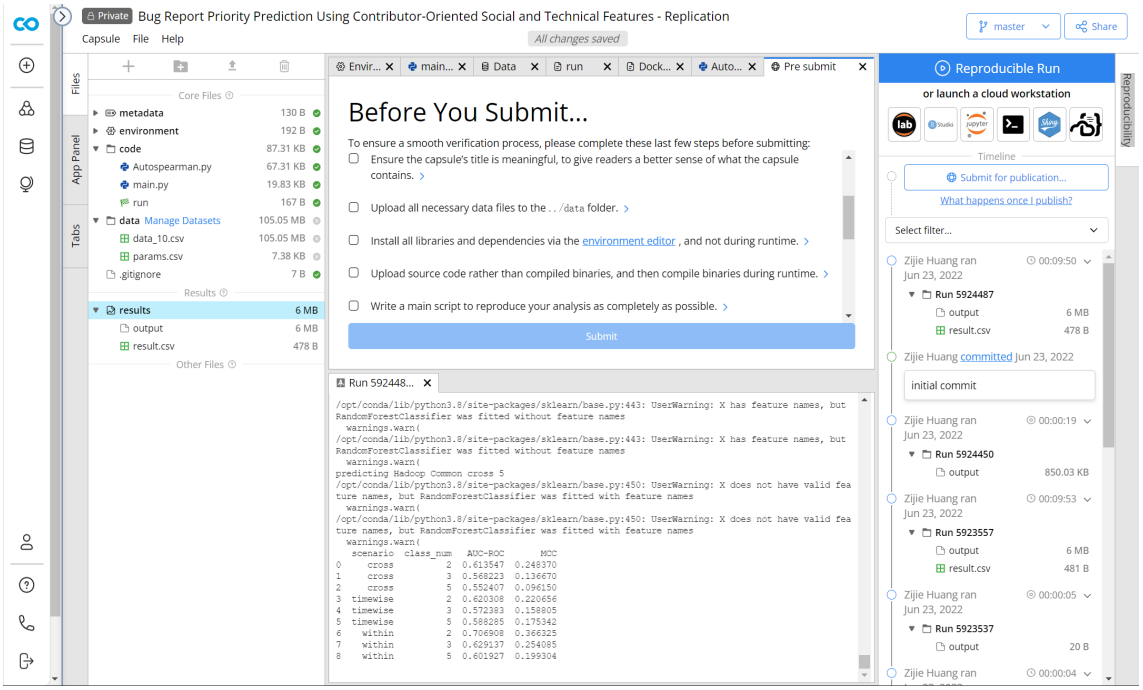
1 | CODEOCEAN REPLICATION



(a) (b)

FIGURE 1 A screenshot of CodeOcean replication run executed by the first author (Zijie Huang) in Jun 23, 2022. The running time is 00:09:50. The performance is printed in the console, and also exported to /results/result.csv.

CodeOcean is a platform for scientific replication officially recommended by established software engineering journal such as Elsevier Science of Computer Programming. It allow users to create VMs for online live replication.

Figure 1 demonstrates the CodeOcean screenshot of replication run. As we stated in the Data Availability Statement section in the paper, the URL to access is https://doi.org/10.24433/CO.7284666.v2. The first URL contains executable code, used data, parameters, and experimental results of model performance. The results are verified by third-parties (CodeOcean) for reproducibility and consistency. The code could also be re-executed by pressing the "Reproducible Run" button on the upper right corner, the execution time is about 10 minutes, and the result will be consistent. The result is located in /results/result.csv. The data could be found in /data/data_10.csv. The tuned parameters of training classifiers is in /data/params.csv. The code is the /code/main.py.

Note that the capsules cannot be published if the results are not consistent. The results are MANUALLY verified by CodeOcean third-parties. Since the first author set the random_states to fixed values, we believe the result will be the same after each execution.

To rerun the code, free registration may be required. Mainland China users may require network environment in other regions to enter the captcha for registration. If not applicable, the files in the capsule are all downloadable, and it could also be ran offline.

2 | PARAMETER TUNING FOR OTHER CLASSIFIERS

See Table 1 for all tuned parameters. Please check **best_clf.csv** in the GitHub repository for the best settings of the classifiers for every project (in each scenarios and different numbers of predicted classes).

TABLE 1 Tuned Parameters

	parameter name	values
NB	var_smoothing	[0,1e-9,1e-7,1e-5,1e-3,0.1,1]
DT	min_samples_split	[2,6,10,14,18]
SVM	C	[0.1, 0.01, 0.001, 0.0001, 0.00001]
LR	tol (tolerence)	[0.1, 0.01, 0.001, 0.0001, 0.00001]
MLP	alpha	[0.1, 0.01, 0.001, 0.0001, 0.00001]
ADA	n_estimators	[10, 60, 100, 150, 200, 250, 300, 350, 400]

3 | DATA GENERATION

Since we are designing a prototype for practice, and the data source are from multiple tools and sources (including tools like Codeface4smells that require VMs to execute, and data stocked in PostgreSQL databases), we cannot provide a unified script for one-click data generation. However, we can still provide instructions for generating such data.

3.1 | Social (Sentiment) Features in Table 2, Surface Features in Table 4, and the JIRA dataset

Please download The sentiment dataset available in: http://ansymore.uantwerpen.be/system/files/uploads/artefacts/alessandro/MSR16/archive3.zip

The archive is a PostgreSQL dumped sql file. PostgreSQL should be installed to import restore the SQL.

3.2 | Socio-Technical Features in Table 3

The .conf files are configurations to perform Codeface4Smells detections. They are provided in **conf.7z** in our GitHub repository. The manual and source code of the tool is available in: https://github.com/maelstromdat/CodeFace4Smells

3.3 | Experience Features in Table 5

These features could be generated based on the data in Section 3.1. Practically we use Python-based ORMs to generate them.

3.4 | Extended Features in Table 6

Code features generation process is two-fold: (1) Retrieving the updated date of bug reports from the dataset in Section 3.1, checking them out to the latest commit before the updated date using PyDriller[1], and executing the CKAniche tool[2] to generate results for every class, (2) Training LSI models using corpus of all codes and the concerned bug report, and calculate similarity to retrieve the most similar class file. Finally, we use the result in step (1) of the most similar class file as features. Gensim[3] is a Python-based topic modeling and word vector generation library.

LDA topic features and the u_mass coherence calculation are also based on Gensim.

3.5 | The Data Generation Script

After configuring the database environment in Section 3.1, the features in Section 3.1 to 3.3 could be generated using the extract_code.py script in extract_data.zip (we provide raw results of Section 3.2 in the zip). Afterwards, executing append_code.py could generate the extended features.

3.5.1 | Adversarial Training Data Generation

This first revision of this paper introduced the adversarial training process. The script used for data generation is “adv.py”, and the used input.pk file is available in

https://drive.google.com/file/d/1LDAd4zuOuEHTjndgeInHEyRlkm8qEutb/view?usp=sharing

References

1. Spadini D, Aniche MF, Bacchelli A. PyDriller: Python framework for mining software repositories. In Proc. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE). 2018: 908–911

2. Aniche M. Code metrics calculator (CK). https://github.com/mauricioaniche/ck/; 2022.

3. Gensim . *Gensim*. 2022. https://radimrehurek.com/gensim/.