models, both distinct, explores a descriptive rather than a causal aspect. To address RQ1, MLExp was executed using the same four models under conditions equivalent to those of the original analysis using distinct datasets, integrated with Jupyter Notebook and the MLflow platform. To examine RQ2, the tool and its generated reports were presented to the same data scientists who had previously conducted the manual analysis, thereby enabling feedback on perceived usefulness and future adoption intentions.

5.3 Application

To use the tool, a Jupyter Notebook was created in an isolated workspace within the organization's internal Databricks [2] environment and run on a single cluster. MLExp was installed at runtime via *pip*, along with the required libraries for loading data and models.

The four ML model versions were retrieved from MLflow artifacts stored as *Pickle* files and converted into Python objects. The same procedure was applied to the test datasets, which were also retrieved from MLflow artifacts and loaded as *Pandas DataFrames*. Each test dataset was then iteratively mapped in MLExp and associated with its corresponding trained ML model, creating the experiment contexts. MLExp was subsequently executed, and the resulting reports were generated and presented to data scientists for analysis. The artifacts and code cannot be made public due to confidentiality requirements.

5.4 Results and Discussion

After configuring the models and loading the corresponding test datasets into the Databricks environment, MLExp successfully generated the comparative report. The generated report is available in the replication package (Section 6). The results showed no statistically significant differences in accuracy among the four models. Regarding the ROC AUC metric, versions M_1, M_2, and M_3 showed no statistically significant differences, whereas model M_4 performed significantly differently from the others. These findings were consistent with the conclusion previously reached by the data scientists.

Compared to the previously adopted workflow, the procedure relied on iterative steps (generating model predictions, collecting results, and grouping results), combined with direct comparisons of performance metrics without statistical testing. This entailed considerable manual effort and hindered consistent replication of the entire comparison process, while also complicating productive analysis due to the decentralized nature of the workflows. In contrast, MLExp provided an automated, pre-implemented workflow for statistical comparison, facilitating experimental reproducibility and integration with existing MLflow-based practices with methodological rigor. Furthermore, participating data scientists reported positive impressions regarding the tool's consistency and utility in supporting model comparison activities. A data scientist made the following statement regarding how the report generated by the tool supported decision-making:

> "I'm still analyzing it [results generated by the tool in a small-scale evaluation case combined with other causal analyses] carefully because there are lots of awesome statistical tests, but yes, it totally makes sense to switch models."

The transcripts of the recorded feedback during this case study are available in the replication package (Section 6).

[2] https://www.databricks.com/

6 Concluding Remarks

In this work, we presented MLExp, an open-source tool designed to support the comparison of supervised ML models through automated statistical hypothesis testing and performance benchmarking. The tool enables systematic experimentation by evaluating whether different model versions exhibit statistically significant performance differences, thereby supporting more informed model comparison throughout deployment.

An evaluation in a real-world industry case showed that MLExp reproduced the conclusions previously reached through manual analysis in an MLOps setting, while removing the interactive statistical work those analyses required. This indicates that automating the workflow preserves analytical validity and lowers the effort of applying it consistently across retraining cycles.

Furthermore, participating data scientists reported positive perceptions regarding the tool's usefulness and the consistency of its results. By providing transparent reports that describe the experimental process and statistical outcomes, MLExp can support data scientists in interpreting model comparison results and informing decision-making.

Although this evaluation covers the context of a single fintech and does not aim to offer generalizations for the entire sector, it is based on evidence from a real-world production environment where ML implementation decisions entail direct financial consequences. Future work includes replicating the evaluation across other organizations and domains, as well as exploring comparative scenarios using the same dataset with different models and the same model with different datasets.

Artifact Availability

MLExp source code and other supporting artifacts (example, presentation and small-scale evaluation data) are publicly available at: https://doi.org/10.5281/zenodo.20369248.

References

[1] Antoine Al-Achi. 2019. The student's t-test: a brief description. *Research & Reviews: Journal of Hospital and Clinical Pharmacy* 5, 1 (2019), 1.

[2] Florian Auer, Rasmus Ros, Lukas Kaltenbrunner, Per Runeson, and Michael Felderer. 2021. Controlled experimentation in continuous experimentation: Knowledge and challenges. *Information and Software Technology* 134 (2021), 106551. doi:10.1016/j.infsof.2021.106551

[3] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 1387–1395. doi:10.1145/3097983.3098021

[4] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society: Series B* 57, 1 (1995), 289–300.

[5] Alvah Bittner. 2022. Analysis-of-variance (ANOVA) Assumptions Review: Normality, Variance Equality, and Independence. 28–33. doi:10.47461/isoes.2022_bittner

[6] Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, and Corey Zumar. 2020. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle *(DEEM '20)*. Association for Computing Machinery, New York, NY, USA, Article 5, 4 pages. doi:10.1145/3399579.3399867

[7] ClearML. 2024. ClearML - Your entire MLOps stack in one open-source tool. https://clear.ml/ Software available from http://github.com/clearml/clearml.

[8] Antonio Crespí, Antoni-Lluís Mesquida, Maria Monserrat, and Antonia Mas. 2025. Lifecycle Models in Machine Learning Development. *Expert Systems* 42, 4 (2025), e70029. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/exsy.70029 doi:10.1111/exsy.70029 e70029 EXSY-Nov-24-4780.R1.

[9] Tatiana Escovedo, Marcos Kalinowski, and Thiago Marques. 2024. *Introdução à estatística para ciência de dados: Da exploração dos dados à experimentação contínua com exemplos de código em python E R.* Casa do Código.

[10] Leonhard Faubel and Klaus Schmid. 2024. MLOps: A Multiple Case Study in Industry 4.0. In *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 01–08. doi:10.1109/ETFA61755.2024.10711136

[11] Milton Friedman. 1937. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *J. Amer. Statist. Assoc.* 32, 200 (1937), 675–701.

[12] Satvik Garg, Pradyumn Pundir, Geetanjali Rathee, P.K. Gupta, Somya Garg, and Saransh Ahlawat. 2021. On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps. In *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*. 25–28. doi:10.1109/AIKE52691.2021.00010

[13] Kanwarpartap Singh Gill, Vatsala Anand, Rahul Chauhan, Ruchira Rawat, and Pao-Ann Hsiung. 2023. Utilization of Kubeflow for Deploying Machine Learning Models Across Several Cloud Providers. In *2023 3rd International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*. 1–7. doi:10.1109/SMARTGENCON60755.2023.10442069

[14] Samuel W. Greenhouse and Seymour Geisser. 1959. On Methods in the Analysis of Profile Data. *Psychometrika* 24, 2 (1959), 95–112.

[15] Samuel Idowu, Osman Osman, Daniel Strüber, and Thorsten Berger. 2024. Machine Learning Experiment Management Tools: A Mixed-Methods Empirical Study. *Empirical Software Engineering* 29, 4 (2024), 74. doi:10.1007/s10664-024-10444-w

[16] Ioannis Karamitsos, Saeed Albarhami, and Charalampos Apostolopoulos. 2020. Applying DevOps Practices of Continuous Automation for Machine Learning. *Information* 11, 7 (2020). doi:10.3390/info11070363

[17] Ron Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2* (Montreal, Quebec, Canada) *(IJCAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1137–1143.

[18] Howard Levene. 1960. Robust Tests for Equality of Variances. In *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, Ingram Olkin et al. (Eds.). Stanford University Press, 278–292.

[19] Lucy Ellen Lwakatare, Ivica Crnkovic, Ellinor Rånge, and Jan Bosch. 2020. From a Data Science Driven Process to a Continuous Delivery Process for Machine Learning Systems. In *Product-Focused Software Process Improvement: 21st International Conference, PROFES 2020, Turin, Italy, November 25–27, 2020, Proceedings* (Turin, Italy). Springer-Verlag, Berlin, Heidelberg, 185–201. doi:10.1007/978-3-030-64148-1_12

[20] Katherine R. Maffey, Kyle Dotterrer, Jennifer Niemann, Iain Cruickshank, Grace A. Lewis, and Christian Kästner. 2023. MLTEing Models: Negotiating, Evaluating, and Documenting Model and System Qualities. In *Proceedings of the 45th International Conference on Software Engineering: New Ideas and Emerging Results* (Melbourne, Australia) *(ICSE-NIER '23)*. IEEE Press, 31–36. doi:10.1109/ICSE-NIER58687.2023.00012

[21] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50 – 60. doi:10.1214/aoms/1177730491

[22] Sandeep Bharadwaj Mannapur. 2025. Understanding data drift and concept drift in machine learning systems. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 11, 1 (2025), 318–330.

[23] Azrajabeen Mohamed Ali. 2025. Optimizing Software Architecture: Using the Repository Pattern in Decoupling Data Access Logic. *International Scientific Journal of Engineering and Management* 04 (01 2025), 1–7. doi:10.55041/ISJEM00104

[24] Sergio Moreschini, David Hästbacka, valentina lenarduzzi, Andrea Janes, and Davide Taibi. 2024. Initial Insights on MLOps: Perception and Adoption by Practitioners. (7 2024). doi:10.6084/m9.figshare.26408197.v2

[25] Anita Nanda, Abikesh Mahapatra, Bibhuti Mohapatra, and abinash mahapatra. 2021. Multiple comparison test by Tukey's honestly significant difference (HSD): Do the confident level control type I error. *International Journal of Applied Mathematics and Statistics* 6 (01 2021), 59–65. doi:10.22271/maths.2021.v6.i1a.636

[26] Moses Openja, Amin Nikanjam, Ahmed Haj Yahmed, Foutse Khomh, and Zhen Ming Jack Jiang. 2022. An Empirical Study of Challenges in Converting Deep Learning Models. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 13–23. doi:10.1109/ICSME55016.2022.00010

[27] Eva Ostertagova, Oskar Ostertag, and Jozef Kováč. 2014. Methodology and Application of the Kruskal-Wallis Test. *Applied Mechanics and Materials* 611 (08 2014), 115–120. doi:10.4028/www.scientific.net/AMM.611.115

[28] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. 2022. Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Comput. Surv.* 55, 6, Article 114 (Dec. 2022), 29 pages. doi:10.1145/3533378

[29] Anrafel Fernandes Pereira, Maria Teresa Baldassarre, Daniel Mendez, Jürgen Börstler, Nauman bin Ali, Rahul Mohanani, Darja Smite, Stefan Biffl, Rogardt Heldal, Davide Falessi, Daniel Graziotin, and Marcos Kalinowski. 2026. Attributes to Support the Formulation of Practically Relevant Research Problems in Software Engineering. In *Proceedings of the 3rd IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering* (Rio de Janeiro, Brazil) *(WSESE@ICSE '26)*. Association for Computing Machinery, 1–8. doi:10.1145/3786149.3788303

[30] Federico Quin, Danny Weyns, Matthias Galster, and Camila Costa Silva. 2024. A/B testing: A systematic literature review. *Journal of Systems and Software* 211 (2024), 112011. doi:10.1016/j.jss.2024.112011

[31] Keerthiga Rajenthiram, Milad Abdullah, Ilias Gerostathopoulos, Petr Hnětynka, Tomáš Bureš, Gerard Pons, Besim Bilalli, and Anna Queralt. 2025. Towards Continuous Experiment-Driven MLOps. In *2025 IEEE/ACM 4th International Conference on AI Engineering – Software Engineering for AI (CAIN)*. 89–94. doi:10.1109/CAIN66642.2025.00018

[32] Seldon Technologies. 2026. Seldon Core: An MLOps framework to package, deploy, monitor and manage thousands of production machine learning models. https://github.com/SeldonIO/seldon-core. Version 2. Accessed: 2026-07-15.

[33] Mali Senapathi, Jim Buchan, and Hady Osman. 2018. DevOps Capabilities, Practices, and Challenges: Insights from a Case Study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018 (EASE'18)*. ACM, 57–67. doi:10.1145/3210459.3210465

[34] Samuel S. Shapiro and Martin B. Wilk. 1965. An Analysis of Variance Test for Normality (Complete Samples). *Biometrika* 52, 3-4 (1965), 591–611.

[35] Rini Solingen, Vic Basili, Gianluigi Caldiera, and Dieter Rombach. 2002. *Goal Question Metric (GQM) Approach.* doi:10.1002/0471028959.sof142

[36] Student. 1908. The Probable Error of a Mean. *Biometrika* 6, 1 (1908), 1–25.

[37] Yuan Sun, Qiurong Song, Xinning Gui, Fenglong Ma, and Ting Wang. 2023. AutoML in The Wild: Obstacles, Workarounds, and Expectations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. ACM, 1–15. doi:10.1145/3544548.3581082

[38] Kumar Venkateswar. 2019. Using Amazon SageMaker to Operationalize Machine Learning. USENIX Association, Santa Clara, CA.

[39] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83.

[40] Claes Wohlin and Austen Rainer. 2022. Is it a case study?—A critical analysis and guidance. *Journal of Systems and Software* 192 (2022), 111395. doi:10.1016/j.jss.2022.111395

[41] ZenML GmbH. 2026. ZenML: One AI Platform from Pipelines to Agents. https://github.com/zenml-io/zenml. Version 0.94.4. Accessed: 2026-07-15.