Continuous Integration for Machine Learning Experiments Reproducibility: a Practical Study

A. M. Andrade
Federal University of Ceará
Crateús-CE, Brazil

M. B. Pereira
Federal University of Ceará
Crateús-CE, Brazil

S. H. S. Silveira
Federal University of Ceará
Sobral-CE, Brazil

F. I. F. Linhares
Federal University of Ceará
Sobral-CE, Brazil

A. H. O. Neto
Federal University of Ceará
Sobral-CE, Brazil

R. M. C. Andrade
Federal University of Ceará
Fortaleza-CE, Brazil

I. L. Araújo
Federal University of Ceará
Fortaleza-CE, Brazil

ABSTRACT

The development of a Machine Learning (ML) model depends on many variables in its training. Both model architecture-related variables, such as initial weights and hyperparameters, and general variables, like datasets and framework versions, might impact model metrics and experiment reproducibility. An application cannot be trustworthy if it produces good results only in a specific environment. Therefore, in order to avoid reproducibility issues, some good practices need to be adopted. This paper aims to report a practical experience in developing a machine learning application adopting a workflow that assures the reproducibility of the experiments and, consequently, its reliability, improving the team productivity.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; • **Computing methodologies →** *Machine learning*.

KEYWORDS

Machine Learning Systems, Software Development Process, Software Engineering, Intelligent Software Engineering, MLOps

ACM Reference Format:
A. M. Andrade, M. B. Pereira, S. H. S. Silveira, F. I. F. Linhares, A. H. O. Neto, R. M. C. Andrade, and I. L. Araújo. 2021. Continuous Integration for Machine Learning Experiments Reproducibility: a Practical Study. In *Proceedings of Brazilian Workshop on Intelligent Software Engineering (ISE' 21).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.5753/ise.2021.17279

1 INTRODUCTION

The reproducibility of computational experiments in scientific works is a fundamental challenge in academia and industry. According to Beaulieu-Jones and Greene [4], scientific results can often only be reproduced with help from the original authors. Many factors impact the reproducibility of the software-based experiment, such as the development environment, i.e., hardware and software specifications and versions of employed Application Programming Interface (API). ML-based experiments, whether from industry or academia, are even more complex to reproduce due to many variables that impact models performance metrics, such as dataset samples, models' hyperparameters, and dependency of random parameters used to initialize and update models.

The work of Amershi et al. [1] reports practices that Microsoft teams used in ML projects. The authors state that the introduction of ML capabilities in software developed with an agile process comes with three main complications: the data management; model development require rare skills; modularization is impaired. All these three factors impact reproducibility in the sense that the datasets and model versions need to be trackable, other teams need to be capable of reproducing and adapt the experiment. It is hard to achieve that if the software has a high coupling.

All the factors previously described related to reproducibility impact the reliability of scientific work considering the risk of being outlined by its peers. According to Pineau et al. [5], the establishment of methodologies to ensure reproducibility of scientific works in ML field is one of the trends in academia since the same results could be obtained by experimenting with the same conditions. This is also essential in an industry ML project, which must follow a scientific method process of hypothesis, experiment, and resulting assessment.

To handle the reproducibility problem of ML experiments, this proposes a work process composed of pipeline automation tools and a well-defined workflow to perform the experiments and models optimization. In the development process, the following automation tools were used: the Gitlab[1] Continuous Integration, Deployment or Delivery (CI/CD) to execute an experiment pipeline, Data Version Control (DVC)[2] to handle the datasets versioning, Continuous Machine Learning (CML)[3] to provide reports automatically, Git to version control the software and Docker to run the container that supports the pipeline execution.

This article presents the work process applied in the development of a ML-based Fall Detection System (FDS), executed by academy and industry in partnership, with the description of tools, methods, and expected results. The content is divided as follows. Section 2 presents a literature review related to this work. It shows details of our proposed approach for the development of this research in Section 3. Section 4 exhibits major results of our work. Section 5 presents a discussion of the main issues found in project development. Finally, conclusions are presented in Section 6.

2 RELATED WORKS

The literature reports some issues related to reproducibility in computational experiments and ML projects as a significant question that could compromise the work.

In [4], Beaulieu-Jones and Greene developed a general framework for computational experiments using Docker and Continuous Integration (CI) practices. The authors also describe the recurrent

[1] https://gitlab.com/
[2] https://dvc.org/
[3] https://cml.dev/

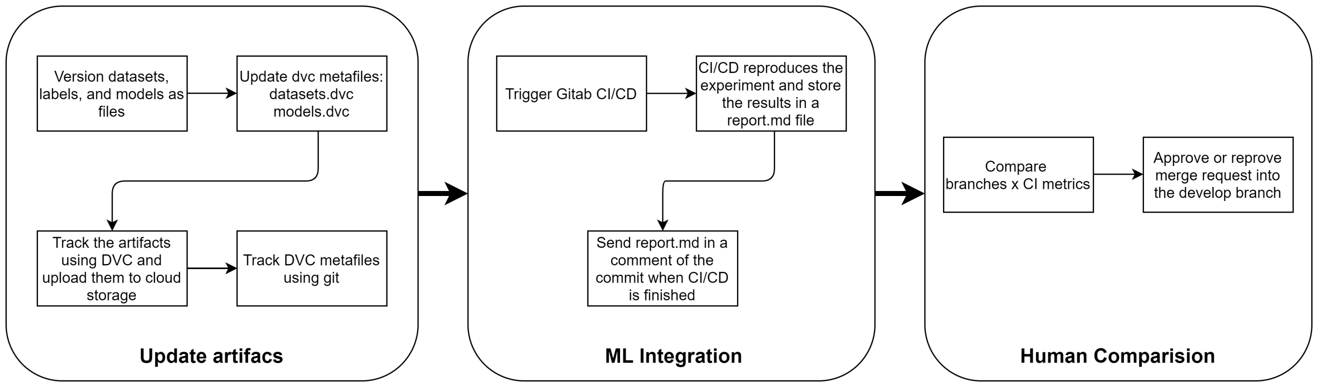
Version datasets, labels, and models as files
Update dvc metafiles: datasets.dvc models.dvc
Track the artifacts using DVC and upload them to cloud storage
Track DVC metafiles using git
Update artifacs
Trigger Gitab CI/CD
CI/CD reproduces the experiment and store the results in a report.md file
Send report.md in a comment of the commit when CI/CD is finished
ML Integration
Compare branches x CI metrics
Approve or reprove merge request into the develop branch
Human Comparision

Figure 1: Process for verification of reproducibility.

problem of the lack of reproducibility and its impacts on scientific papers. Their work has several intersections with ours, but it does not specifically address the problems of the ML process, such as its random nature. Our process also involved dataset versioning, hyperparameter search and optimization, automatic reporting with CI/CD tools, among others.

Wan et al. [6] presented a review of software engineering methods applied in MLmodel development. The review points some directions to choose the most suitable method to implement. They also defend the thesis that the lack of a software engineering method in ML development can cause several issues, including the possibility of reproducing the experiment. In our work we adopted some practices listed in [6], such as data visualization, pre-processing, cleansing and keep the initial state of experiments. Also, we started the project with the mindset that the ML process is inherently uncertain, finding out either intuitively, by implementing in ptactice or foreseeing possible issues.

Wan et al. has also presented some practices in the ML software engineering. The main difference between our approach and those used by ML practitioners is that the tasks were well defined, although an accurate plan is more challenging to implement in such kind of process.

Ashmore et al. [3] presented a survey related to ML software development practices containing directions to define a life cycle for ML models and some security requirements for each stage. For reproducibility, the most relevant activities are: Preprocessing for the Data Management phase; for the Model Verification phase, they recommend formal methods; for the Deployment phase, the traditional software engineering can handle integration and monitoring. The Model Training phase still needs some study to improve reuse and interpretation. In our work, we implemented most of the sugestions, however, the formal methods in the Model verification phase is still to be done, even though we have an well defined approach to it.

The work of Argesanu and Andreescu [2] presented an analysis of the problems that occur in the ML life cycle related to automation and reproducibility. The authors develop a framework to deal with these issues and present a study case applying the framework in an image processing problem. Their framework is batch-inference oriented, with some differences to ours, but still similar in several points, such as activities, automation tools, versioning and containers.

3 METHOD

We applied the proposed workflow in a ML project that developed a FDS using data from wearable devices. The inference solutions consist of a model deployed in two different environments: cloud server and edge node. The project had six months of duration an the team was composed by 6 members of ML team, 3 members of service team, 1 member of QA team and 2 managers.

Our proposal focused on the reproducibility of experiments, model maintenance, and validation of the results at the beginning of project development. In order to achieve these goals, we selected the following tools: the Data Version Control (DVC) to version ML artifacts labeling each new one; GitLab Continuous Integration, Deployment or Delivery (CI/CD) to handle integration process; and the Continuous Machine Learning (CML) to auto-generate reports containing metrics for ML models, which were attached to each commit.

Before the process start, the team performed a Grid Search in order to find or optimize the model Hyperparameters. The dataset was randomly separeted by training and test.

Although GitLab repository is a great option for versioning project source codes, it is not recommended to versioning datasets and models artifacts because of the expected big size of these artifacts. Hence, DVC is responsible for version datasets and models and also create metafiles for each project artifact. Instead of versioning files, GitLab tracks only these metafiles, which DVC changes every time an artifact is updated. Therefore, DVC sends updated artifacts to an external storage service and updates the respective metafile. Finally, developers commit the changes on the metafiles.

We present in Figure 1 a visual description of each step of our proposed process for this work. Firstly, at the Update artifacts step, the datasets and models architecture are commited to the DVC versioning system and its metafiles are updated and tracked through Git. So, ML Integration step, whenever developers update codes or artifact versions in GitLab, the CI/CD is triggered to reproduce the current experiment in a Docker environment. After that, the

CML generates a report with all the relevant metrics of the experiment. In the Human Comparison step, the new self-generated results help the development team to create merge requests of new model and software versions to be analyzed by technical leaders by comparing report results generated by CML with the results on the README.md, file generated in pasts merges.

Then, developers created git tags in the repository for each new stable version of our model and deployed model on cloud and edge proposed environments.

In order to verify the hypothesis that the application of the proposed workflow can detect reproducibility issues, we worked as defined in Figure 2. Firstly, the team needs to configure the environment and the CI/CD pipeline. Moreover, an interactive process performs the generation, train, and execution of new models. The team leader is responsible for registering and reporting the reproducibility issues to the team. Therefore, the team performs fixes and verification if the experiment is correctly reproduced automatically by the Gitlab CI/CD and reviewed by the team leader.

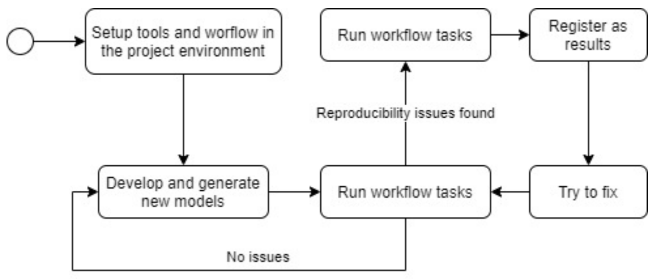
Setup tools and worflow in the project environment
Run workflow tasks
Register as results
Reproducibility issues found
Develop and generate new models
Run workflow tasks
Try to fix
No issues

Figure 2: Reproducibility Issues Identification

4 RESULTS

The studied project consists of intelligent models deployed on cloud or edge, to detect falls based on accelerometer and gyroscope sensor data. The adoption of the proposed code integration workflow makes it possible to reproduce ML experiments on their stable versions considering the correct use of datasets, pre-processing methods, and model hyperparameters in different environments, generating the same models and measures.

The iterative approach provided a gradual increase in the reproducibility, since it allowed to run the pipelines and the experiment as a whole in different environments. The total amount of Merge Requests is 44, which 38 was merged and 6 was closed. Through these Merge Requests, was runned 469 pipelines from which 240 has passed. The remaining pipelines were either canceled or some issue prevented it to succeed. At the beginning of the project, a rate of 50.02% of the jobs succeeded and several has failed due to environment issues. At the end of the project, a rate of 87.90% of jobs has succeded. The pipelines and experiments are running in different environments with the same results, which indicates a increase in reproducibility.

In the following subsections, we present the main issues identified by the use of the CI workflow and how they were solved.

4.1 Incompatibility with frameworks versions

Many ML libraries compatible with Python language have a large number of versions that may have internal variations in their functions from one to another. Thus, the slightest variation in the library version can compromise system integration compatibility, generating errors that require some time to find the solution by the team. The team reported failures regarding the execution of Gitlab CI/CD jobs compilation while the developer's environment works properly. The use of a virtual environment of *virtualenv tool*[4] solved this issue, with the configuration of a set of requirements in a *requirements.txt* file that stores all the Python dependencies and their respective versions, enabling to have similar environments for development and CI. These adjustments solved compiling issues for following Gitlab CI/CD job executions. Moreover, other possible errors regarding specific libraries versions could be verified by compare metrics generated by the Gitlab CI/CD job and the metrics obtained in the development environment.

4.2 Random Seed issues

Seeds are crucial parameters to produce pseudo-random numbers, which, in general, are used to initialize ML models before training. ML models, even if trained under the same conditions, cannot produce the same metrics if chosen seeds are different. Since random seeds change every time in ML training process thus, outcomes might not be as expected because of the difference of seeds. To avoid that, developers had to set seeds fixed for all experiment executions. In the developed application, random seeds can impact the dataset partitioning in train and test and the random initial weights and biases of a deep learning model.

4.3 Multiple Datasets issues

The dataset used in ML training and validation process is composed of multiple datasets with compatible sensor data and class outputs after performing cleaning and standardization. The generation of a single dataset from multiple sources can impact the experiment's reproducibility because of procedures to handle and standardize data types. To solve the issues, DVC was used to version not only the original datasets but also the final dataset used for training. Once the training code uses a specific dataset (DVC allows organize datasets by name and version), if the developer correctly commit the code and DVC metadata files, the CI training job selects exactly the same dataset used by the developer and repeat the experiment with the same dataset.

4.4 Hyperparameters variability

The optimization of ML models requires the adjustment of hyperparameters, which are, e.g., the number of kernels of Support Vector Machine (SVM) or estimators in Random Forest. Developers need to track hyperparameters for each generated model because of the variation of models' metrics. The tracking of hyperparameters list in a separated file and its use in the model training solved this issue.

[4]Python Virtual Env - https://docs.python.org/pt-br/3/library/venv.html

5 DISCUSSION

Our approach to ML development provides an automated pipeline for CI, which helps to achieve the model's reproduction while increases the ML system reliability. As stated in [6], the main change from traditional software engineering to ML development is the uncertainties in the process, which suggests that a well-defined pipeline does not guarantee system reproducibility. The authors in [3] presented crucial factors to apply in each process step with high priority and risk requirements. Therefore, adopting practices described in this work with the right mindset improves the system's reproducibility.

As described in Section 3, the ML models for the proposed system had to be able to be deployed in two environments: cloud and edge. The ML lifecycle used in the evaluated project included tests in these two architectures, so reproducibility was a requirement since the very beginning of the project. The project team reported that, before this work, they had adopted a similar flexible lifecycle integration with a focus on reproducibility as proposed by Beaulieu-Jones and Greene [4]. This project's main concern is to make the models reproducible at the cloud, edge, and test systems. However, its main objective is to store all needed artifacts to reproduce an experiment with the suggestion of Docker images as a central component. Although the proposed approach focuses on the ML development, there is a big intersection with this pipeline and the one presented in [4] with the difference that this work requires more components to handle ML specific tasks.

The authors in Argesanu and Andreescu developed a platform and a study case to deploy a ML system with less effort. As in [4], Argesanu and Andreescu used Docker images to preserve the state of the software used in the experiment and achieve identical results in deployment and production. Both of the works generate the same results in a different environment using CI/CD automated tools. Thus, it has its limitations for a final product, such as that results can only be recovered from a Docker image, which does not go entirely to a production environment. Also, in this work, the models and the results were reproduced outside a container, making access to the resulting artifacts, i.e., models and datasets, more effortless.

Beyond that, the proposed CI increased the productivity because, before our approach was adopted, the reproducibility was verified by the reviewer in its environment. After we apply our approach, the Merge Request review tool allowed an imediate check of the reproducibility, reducing the time spent reviewing. The majority of the related problems had a quick solution, decreasing the time and the need of team meetings.

The adaptation of software engineering methodologies in ML projects that allows reproducibility in industry projects can increase the productivity. This kind of process accepts the uncertain nature of the process and deals with it in the best possible way. That means to deal with hypothesis, testing and adaptation cycle with direction and objective. Also, this can generate ML based system applicable on high priority and ensurance requirements.

For academy, the main points of contribution are the pipelines and reproducibility for computational simulations, which still is a weak point of these works [4]. Another contributions are the reliability that depends on the reproducibility, implying that the academic ML works would need less retraction.

An improvement identified by the project team was that the verification of reproducibility of an experiment depends on a human that needs to analyze the metric and reject the merge/pull request. This approach's main restriction is its dependency on third-party software, limiting its application on time as the technology keeps evolving. Thus, the work would need to be updated indefinitely as long as the used technologies have no more extended support. The automation of this task is feasible by the use of automated testing frameworks integrated into the Gitlab CI jobs.

Our work deals with small-sized datasets. In Big Data or Computer vision scenarios, the use of DVC may not be suitable. The same is valid for problems with the training of the model has a high time cost. In our case, the dataset, composed only of sensor data, has 54.2 MB, and the training time was less than five minutes.

6 CONCLUSION

This work presented a practical workflow to help the development of a ML project considering integration, maintenance, and reproducibility requirements. During the application of the proposed process, we found similar issues to those described in the literature. The well-defined life cycle helped the development team reduce time spent with meetings because of the streamlines in the reproduction of experiments. Also, the versioning system and continuous integration made the process trackable, allowed to find a specific dataset, model, or experiment version with ease through the tag system.

As a proposal for improvements of ML workflow strategies, it is necessary to establish criteria to automatically stop continuous integration based on the system's metrics performance and to develop a notification system to alert developers about failures or issues in pipeline execution. Despite the need for improvements, the proposed CI workflow presented more security in development and organization, as well as helped the team to manage the uncertainty of ML projects and improved productivity.

REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[2] Adrian-Ioan Argesanu and Gheorghe-Daniel Andreescu. 2021. A Platform to Manage the End-to-End Lifecycle of Batch-Prediction Machine Learning Models. In *2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. 000329–000334. https://doi.org/10.1109/SACI51354.2021.9465588

[3] Rob Ashmore, Radu Calinescu, and Colin Paterson. 2021. Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. *ACM Comput. Surv.* 54, 5 (05 2021), 39.

[4] Brett K. Beaulieu-Jones and Casey S. Greene. 2017. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology* 35, 4 (01 Apr 2017), 342–346. https://doi.org/10.1038/nbt.3780

[5] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. 2020. Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program). arXiv:2003.12206 [cs.LG]

[6] Zhiyuan Wan, Xin Xia, David Lo, and Gail C. Murphy. 2019. How does Machine Learning Change Software Development Practices? *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2019.2937083