SnapperML: A python-based framework to improve machine learning operations

Antonio Molner\textsuperscript{a}, Francisco Carrillo-Perez\textsuperscript{b}, Alberto Guillén\textsuperscript{a,\ast}

\textsuperscript{a} Computer Engineering, Automatics and Robotics Department, University of Granada, Spain
\textsuperscript{b} Stanford Center for Biomedical Informatics Research (BMIR), Stanford University, School of Medicine, USA

ARTICLE INFO

Keywords:
MLOps
Machine learning
Deep learning

ABSTRACT

Data Science has emerged as a vital discipline applicable across numerous industry sectors. However, achieving reproducibility in this field remains a challenging and unresolved problem. Additionally, transitioning trained models from development to production environments often proves to be a non-trivial task. In this study, we propose SnapperML, a comprehensive framework designed to address these issues by enabling practitioners to establish structured workflows that facilitate result reproducibility. Leveraging DevOps techniques, SnapperML ensures seamless model deployment from the lab to production, mitigating the risks associated with compatibility issues and model selection errors. The framework enables meticulous tracking of every aspect of model training, including hyperparameter selection, tuning, and distributed training on a server. By offering a suite of tools for model tracking and optimization, SnapperML presents a promising solution to the reproducibility challenge in the field of data science.

1. Motivation and significance

How to deal with the problem of reproducibility is a hot topic that has been discussed recently by many authors [1–4]. Furthermore, in the academic world there is a big consensus on the reproducibility crisis. Baker et al. [4] showed that 90% of researchers answered “Yes” to the question: Is there a reproducibility crisis? In addition, the 52% of the scientist considered that it was a significant crisis whereas the 38% thought it was a slight crisis. This problem is not intrinsic for the academic world but it is also present in industry [5]. Some researchers are proposing checklists and guidelines to improve it, however, they are not yet widely accepted as standard [6,7]. Machine Learning (ML) techniques are being widely adopted both in science research and industry, specifically those sets of techniques within the Deep Learning (DL) field [8]. Given the stochastic nature of DL techniques, reproducibility is expected to be a key factor when optimizing procedures, testing, debugging and providing quality and confidence.

Traditional reproducibility in computer science can be defined at three different levels: computational (related to code hardware and implementation details), empirical (information about the experiments and observations), and statistical (details on the statistical parameters...
and thresholds chosen (p-value, i.e.,)). These three levels are not directly applicable to ML as they overlap in such a way that it is very difficult to achieve the same final result if all of them are not treated carefully. Even if the practitioner is using the same code, hardware configuration, virtualised or deployment environment, and all the information about the experiments and observations, the results obtained can vary from the original ones. This might be due to other factors, such as if the dataset were shuffled with a different random seed. Thus Stodden et al. [9] define more general terms from the perspective of the reviewer with the research as subject.

Thus, the study of this problem has led to a whole new field in industry, MLOps, that approaches the problem of maintaining ML models in production reliably and efficiently [10]. While this practice is widely extended in industry, it has not been adopted at the same level in research. Nevertheless, scientific research can greatly benefit from MLOps practices, as the solution to the reproducibility crisis that is suffering.

In this work, we propose a novel open source MLOps framework, SnapperML, that allows to efficiently track experiments, log hardware details, perform stochastic control among other features. Our framework focuses mainly in the computational aspects. Nevertheless, it also includes the capacity of controlling the statistical aspect of ML as well as implicit empirical information if the data set files are included in the workflow. Table 1 makes a comparison available solutions versus the one proposed in this work.

2. Software description

2.1. Software architecture

In this work, we propose SnapperML to fill the gaps present in the MLOps field. We decided to develop the framework to avoid code modification as much as possible. We achieved this goal by defining proper configuration files and using Python default features. Following practitioner’s workflows, the fact of being able to achieve reproducibility is difficult unless random elements are tracked or defined beforehand. Furthermore, if new configuration parameters need to be defined, not having to modify existing code seems the most logical behavior to avoid introducing bugs or augmenting the technical debt.

Thus, the generated config files should be as readable and as standardized as possible. Although JSON seems a good alternative as it is the web standard, YAML configuration files were chosen due to their good structure and readability. Another benefit of using these files is the easy access to the contents once parsed in Python. This election is becoming a standard even for Linux/GNU files like netplan’s configuration files for network interfaces [11] in some distributions as well as other widely adopted software like Ansible [12], Kubernetes [13], Terraform [14], Jenkins [15]. One notable advantage of employing this experiment configuration is its compatibility with version control software such as Git. By incorporating each experiment configuration and its results into the version control system, it ensures traceability and reproducibility.

Python has emerged as the go-to language for the majority of Data Scientists and ML practitioners, gaining popularity across various disciplines. As of March 2021, it ranked just after C and Java [16]. One of Python’s strengths lies in its decorator functionality, which enables the seamless addition of logging or tracking capabilities, as well as effortless integration with other technologies.

SnapperML has been design as a layer that integrates several modules and tools through a Command-Line Interface (CLI), programmed in Python, that orchestrates all the other components in a transparent way for the user. The metadata for the Python package is detailed in The main components of the application are

- MLflow: is a platform which focuses on the lifecycle of machine learning applications. The functionality used is the tracking of the training process as well as the model versioning.
- Optuna [17]: is defined as an optimization framework that allows users to automate the search for hyperparameters.
- Ray: allows developers to easily distribute and scale the computing needs for Python workloads.

The four pillars are depicted in Fig. 1 where it is possible to identify the cited CLI which communicates with the experiment engine MLflow. The experiments are executed in an efficient way by means of the Ray (https://docs.ray.io/) API for distributed applications, requiring zero code changes. Both the experiments and its parallel execution can be monitored and controlled by their respective web-user interface. The last element added to Snapper is the Optuna software, which requires its own database. This allows to save time when retraining the models and has many of the state-of-the-art algorithms already implemented.

Thanks to this modular implementation, very little effort is required from the user to use the framework in a local setup or in a remote cluster. This allows practitioners to have distributed training, experiment tracking, visualizations and automatic control over the random seeds in their experiments. The Job runner takes care of calling and integrating the code developed by the practitioner, which just need to call the script after configuring the experiment using a YAML file.

2.2. Software functionalities

2.2.1. Experiment tracking

Jobs (single experiments and groups of them) can be easily tracked to keep track of useful information regarding the experiment. To do so, it is only required to instrument the source code with the @job decorator as mentioned in the previous section. Among the details that are recorded for each job, we have:

- Line arguments: if the experiment receives a number of arguments through the CLI or the configuration file (see Listing 2, these are recorded so there are no doubts about their value.

<table>
<thead>
<tr>
<th>Feature</th>
<th>SnapperML</th>
<th>MLOps</th>
<th>Reproxp</th>
<th>Polysign</th>
<th>Sacred</th>
<th>CometML</th>
<th>Superscaler</th>
<th>Google AI</th>
<th>Azure ML</th>
<th>Neptune</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment tracking</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Log hardware details</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic stochasticity control</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dependence logs</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Native support for HPOa</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Project package</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Running in Cloud or cluster</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Open Source</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Hybrid</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

a HyperParameter Optimization.
Antonio Molner et al.

Fig. 1. SnapperML’s framework overview. The framework is controlled using a Command-Line Interface (CLI) and allows the communication with the different modules. Experiments are tracked using the mlflow experimental engine, that allows to save important information about the experiment (metrics, parameters, info and logs). The Ray module allows to seamlessly perform distributed training, and the Optuna module provides hyperparameter search capabilities. All the information can be easily accessed using the implemented web interfaces.

Fig. 2. Web interface for Jobs analysis after execution.

- **Running System Details**: this includes both hardware and software information such as CPU exact model, GPU models and drivers used.
- **Git commit hash**: by keeping track of this parameter, it is possible to know exactly which code was running for this experiment. It can be easily restore by using these value even after posterior modifications.
- **Metrics and artifacts/models**: if there is an output with one or several metrics logged explicitly by the user, they can be included as well using SnapperML without writing them to a file. Furthermore, it is possible to keep track of the artifacts (models) that were generated during the script.
- **Console output**: all messages printed out to console are stored in a file which is sent to the server and will be accessible afterwards.

### 2.2.2. Web interface and execution monitoring

Once the Jobs have been executed, all the tracked information becomes accessible through a user-friendly web interface. While it is technically feasible to connect to it and directly query the databases, utilizing the web interface is the recommended method for most users. Fig. 2, exemplifies an experiment where all the tracked elements are clearly visible at a glance. Additionally, during the execution of experiments, users can monitor the machine’s status through another dedicated user interface. To access these interfaces, users need to indicate the host running the service and the corresponding ports. This intuitive approach ensures convenient access to the tracked information, and facilitates monitoring of machine status throughout the experiment execution.
2.2.3. Hyperparameter optimisation

SnapperML seamlessly incorporates Optuna [17] for automated hyperparameter optimization, empowering users to include it as part of a Job definition. This feature proves highly advantageous for fine-tuning models by specifying a range of possible values, which can be derived from metaheuristics or expert knowledge. The practitioner only needs to define the desired interval in the YAML file and iterate through the potential values within the code.

2.2.4. Distributed execution using Ray

The integration of Ray in the SnapperML framework allows users to take advantage of all the hardware resources available at their computing center. This is specially important when using the auto-hyperparameter optimization, as it can reduce significantly the computational cost.

2.2.5. SnapperML’s knowledge center/data management

During the execution of Jobs, it becomes necessary to store information regarding the experiments and their progress. Certain components integrated within SnapperML already employ databases as a recommended approach for storage. However, there are certain aspects that need to be considered. Various adjustments for each of the following subsections can be made through shell environment variables, or by including them in a .env file. This flexible configuration allows users to customize the settings according to their specific requirements.

2.2.6. Using databases vs storing locally

A database service (MySQL/PostgreSQL) can be configured to interact with MLflow and Optuna but that implies an overhead which is not easily justified. It is true that this service can be run remotely but, if it is running in the same machine than is executing the experiments, there could be a waste of resources. On the other hand, the use of such service would require humanware extra costs to configure and maintain the service.

Optuna allows to store the data straight into memory without the need of an intermediate service. This is the most optimal approach in terms of computational costs, although it presents a flaw: distributed execution is not allowed anymore, so resources could be wasted in a multiCPU/GPU cluster architecture. As a compromise solution, the most flexible approach would be to set an SQLite database in a distributed filesystem environment so all nodes could update information.

2.2.7. Using containers and volumes

If reproducibility is the main goal when deploying the Jobs, the approach based on setting up containers is the one that minimizes the possibility of adding entropy due to misconfigurations. On the other hand, this would consume more resources from the machine running the framework. A possible configuration could be based on three containers: MySQL image to store MLflow information, PostgreSQL image to store Optuna’s data and a container running the service of MLflow. This scheme can be easily reproduced in any computer or, furthermore, be deployed in any cloud service of the current major providers. Listing 1 shows a possible deployment using docker-compose.

From the user perspective, encapsulating functionalities by using containers allows practitioners to easily scale according to their needs. For example, if the data set increases significantly, more pods can be deployed for the training stage using Ray. If the search for better hyperparameters is required to be wider, more pods can be assigned to Optuna module. The computing resources for storing model information are not prone to be increased, however, to have a separate storage volume and service can make the model deployment and tracking easily integrable within a pipeline following the Continuous Integration/Continuous Deployment paradigm.

Listing 1: Docker-compose file to carry out a deployment separating the storage used for the hyperparameter search and the one required to store the models training process and versions.

3. Illustrative examples

Here, we are going to present the example on how to train a SVM using the framework. The values of the job are provided through the YAML configuration file. In Listing 2, there is an example for an script that would train an SVM using the Python code available in Listing 3. In this example, all the parameters under the label “params” included in the YAML file will be stored in the database for future reference, as well as the variable “accuracy” which is returned by the script. Please, note that the main difference respect a regular script are the @job decorator and the definition of the YAML file.

name: "SVM with RBF kernel"
kind: ‘experiment’
params:
  C: 0.5
  kernel: ‘rbf’
  gamma: ‘auto’
run:
  - model/ing/3ain_train_svm.py

Listing 2: Job configuration (an experiment in this case) file example.

Listing 4 showcases an example where the parameter C is defined within a range of permissible values. This flexibility enables researchers to effortlessly modify the interval range without altering the underlying code, thus facilitating the definition of novel metaheuristics within a

version: ‘3’
services:
  mlflow_storage:
    image: mysql
    env_file:
      - ../.env
    volumes:
      - mlflow_storage_data:/var/lib/mysql
    ports:
      - "$(MYSQL_PORT:-3306):3306"
  optuna_storage:
    image: postgres
    env_file:
      - ../.env
    volumes:
      - optuna_storage_data:/var/lib/postgresql/data/
    ports:
      - $(POSTGRES_PORT:-5432):5432

mlflow:
  build:
    context: .
    ports:
      - "5000:5000"
    volumes:
      - ../:/mnt
eentrypoint: |
    bash -c "./wait-for mlflow_storage:$(MYSQL_PORT:-3306) -- /start_mlflow_server.sh && /bin/bash"
depsends_on:
  - mlflow_storage
  - optuna_storage
stdin_open: true
tty: true
env_file:
  - ../.env
volumes:
  mlflow_storage_data:
  optuna_storage_data:

Listing 4: Docker-compose file to carry out a deployment separated

SoftwareX 26 (2024) 101648
main(C, kernel, gamma='scale'):

def @job
from snapperml import ...

Antonio Molner et al.
only modification that is necessary is to add the @job decorator on the

function.

new meta-pipeline. The optimization module supports both cutting-
edge algorithms, such as TPE and CMA-ES, as well as traditional
methods like GridSearch and RandomSearch [18]. This offers a com-
prehensive array of options to suit different optimization requirements.
In the example of Listing 4, there are 12 experiments defined that will be
executed using a total of 4 CPUs. Again, we would like to emphasize
that specifying this parameters on the configuration file side avoids any
modification of the code, and allows to limit the resources in shared
machine environments.

name: 'SVM'
kind: 'group'
num_trials: 12
sampler: TPE

Listing 3: Experiment Python script example to train an SVM. The
only modification that is necessary is to add the @job decorator on the

function.

4. Impact

Reproducibility is a growing concern in both the scientific and
industrial communities. Distinguishing genuine discoveries from false
leads can be unsettling and may lead researchers down the wrong path [1,4,9]. Likewise, within a company, results often remain iso-
lated within specific departments, preventing other teams from benefit-
ing from valuable findings. Consequently, SnapperML offers a com-
prehensive solution for organizing, executing, and sharing ML experiments
in a transparent manner.

To our knowledge, existing tools lack the necessary deployment
and packaging functionalities to ensure experiment reproducibility.
While Feurer et al. [20] extensively covers research on automated ML
methods, systems, and challenges, it fails to address the crucial aspect
of reproducibility. Despite the use of well-known tools like WEKA,
Sklearn, TPOT, Auton-Net, and Automatic Statistician, no guidelines
are provided for enabling third parties to reproduce the experiments
and achieve consistent results. Unlike other frameworks that demand
extensive code modifications, we have leveraged Python’s decorator

functions to minimize the changes required from users, facilitation
the adoption of the framework.

Our software has a total of 15 stars in Github, and has been used for
multiple degree thesis works in the University of Granada. Further
work it is needed to increase the impact of the software in a broader
community.

5. Conclusions

In this work we have proposed a new framework, SnapperML, to
enhance MLOps both in industry and research labs. Our framework
is the only one available in the MLOps space that allows to track all
the different aspects of the ML development cycle, from metrics
to hyperparameters search. In addition, thanks to the minimal code
changes required, it can be quickly implemented in on-going projects.
In future work we would like to develop a more polished web interface
that can bring easy of use to not so technical users that are embracing
machine learning as a regular tool.

CRediT authorship contribution statement

Antonio Molner: Investigation, Methodology, Software, Visualiza-
tion. Francisco Carrillo-Perez: Conceptualization, Investigation,
Methodology, Validation, Visualization, Writing – original draft, Writ-
ing – review & editing. Alberto Guillen: Conceptualization, Formal
analysis, Funding acquisition, Investigation, Methodology, Project ad-
ministration, Software, Supervision, Visualization, Writing – original
draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial
interests or personal relationships that could have appeared to
influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This research has been possible thanks to the support of projects:
PID2021-128317OB-I00 and PID2019-104676GB-C32 (Spanish Min-
istry of Economy and Competitiveness –MINECO–) as well as Circuits
And Systems for Information Processing (TIC-117) Research Group.

References

[2] Freire Juliana, Fuhr Norbert, Rauber Andreas. Reproducibility of data-
oriented experiments in e-science (dagsuhln seminar 16041). In: Freire Juliana,
[3] Freire Juliana, Bonnet Philippe, Shasha Dennis. Computational reproducibility:
State-of-the-art, challenges, and database research opportunities. In: Proceedings
of the 2012 ACM SIGMOD international conference on management of data.
SIGMOD ’12, New York, NY, USA: Association for Computing Machinery; 2012,
p. 593–6.
portable MLOps, reusable automation actions and reproducible benchmarking.
2020.
[6] Pineau Jollie, Gilbey Elizabeth. This AI researcher is trying to ward off a
Begelezimer Alina, d’Achêl Buc Florence, et al. Improving reproducibility in
machine learning research (a report from the NeurIPS 2019 reproducibility
[9] Strodden Victoria, Borwein Jonathan, Bailey David H. Setting the default to


