Empirical Software Engineering (2024) 29:60
https://doi.org/10.1007/s10664-024-10467-3

Comparative analysis of real issues in open-source machine learning projects

Tuan Dung Lai[1] · Anj Simmons[1] · Scott Barnett[1] · Jean-Guy Schneider[2] · Rajesh Vasa[1]

Accepted: 14 February 2024

Abstract
Context In the last decade of data-driven decision-making, Machine Learning (ML) systems reign supreme. Because of the different characteristics between ML and traditional Software Engineering systems, we do not know to what extent the issue-reporting needs are different, and to what extent these differences impact the issue resolution process.
Objective We aim to compare the differences between ML and non-ML issues in open-source applied AI projects in terms of resolution time and size of fix. This research aims to enhance the predictability of maintenance tasks by providing valuable insights for issue reporting and task scheduling activities.
Method We collect issue reports from Github repositories of open-source ML projects using an automatic approach, filter them using ML keywords and libraries, manually categorize them using an adapted deep learning bug taxonomy, and compare resolution time and fix size for ML and non-ML issues in a controlled sample.
Result 147 ML issues and 147 non-ML issues are collected for analysis. We found that ML issues take more time to resolve than non-ML issues, the median difference is 14 days. There is no significant difference in terms of size of fix between ML and non-ML issues. No significant differences are found between different ML issue categories in terms of resolution time and size of fix.
Conclusion Our study provided evidence that the life cycle for ML issues is stretched, and thus further work is required to identify the reason. The results also highlighted the need for future work to design custom tooling to support faster resolution of ML issues.

Keywords Machine learning · Applied AI · Empirical analysis · Bug · Issue

Communicated by: Maria Teresa Baldassarre and Neil Ernst

This article belongs to the Topical Collection: *Special Issue on Registered Reports*.

✉ Tuan Dung Lai
tuan.lai@deakin.edu.au

Extended author information available on the last page of the article

1 Introduction

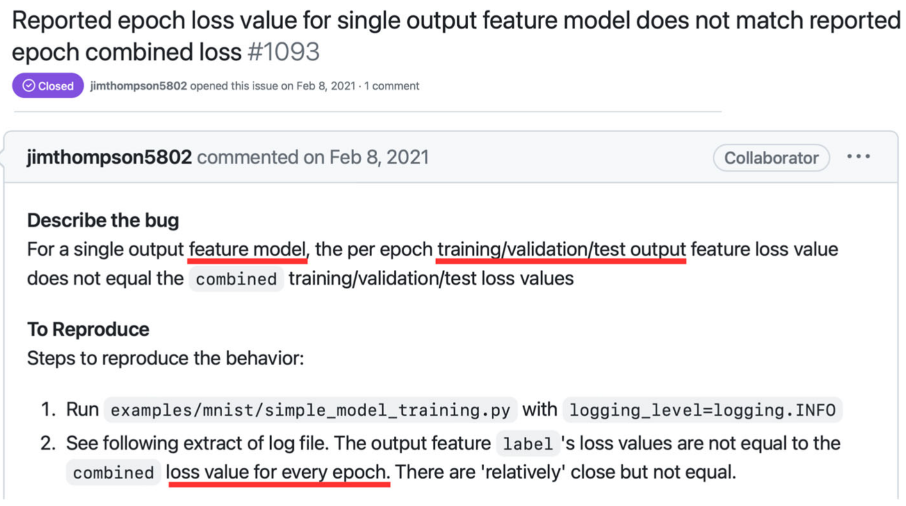
Broad adoption of machine learning (Janssen et al. 2022; Rawindaran et al. 2021; Baskaran et al. 2021) instigates an exploration into whether this impacts the nature and attributes of software issues. In contrast to traditional software systems where instructions are explicitly implemented in the code, machine learning involves the development of statistical algorithms capable of generalising data to perform tasks without explicit instructions. As a result, software teams encounter issues specific to a) bugs from changes in the input data streams, b) training and evaluation pipelines, and c) models and choice of hyper-parameters. Understanding the effort required to complete fixes for the issues that arise in machine learning applications has implications on project estimates and schedules.

Software defects are well studied and best practices for reporting traditional software system issues are well known (Chou et al. 2001; Li et al. 2006; Lal and Sureka 2012; Bhattacharya et al. 2013; Davies and Roper 2014; Tan et al. 2014; Ghanavati et al. 2020). However, it is unknown if this research applies to issues found in machine learning applications. Works that focus on the ML aspect fit into two groups 1) issues that occur in machine learning frameworks (Morovati et al. 2023) and 2) issues that arise in ML applications. However, these works do not distinguish between issues related to machine learning and issues in the surrounding software. We aim to close these gaps. Our goal is to collect empirical evidence to help inform resource allocation, reporting tools, and maintenance infrastructure specific to ML issues. In this paper, we use *ML issues* to refer to tickets raised in open-source ML projects that indicate an ML-specific problem in the ML code of the software (as opposed to non-ML issues related to the system surrounding the ML code, such as configuration, data collection, and data verification components). Figure 1 shows an example ML-issue from an open-source project[1] (a data-centric declarative deep learning framework) that arises during training time and does not immediately cause a failure. The highlighted red indicates issues around components related to the model and the training process which are unique to ML.

Current research on ML issues focuses on bug classification (Sun et al. 2017; Humbatova et al. 2020; Thung et al. 2012), root cause analysis (Chen et al. 2022; Shen et al. 2021; Zhang et al. 2020, 2019; Islam et al. 2019; Zhang et al. 2018), testing techniques for ML (Braiek and Khomh 2020), defect detection (Nikanjam et al. 2021; Wardat et al. 2022; Liu et al. 2021; Yan et al. 2021) and fault localisation (Wardat et al. 2021). These studies provide an understanding of the characteristics of ML issues but do not compare to the existing body of knowledge on software issues. Without drawing a direct comparison to prior knowledge, we cannot assess if existing best practices apply to ML issues. Specifically, we do not know if ML issues follow the same distribution for i) resolution time or ii) fix size. By investigating the differences between ML and non-ML issues we discover if bespoke ML-specific adaptation of software engineering best practices is required. To the best of our knowledge, we are the first to study the differences between ML and non-ML issues.

There are no known techniques for guaranteeing robustness against all possible failure modes; ML will eventually fail due to data shift (Wang et al. 2018; Parker and Khan 2015). ML applications also have a higher complexity level compared with the traditional ones (Amershi et al. 2019) and possess multiple unique challenges in engineering (Galin 2004) due to added components in the architecture that increase technical debt such as the data pipeline, model training, and monitoring systems (Sculley et al. 2014). Development processes also change with the inclusion of machine learning (Seymoens et al. 2018). Model training is one example of an extra component in ML applications compared to traditional software systems. Unlike

[1] https://github.com/ludwig-ai/ludwig/issues/1093



Reported epoch loss value for single output feature model does not match reported epoch combined loss #1093

Closed jimthompson5802 opened this issue on Feb 8, 2021 · 1 comment

jimthompson5802 commented on Feb 8, 2021 Collaborator

Describe the bug
For a single output feature model, the per epoch training/validation/test output feature loss value does not equal the `combined` training/validation/test loss values

To Reproduce
Steps to reproduce the behavior:

1. Run `examples/mnist/simple_model_training.py` with `logging_level=logging.INFO`
2. See following extract of log file. The output feature `label`'s loss values are not equal to the `combined` loss value for every epoch. There are 'relatively' close but not equal.

Fig. 1 A closed ML issue related to the training process. (ML-specific aspects underlined)

traditional software systems, ML functionality is learned from data which requires ML-aware tools (i.e. debuggers, linters, and development environments) and operating procedures. ML-specific testing strategies help, but are insufficient for guaranteeing bug-free software – failure cases will be missed during testing. Thus, rapid response to machine learning failures is required to ensure reliable software.

The research aims to answer the following research questions:

– **RQ1.** *What is the frequency of ML issue categories in open-source applied ML projects?*
– **RQ2.** *How does the distribution of ML and non-ML issues compare in terms of resolution time and size of fix?*
– **RQ3.** *How does the distribution of different ML issue categories compare in terms of resolution time and size of fix?*

In our research, we have 6 categories of ML issues under consideration: GPU Usage, Mode, Tensor and Input, Training Process, Third-party Usage, and Other. The definitions of each category are described in Table 3. To answer the research questions, we studied ML issues and non-ML issues from 4,524 open-source applied AI projects released from Gonzalez et al. (2020). Our research aims to benefit software engineers, and industry practitioners who work on software systems with ML components. Most software engineers are not building ML frameworks, they build applications that utilise them. To be able to generalise what SE works on in the industry, we focus on empirical analysis of ML applications where ML frameworks and libraries are used instead of ML frameworks. Our study compared 147 ML with 147 non-ML issues. Our key findings include 1) ML issues take a median difference of 14 days longer to fix compared to non-ML issues, 2) the most frequently reported types of ML issues are due to the Model, Tensor and Input, or Training Process, rather than GPU Usage or Third-party Usage, in alignment with Humbatova et al. (2020) (albeit we obtain a different distribution), and 3) no statistical difference in size of fix between ML and non-ML issues. These findings provide insight into the prevalence and characteristics of ML issues, emphasizing the importance of addressing them effectively in the field of machine learning. The methodology of this study has been published in a registered report (Lai et al. 2022). A minor change from the original study was a revised taxonomy from Humbatova et al. (2020)

as an agreement between the authors could not be achieved when classifying issues. See the technical report in the supplementary materials for more details. We release data and code for our analysis online[2].

The contribution of the research is summarised below:

– Empirical evidence showing the differences between ML and non-ML issues in terms of resolution time and size of fix.
– Replicable methodology to automatically filter machine learning issues in open-source projects.
– Validation of a known taxonomy of machine learning issues, including testing applicability of the taxonomy on known issues using an iterative manual process.
– A dataset of closed ML and non-ML issues, consisting of the project name, issue title, associated pull requests (PR) to fix, resolution time, and size of the fix. For ML issues, the dataset also includes the particular ML issue category, which has been validated and labelled using an iterative method with a moderate level of consistency.

The structure of the paper is as follows. We first explain the background of the research in Section 2. The methodology followed to answer each research question is then explained in Section 4. The results of each research question are presented in Section 5 which are discussed in Section 6. Related works are mentioned in Section 3 and threats to the validity in Section 7. Finally, we conclude the paper in Section 8.

2 Background

In this section, we describe how recent literature in empirical Software Engineering for AI defines and classifies ML issues. After that, we will define the ML issue lifecycle to highlight the differences between ML issues and non-ML issues.

2.1 Definition of ML Issues

Machine learning issues in open-source projects have been empirically studied, different terminologies are used in those studies including bugs in ML projects (Sun et al. 2017), deep learning failures, deep learning faults (Humbatova et al. 2020), deep learning bugs (Islam et al. 2019; Zhang et al. 2018), ML issues (Thung et al. 2012), and issues in AI projects (Arya et al. 2019). However, the authors of these works only consider open-source ML frameworks and treat all the issues from the projects as ML issues. Table 1 shows the open-source projects used in recent studies to understand ML issues. In our study, we define an ML issue as a problem, bug, enhancement, or task related to the development, implementation, or improvement of machine learning functionality in the context of developing, training, or deploying machine learning models within an open-source project.

2.2 Classification of ML Issues

We classified ML issues as a subset of issues found in ML applications that contain ML keywords found in the issue title and involve the utilisation of ML libraries to address them. Humbatova et al. (2020) introduced a taxonomy of faults in Deep Learning systems containing

[2] https://github.com/DungLai/EMSE_dataset

Table 1 Number of projects included in recent empirical studies on ML issues

Study	Total	Projects
Morovati et al. (2023)	2	TensorFlow, Keras
Chen et al. (2022)	4	TensorFlow, PyTorch, MXNet, DL4J
Humbatova et al. (2020)	3	TensorFlow, Keras, PyTorch
Tambon et al. (2021a)	2	Keras, TensorFlow
Shen et al. (2021)	3	TVM (Apache), Glow (Facebook), nGraph (Intel)
Islam et al. (2019)	5	Caffe, Keras, Tensorflow, Theano, Torch
Sun et al. (2017)	3	Scikit-learn, Paddle, Caffe
Thung et al. (2012)	3	Apache Mahout, Lucene, and OpenNLP
Our study	27	Shown in Table 4

5 top-level categories by conducting interviews with practitioners and analysing GitHub artefacts and Stackoverflow posts related to TensorFlow, Keras, and PyTorch. We use a revised version of Humbatova et al.'s taxonomy to label our ML issue dataset.

2.3 Challenges of ML in Issue Lifecycle

The life cycle of a software issue is well understood (Gegick et al. 2010). An example life cycle for the Bugzilla issue tracker is shown in Fig. 2. Our overall hypothesis is that ML issues will take longer to fix and have a larger size of fix than non-ML issues because ML systems have a higher complexity level compared with the traditional ones (Amershi et al. 2019) and possess multiple unique challenges in engineering (Galin 2004) due to added components in the architecture that increase technical debt such as the data pipeline, model training, and monitoring systems (Sculley et al. 2014). Below we motivate how delays for ML issues can

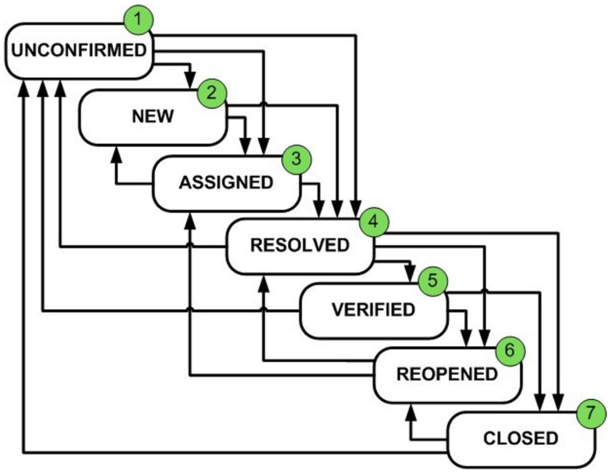


Fig. 2 Bugzilla issue life cycle (Gegick et al. 2010)

occur at each stage of the issue lifecycle. Issues that take an unexpected amount of time to close impact project schedules and increase maintenance costs.

Unconfirmed, New Issues raised for a software system occur when a user experiences a system failure. Users fill out an issue report that documents the failure in a ticketing system to be assigned to a developer. Project teams need a thorough understanding of the application and the category of the issue to be able to estimate the time to create a fix. However, ML issues have specific defect categories (Humbatova et al. 2020). In some cases, the only symptom of these defects is that the prediction is incorrect (Tambon et al. 2021a). This requires exploration time to understand the root cause. Whether the categories of ML issues impact resolution time is unknown.

Assigned Knowing who to assign an ML issue to is challenging as ML issues occur in a) the architecture of the model, b) training pipelines, c) choice of hyperparameters, d) serving pipelines, and d) assumptions about inputs. Unlike traditional bugs, solving ML issues requires different skills from data scientists, software engineers, and domain experts to collaborate together to address data, code and model-related bugs (Seymoens et al. 2018).

Resolved Once a developer has been assigned an issue they attempt to locate, and resolve the issue. To resolve an ML issue like the one in Fig. 1, understanding is required of the different model architectures and the training process that causes the error. Resolving ML issues requires extra time to prepare, train, and validate an ML model.

Verified Verifying of ML is an on-going research area (Xiang et al. 2018). The difficulties of verifying if an ML issue has been resolved lie in understanding the unbounded function learned by the machine – the limitations of the machine cannot be inspected by a human. ML have inference and training, inferring the root cause of failures and unexpected behaviour is challenging, usually requiring much human thought, and is both time-consuming and error-prone Lourenço et al. (2019).

Closed, Re-Opened ML issues resolved by retraining involve no code or configuration changes (i.e., only model versions and training data are updated). This requires additional infrastructure outside of the source code repository indicating how the training strategy (i.e., hyperparameters, and training data) addressed the issue for replication.

In conclusion, due to the new issues that arise in each stage of the life cycle because of the unique characteristics of ML, we hypothesize that the resolution time is higher and the issue life cycle is stretched. Our research will further confirm this hypothesis. After that evidence is established to look further into what steps of the life cycle actually differ for ML and cause the resolution time to increase. We will investigate different stages of the life cycle and the corresponding ML problem raised.

2.4 Comparison of ML and Non-ML Systems

Machine Learning (ML) systems, as a superset of traditional non-ML systems, inherit all the challenges of the latter while introducing unique complexities related to their data-driven nature Sculley et al. (2015). Unlike non-ML systems, which primarily rely on logic, rules, and direct computation, ML systems are heavily dependent on the quality and quantity of data for training algorithms to learn patterns or features Zhou et al. (2017). This leads to challenges such as overfitting, where the model excessively learns from training data including its noise, and underfitting, where the model fails to capture the underlying data patterns. Bias in ML systems can arise from skewed training data, where certain groups are under-represented in the dataset, leading to prejudiced outcomes Mehrabi et al. (2021). Other critical issues include interpretability, the difficulty in understanding complex model decisions, fairness in decision-making, the generalisation capability of models to perform well on unseen data, and drift, which refers to changes in data patterns over time affecting model performance Lu et al. (2018).

In contrast, non-ML systems focus on direct problem-solving, manual processes, and designing infrastructure, with challenges centred around scalability, maintainability, and efficiency. Their evaluation is based on correctness and resource usage, and they lack the dynamic adaptation and self-evolution capabilities of ML systems. The complexity in non-ML systems is not related to data but to the nature of the problem and application scale. The knowledge domain for ML includes data science, statistics, and machine learning algorithms, whereas non-ML requires expertise in software development and engineering. Therefore, ML systems, with their additional layers of data dependency, training, and algorithmic challenges, represent a more complex and evolving subset of traditional computing systems. Table 2 summarises the different characteristics of ML and non-ML systems.

3 Related Work

Research on understanding ML issues has been conducted on open-source ML frameworks and has focused on categorising bugs and building taxonomies (Sun et al. 2017; Humbatova et al. 2020; Thung et al. 2012), finding root causes (Zhang et al. 2019, 2020; Islam et al. 2019; Zhang et al. 2018), ranking bugs by frequency and severity (Zhang et al. 2020; Thung et al. 2012; Zhang et al. 2019; Islam et al. 2019), extracting common fix patterns (Sun et al. 2017; Zhang et al. 2020), and measuring resolution time (Sun et al. 2017; Thung et al. 2012). Some classes of ML defects are more severe than others and require an extended time to fix (Sun et al. 2017). In 2017, Sun et al. (2017) conducted an empirical study on real bugs in ML frameworks from 329 ML issues from 3 large open-source ML projects from Github and found that nearly 70% of bugs are fixed within one month, certain types of defects are more severe than others and require an extended time to fix (Sun et al. 2017). However, they did not compare the empirical statistics against a baseline for non-ML issues. In 2019, Arya et al. used supervised learning to extract common topics from 15 complex issues reports and their comments from 3 open-source ML projects Arya et al. (2019). The aim of their study was not to understand ML-specific issues but to generalise these topics to software projects. In 2020, Humbatova et al. analysed Github issues/ pull requests/ commits and Stack Overflow discussion to propose a taxonomy of deep learning faults and conducted interviews with practitioners to rank categories of bugs based on severity and effort required (Humbatova et al. 2020). However, the severity and effort estimates for each category are based on interviews rather than an empirical analysis of issues.

Software engineering research in understanding issues in open-source software projects focuses on effort estimation techniques to predict resolution time which can benefit bug fix scheduling and team allocation tasks (Du et al. 2022; Al-Zubaidi et al. 2017; Ardimento and Boffoli 2022). Akbarinasaji et al. conducted a study on open-source software to verify an existing model for predicting resolution time and found that the existing model that estimated

Table 2 Comparison of ML and non-ML systems characteristics

Aspect	ML systems	Non-ML systems
Data Dependency	Highly dependent on data quality and quantity	Relies less on data, more on logic, rules, and direct computation
Training Process	Requires training algorithms on a dataset to learn patterns or features	Involves direct problem-solving techniques, a manual process, and designing infrastructure
Testing	Depends on performance metrics such as accuracy, precision, recall, F1 score	Evaluation is based on correctness, efficiency, and resource usage
Evolution	Self-evolves over time, dynamic adaptation due to drift	Solutions are static, require manual updates from developers
Challenges	Overfitting, underfitting, bias, interpretability, drift, fairness, generalization, data privacy	Scalability, maintainability, efficiency, compatibility
Domain Specific	Requires knowledge in data science, statistics, machine learning algorithms	Requires knowledge in software development, engineering, and deployment
Complexity	Complexity arise from the volume of data, feature selection, tuning of parameters, and algorithms. Hidden technical debt from model, training, and monitoring components	Complexity is generally not related to data but to the nature of the problem itself (algorithmic complexity). Complexity evolves with the scale of the applications and new features

the bug fixing time is robust enough to be generalized Akbarinasaji et al. (2018). However, with the unique characteristics of ML applications, we do not know if this is still applicable to ML issues. To understand the relationship between the size of fix (which was referred to as code-churn value) and resolution time, Vieira et al. analysed 55 projects from the Apache ecosystems to conclude that if an issue has a higher size of fix and linked to other existing issue reports, resolution time will be at least twice as long (Vieira et al. 2022).

4 Methodology

In this section, we describe the methodology followed to understand the differences between ML and non-ML issues. First, we explain the changes between the methodology and the registered report for this research, and then we describe the data collection methodology for the ML issue and non-ML issue datasets. Finally, we explain how we conducted the statistical analysis to answer our three research questions.

4.1 Deviation from the Original Proposed Study Registered Report

We made a change to the registered report: the process of labelling ML issues. For labelling ML issues: In our registered report for this research (Lai et al. 2022), we planned to label the ML issues against the 18 sub-categories present in Humbatova's taxonomy (Humbatova et al. 2020) and to conduct the comparison among those categories. However, our attempt to apply the taxonomy to issues in applied AI projects indicates that the taxonomy is inadequate for labelling our dataset, and resulted in no agreement measured by Light's Kappa metric (Light 1971). As Humbatova's taxonomy was derived from artifacts and discussions related to deep learning frameworks for Python, we adapted the taxonomy to be able to use it on real bugs in applied AI projects. For this reason, we narrowed down the number of ML categories that will be used to 6 including an "other" category, the other 5 categories are derived from the 5 parent nodes in the original taxonomy, i.e. GPU Usage, Model, Tensor and Input, Training, and API. We also proposed a protocol for the labelling process and a clearer definition for each category to avoid ambiguity. A detailed process for labelling ML issues, the details of what led to the decision to modify the taxonomy, the results of the labelling iterations, and the extended definitions and adjustments for the 6 ML issue categories were documented in our technical report in the supplementary materials. Table 3 shows the definitions of 6 ML issue categories that are used in our three research questions after the adaptation process.

Overall, after 3 labelling iterations, we made the following 6 changes to the original taxonomy. The decision-making process and the challenges we faced during the labelling process are documented in the technical report in the supplementary material:

- We collapsed and condensed the definitions provided by Humbatova et al. (2020) into the definitions provided in Table 3. In the study by Humbatova et al. (2020), the definitions are provided for each low-level sub-categories, the authors also explain how the interview process leads to the creation of the taxonomy in the text. This makes it difficult for people who have not read the whole Humbatova et al. (2020) study to understand how to apply the taxonomy created independent standalone set of definitions to establish our labelling protocol.
- We changed "API" to "Third-party usage" because some issues belong to other libraries, we generalised it to cover more cases.

Table 3 Extended definitions of 6 ML issue categories

Categories	Definitions
GPU Usage	Incorrect or inefficient usage of GPUs, wrong reference to GPU device, failed parallelism, incorrect state sharing between subprocesses, faulty transfer of data to a GPU device.
Model	Inappropriate, inefficient or incorrect model initialisation, choice of architecture. Inappropriate use of activation function, incorrect properties for a neural network layer, missing, redundant or wrong layer. Errors occur during inference.
Tensor and Input	Error or inefficiencies in data quality such as low-quality data, noisy data, imbalanced data, and insufficient data. Inappropriate preprocessing of data such as scaling, normalisation, and feature engineering. Incorrect shapes of input, wrong dimensions, size, inappropriate file type, encoding, and selection of data format.
Training Process	Inappropriate or inefficient training processes excluding data-related problems, such as inappropriate batch sizes, and learning rates. Hyperparameter issues such as learning rate, dropout rate, and number of epochs. Inappropriate optimiser, inappropriate choice of loss function when using during training. Inefficient or incorrect validation/ testing procedure.
Third-party Usage	Inappropriate usage of third-party programs or libraries, such as TensorFlow, PyTorch, Keras, and Numpy.
Other	Documentation issues or anything unrelated to the 5 categories above.

– "Training data quality" and "Preprocessing of training data" which were originally in "Training" are now moved into "Tensor and Input" categories. This change is to make "Tensor and Input" specifically only contain issues related to data.
– We changed "Training" to "Training process" to ensure all issues in this category are related to the process instead of the data.
– We added an additional "Other" category to fit all issues that do not belong to any category in the original taxonomy.
– We change the definitions of each category so that Enhancement, code refactoring, and non-critical issues (issues that do not crash the program) can be classified and labelled as ML issues.

These changes will not affect RQ2 and RQ3. RQ 2 is unaffected as this question requires a comparison between ML and non-ML issues. The coarse grain application of the taxonomy will not change this result. Using the fine-grain version of the taxonomy (as per the original design) would have made RQ1 and RQ3 unanswerable due to the size of the resulting dataset (147 spread over 6 coarse grain categories).

4.2 Data Collection

In 2020, Microsoft released a paper (Gonzalez et al. 2020) containing all AI and ML relevant projects from Github in the past 10 years. The dataset from this paper is used as a starting point containing 4,524 applied AI & ML projects. After that, we applied filters described in Fig. 3. Overall, we used Github API to retrieve repository metadata to filter the projects. We then downloaded pull requests and issues to filter the issues by ML keywords in the issue's title and ML libraries in the pull request. After that, we labelled the ML issues and sampled the non-ML issues. The data collection process resulted in a dataset containing 147 ML issues from 27 applied AI projects, manually labelled into 6 categories (described in our technical report in the supplementary materials) and 147 non-ML issues. This section provides details of each filtering step shown in Fig. 3 (Table 4).

4.2.1 Filter ML Projects

This step filters 4,524 projects to 428 projects by language, newness, popularity, and activeness. We first chose to focus on Python because 1) it is the most popular language for machine learning and 2) the taxonomy was originally applied to Python projects.

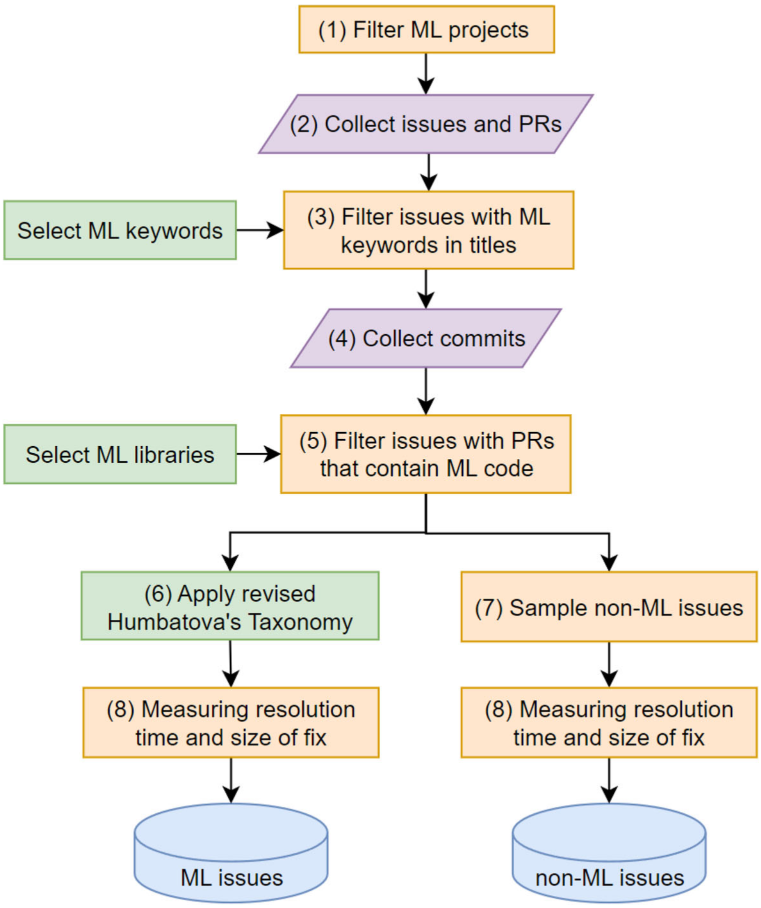

(1) Filter ML projects
(2) Collect issues and PRs
Select ML keywords
(3) Filter issues with ML keywords in titles
(4) Collect commits
Select ML libraries
(5) Filter issues with PRs that contain ML code
(6) Apply revised Humbatova's Taxonomy
(7) Sample non-ML issues
(8) Measuring resolution time and size of fix
(8) Measuring resolution time and size of fix
ML issues
non-ML issues

Fig. 3 High-level methodology for data collection (green: manual process, yellow: automatic approach, blue: dataset, purple: an automatic approach using Github API)

Table 4 List of applied AI projects, number of ML issues and non-ML issues after the data collection

Projects (owner/ name)	ML issues	Non-ML issues
sktime/sktime	32	32
ludwig-ai/ludwig	20	20
NRCan/geo-deep-learning	15	15
kymatio/kymatio	14	14
blue-oil/blueoil	10	10
medipixel/rl_algorithms	6	6
intel/dffml	5	5
brendanhasz/probflow	5	5
aiqm/torchani	5	5
pyro-ppl/funsor	4	4
onnx/onnxmltools	4	4
qubvel/efficientnet	3	3
HunterMcGushion/hyperparameter_hunter	3	3
BindsNET/bindsnet	3	3
philipperemy/keras-tcn	2	2
netrack/keras-metrics	2	2
hackingmaterials/automatminer	2	2
atomistic-machine-learning/schnetpack	2	2
DeepLabCut/DeepLabCut	2	2
stared/livelossplot	1	1
mozilla/bugbug	1	1
marl/openl3	1	1
justinshenk/fer	1	1
catalyst-team/catalyst	1	1
autonomio/talos	1	1
ProGamerGov/neural-style-pt	1	1
HazyResearch/fonduer	1	1
Total: 27 Projects	147 ML issues	147 Non-ML issues

For each project, we collected the metadata and other API URLs for the repository in the form of a JSON file. By following this method, we locally downloaded JSON files for 4,524 repositories using the project names and owner names from the Microsoft dataset (Gonzalez et al. 2020). These projects are original (not forked from another project). 2,342 repositories are found to use Python as their programming language. However, we found that 37 projects could not be retrieved using the API due to the projects being deleted, set to private, or renamed since the dataset's release.

We further select ML repositories using the following criteria (commonly used in prior works in empirical open-source software issues research (Biswas et al. 2019; Morovati et al. 2023)): newness (the projects must be created in the last 5 years), popularity (the projects must have at least 100 stars), activeness (the projects must have at least a commit in the last 2 years), we set 1 Jan 2023 as the current date of research. This resulted in a list of 435 projects, we then filtered out projects with zero issues, resulting in 428 projects.

4.2.2 Collect Issues and Pull Request

For each issue in the 428 projects from the previous step, we downloaded issue reports with associated merged pull requests. We noticed that many of the issue numbers were redirected to a pull request. To obtain a list of only issue numbers, we excluded the pull request numbers from the list of issue numbers. This left us with 40,933 issue reports across all 428 projects.

To filter out issues closed by merged pull requests, we downloaded the timeline of each issue, and then selected issues that had at least one pull request mentioning them. When a pull request mentions an issue by using the hashtag followed by the issue number, this event is stored in the issue timeline. A pull request from a different project can mention an issue in another project, we checked that the pull requests originate from the same project as the issue. This process identified 3,969 out of the 40,933 issues. Finally, we further filtered the list of issues to include only those that were closed and only if the pull requests that mentioned the issue were merged into the main repository's codebase. This reduced the number of issues from 3,969 to 3,133.

4.2.3 Filter Issues with ML Keywords in Titles

Overall, we checked the titles of 3,133 issues in the previous steps and 304 issues were found to have at least 1 of the 280 ML keywords. Filtering by keywords is a technique commonly used in empirical software research to filter the topic of a project or issue (Kim et al. 2021; Biswas et al. 2019). We first started with **ML keywords** from the ML Glossary from Google[3]. The ML keywords glossary includes overloaded words such as "class", and "test" which could refer to either ML or non-ML concepts and thus lead to false positives. To mitigate this issue, 3 researchers (the first 3 authors of the study) with applied AI experience manually checked each keyword and we narrowed down the list to 280 keywords (a full list of keywords is provided in the supplementary material). After that, we converted the issue title text to lowercase, removed special characters, commas, and dots, split the titles by white space, trim leading and trailing white space before checking if at least an ML keyword was in the title. This process resulted in 304 issues.

4.2.4 Collect Commits

This step is an automatic process done to prepare for the next step, which is filtering the issues with pull requests that contain ML code. For each merged pull request associated with the issues that we found in previous steps, we use Github API to retrieve the list of commits associated with the pull request and download the raw content of each code file modified by a commit.

4.2.5 Filter Issues with Pull Requests that Contain ML Code

Overall, we checked that the code in the pull requests used to fix the issue has to use at least one ML library as a second filtering layer, this process filtered 304 issues to 151 ML issues. Filtering ML issues only by keywords in the title has the possibility to be imprecise, this is a challenge faced by prior research (Biswas et al. 2019) because the keywords can be used in many contexts. We first **select ML libraries** currently used in popular Python deep

[3] https://developers.google.com/machine-learning/glossary

learning frameworks: TensorFlow, Keras, PyTorch. These libraries are popular and widely adopted in the machine-learning community and commonly used in research to understand deep learning and ML bugs (Zhang et al. 2018; Jia et al. 2020; Tambon et al. 2021b; Chen et al. 2022; Morovati et al. 2023; Islam et al. 2019).

For each of the 304 issues, we have downloaded all the files modified by the PRs that addressed the issue. We then extracted the dependencies using the 'FindImports' library,[4] this library extracts Python module dependencies by parsing source files. It can report names that are imported but not used. We then included only the issues in which at least one ML library (TensorFlow, Keras, PyTorch) was used in the fix. Finally, we got the list of 151 ML issues that satisfy two criteria: 1) issues containing ML keywords in the title, and 2) the issue is associated with at least one pull request that contains a change in the context of a script that uses ML libraries.

4.2.6 Apply Revised Humbatova's Taxonomy

We manually label ML issues in the ML issue dataset against an existing taxonomy (Humbatova et al. 2020) using an iterative approach. The attached technical report in the supplementary materials describes the detail of the labelling, adjustment and how we apply the taxonomy to achieve a moderate level of agreement using the Light's Kappa metric (Light 1971). Initially, three researchers independently (the first three authors) labelled a random sample of 30 ML issues, resulting in no agreement. The taxonomy was revised in the second iteration, merging subcategories to reduce ambiguity. Another set of 30 ML issues was labelled, resulting in weak agreement. After that, the number of categories was reduced to 6, and a labelling protocol was established, adjustments were made to the taxonomy, including changing ”API” to ”Third-party usage” and redefining categories related to data quality issues. The modifications aimed to improve the taxonomy's applicability and clarity in identifying ML issues. The third iteration showed moderate agreement (Kappa = .67), after that, the first author labelled the rest of the ML issues dataset.

4.2.7 Sample non-ML Issues

To compare ML and non-ML issues, we randomly sampled a **non-ML issues dataset** from each selected project, in equal proportion to the number of ML issues detected in that project (this sampling strategy helps to prevent confounding effects arising from project factors). Non-ML issues are those that don't meet the ML criteria (ML keywords in the titles and ML libraries involved in the fix) but meet all other criteria (i.e., the closed issue with an associated pull request). In 4 projects, we were unable to sample non-ML issues due to the project having only one ML-related issue. We removed these projects from the dataset, leaving us with a total of 294 issues, comprising 147 ML and 147 non-ML issues.

4.2.8 Measuring Resolution Time and Size of Fix

Resolution Time This attribute is measured by the number of days. For each issue, because only closed issues were considered in our filtering process, we were able to use Github API to get the date when the issue was created and closed and calculate the difference between these two dates. If an issue is closed on the same day it is opened, the resolution time will be zero.

[4] https://pypi.org/project/findimports/

Size of Fix This attribute is a cumulative count measured by the sum of the number of lines added and deleted that are performed in a pull request that fixes the issue. This calculation method measures the code churn size which has an intuitive relationship with the developer's effort Shihab et al. (2013) and was commonly used in empirical research on software issues El Asri et al. (2019); Zhu and Godfrey (2021). Each issue can have more than one corresponding pull request that mentions it. The first author manually inspected each issue and selected the one pull request that fixed it. In rare scenarios where multiple pull requests were the fix to the same issue, we selected the pull request that was created last. The pull requests created before were partial fixes and were disregarded.

4.3 Data Analysis

RQ1. What is the Frequency of ML Issue Categories in Open-source Applied ML Projects?

To answer this research question, we report the results of our study and compare them to the results obtained by Humbatova et al. (2020). As we used a revised version of Humbatova's taxonomy, we first matched categories in Humbatova's original taxonomy to our revised taxonomy to calculate the number of issues per revised category that would be obtained according to the results reported in Humbatova et al. (2020) study. The chi-square statistical test is conducted to determine if the observed differences in issue frequencies between the two studies are statistically significant or merely due to chance followed by Cramer's V (Cramér 1999) to measure the effect size. Negative results from this research question will invalidate the need to conduct further research and motivate research questions two and three, which are all based on the results of Humbatova et al. (2020) study.

RQ2. How Does the Distribution of ML and Non-ML Issues Compare in Terms of Resolution Time and Size of Fix?

To answer this research question, we first generate tables to compare the mean and median resolution time and size of fix, followed by boxplots to inspect the distribution. For this analysis, we use the Mann-Whitney U Test (McKnight and Najab 2010) (also known as the Wilcoxon rank-sum test), which is used to compare differences between two independent groups when the dependent variable is either ordinal or continuous, but not normally distributed. If the difference is significant by the test, we will measure the rank-biserial correlation (Cureton 1956) to measure the effect size. Furthermore, we manually sample the data to gain insights into the results. Answering this research question will answer our hypothesis that the lifecycle for ML issues is stretched. The results will have an implication for the task scheduling activities, resource allocations and effort estimation for ML projects.

RQ3. How Does the Distribution of Different ML Issue Categories Compare in Terms of Resolution Time and Size of Fix?

To answer this research question, we first generate descriptive statistics tables to compare the mean and median in terms of resolution time and size of fix for each ML issue category. After that, we conduct Kruskal-Wallis H-test (MacFarland et al. 2016) (a non-parametric version of ANOVA) to determine if the medians of the groups differ. The result of this research question provides deeper insights into what will be found in research question 2. The initial evidence derived from this research question will provide insights into what should be investigated to understand why there is a difference in the size of fix or resolution time for ML issues.

5 Results

We investigated and labelled 147 ML issues, 147 non-ML issues, and compared the distribution of ML issues between 6 categories between Humbatova et al. (2020) study and our study. After that, we visualised the distribution, provided a descriptive statistic summary and compared the difference between ML and non-ML issues in terms of resolution time and size of fix. In the following sections, we present our answers to each of our formulated research questions.

5.1 RQ1: What is the Frequency of ML Issue Categories in Open-source Applied ML Projects?

In summary, we analysed 147 ML issues within 6 manually labelled ML categories and ran a chi-square test to compare the frequencies of ML issues between our study and Humbatova et al. (2020) study. The statistical analysis highlights a statistically significant difference between the frequency of ML issue categories in Humbatova et al. (2020) study compared to our study. In our study, Model and Training Process issues were most frequent, whereas GPU Usage and Third-party Usage are the categories with the least number of issues.

We matched and aligned the categories in Humbatova's original taxonomy to our revised taxonomy. "Training data quality" and "Preprocessing of training data" which were originally in "Training" are now moved into "Tensor and Input" categories. The contingency table (Table 5) compares the number of issues per category between Humbatova et al. (2020) study and our study.

The total number of issues being analysed in Humbatova's study and our study were different, and thus, we normalised the data by dividing the number of issues in each category by the total number of issues being collected and analysed in each study (375 and 111 respectively). This is shown in Fig. 4. In contrast to Humbatova et al.'s study, our study indicates that Tensor and Input (data-related issues) are not the most frequent types of issues. Similarly to Humbatova et al, we found GPU Usage issues are the least frequent type of

Table 5 Contingency table: Number of issues per category in Humbatova's study and our study

	Humbatova et al. (2020)	Our study
GPU Usage	11 (2.9%)	2 (1.8%)
Model	74 (19.7%)	32 (28.8%)
Tensor and Input	185 (49.3%)	28 (25.2%)
Training Process	85 (22.7%)	32 (28.8%)
Third-party Usage	20 (5.3%)	17 (15.3%)
Total	375	111

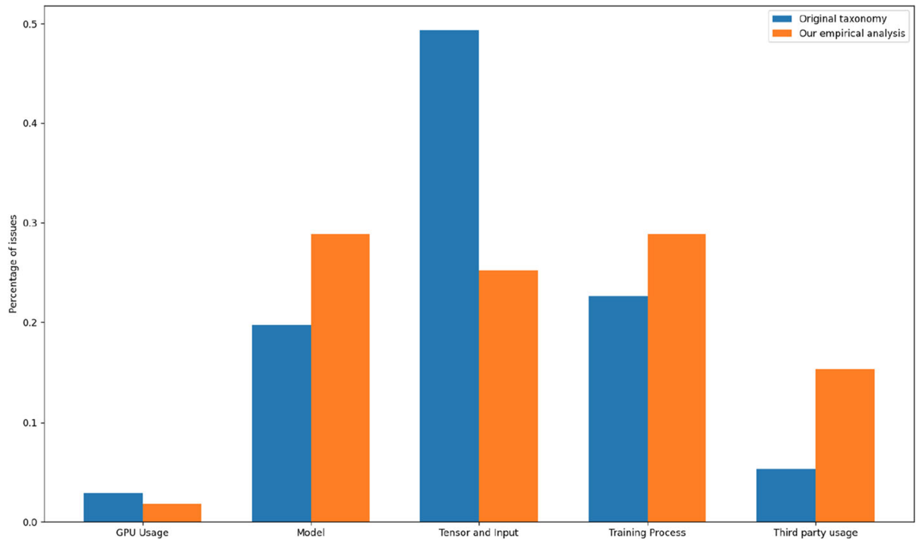


Fig. 4 Normalised distribution of numbers of issues in 5 ML categories - Original taxonomy (Humbatova et al. 2020) vs our empirical analysis

issue, representing 2.9% of issues in Humbatova et al.'s study and only 1.8% of issues in our study.

We conducted a chi-square statistical test to compare the frequency of machine learning issues in our study and Humbatova's study ($p=1.2 \cdot 10^{-25}$, $\alpha=0.05$, $\chi=121.86$, df=4). With such a small p-value, we can reject the null hypothesis and conclude that there is a statistically significant difference in the frequencies of machine learning issues between our study and Humbatova's study, i.e., the observed differences between the two studies are unlikely to have occurred by chance alone. The calculated Cramer's V value (Cramér 1999) is 0.17, which indicates a small effect size. This result indicates that there is a modest association between the categories in the two studies.

5.2 RQ2. How Does the Distribution of ML and Non-ML Issues Compare in Terms of Resolution Time and Size of Fix?

In summary, we compared 147 ML issues against 147 non-ML issues, with the number of issues per project for the two groups being equal. ML issues take a longer time to fix compared to non-ML issues, the difference is statistically significant, and the difference between the medians is 14 days. The mean size of fix for ML issues and non-ML issues are respectively 484 lines and 557 lines, however, the difference is not statistically significant.

5.2.1 Comparing the Resolution Time of ML and Non-ML Issues

The descriptive statistics for resolution time (fix duration) as shown in Table 6 reveal notable differences between ML issues and non-ML issues. On average, ML issues have a resolution time of approximately 101 days, with a wider variation (standard deviation of 177 days). In contrast, non-ML issues have a shorter average resolution time of approximately 61 days,

Table 6 Descriptive Statistics for Fix Duration (Resolution Time)

Descriptive Statistics	ML Issues	Non-ML Issues
Count	147	147
Mean	101 days	61 days
Standard Deviation	177 days	127 days
Minimum	0 days	0 days
25th Percentile	5 days	2 days
Median (50th %ile)	25 days	11 days
75th Percentile	105 days	56 days
Maximum	1008 days	871 days

with a smaller variation (standard deviation of 127 days). The data also shows that ML issues tend to have longer upper quartiles (105 days) compared to non-ML issues (56 days).

The means and median of fix duration in Table 7 show that ML issues take longer time to resolve than non-ML issues based on the data collection used in this research. The box plot in Fig. 6 shows many outliers, 20 outliers for ML and 19 outliers for the non-ML category. The median difference in resolution time between ML and non-ML issues is 14 days, with ML issues taking longer time to fix. Visualisation of the two distributions using a cumulative distribution plot, showing the percentage of resolved issues as a function of resolution time is shown in the survival plot in Fig. 5.

The Mann-Whitney U Test shows that the fix duration for ML issues is significantly longer than for non-ML issues (p=0.01). We use the Mann-Whitney U Test, a non-parametric test that depends only on the rank and is thus robust to outliers. Given that the sizes of the two groups are equal (n1 = n2 = 147) and the U statistic is 12653.5, we measure rank-biserial correlation (Cureton 1956): $r = 1 - (2U)/(n1 * n2) = -0.17$. This suggests a small effect size. The negative sign confirms that the fix duration of ML issues tends to be longer than the fix duration of non-ML issues.

Furthermore, we manually investigated issues that took more than 200 days to close (14 non-ML issues and 23 ML issues), and we found that (i) the issue is non-critical; an enhancement to increase efficiency (ii) the issue requires removing unnecessary dependencies/ refactoring or (iii) The issue requires multiple pull requests to fix (Fig. 6).

5.2.2 Comparing the Size of Fix of ML and Non-ML Issues

The non-parametric Mann-Whitney U Test (Mann and Whitney 1947) indicates that the difference between ML and non-ML issues in terms of the size of fix is not significant (p=0.46). We use a logarithmic scale in the box plot in Fig. 7 because of extreme outliers (issues with large size of fix), there are 15 outliers for the ML category and 20 outliers for the non-ML category. For the size of fix, the values for both ML and non-ML categories are similar, except for the maximum. In ML issues, the maximum size of fix is 9126 lines, while in non-ML issues, the maximum size reaches 23767 lines. These outliers indicate instances where significant code changes were made. We use the Mann-Whitney U Test, a non-parametric test that depends only on the rank and is thus robust to outliers (Table 8).

We manually investigated the 12 ML issues with size of fix more than 1000 lines size of fix, our observation is that these cases are because (i) the pull request is a new feature that fixes multiple issues, (ii) restructured code (.rst) for documentation is included in the fix, or (iii) new file/ model/ optimiser/ components are added with inline documentation.

Table 7 Mean and Median of 6 ML categories for Resolution Time and Size of Fix

ML category	Count	Mean Fix Duration	Median Fix Duration	Mean Line Change	Median Line Change
GPU Usage	2	2	2	29	29
Model	32	121	15	546	142
Tensor and Input	28	143	38	350	71
Training Process	32	78	20	854	52
Third-party Usage	16	70	11	94	32
Other	37	90	35	412	159

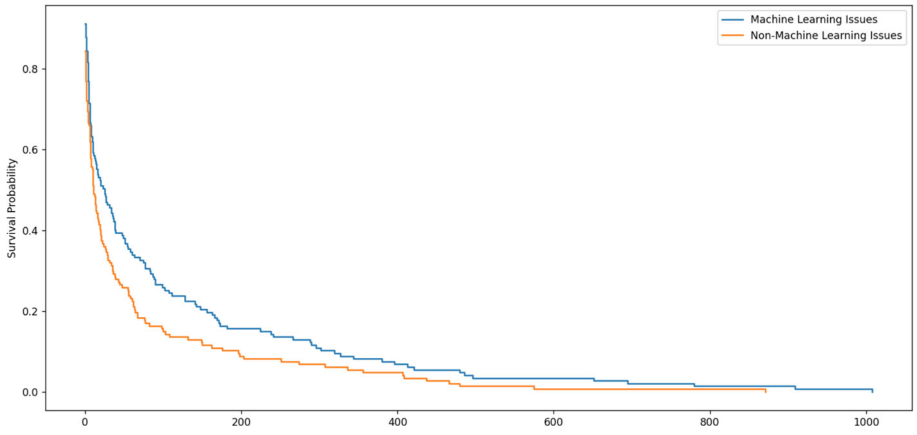


Fig. 5 Survival function of resolution time between ML and non-ML issues

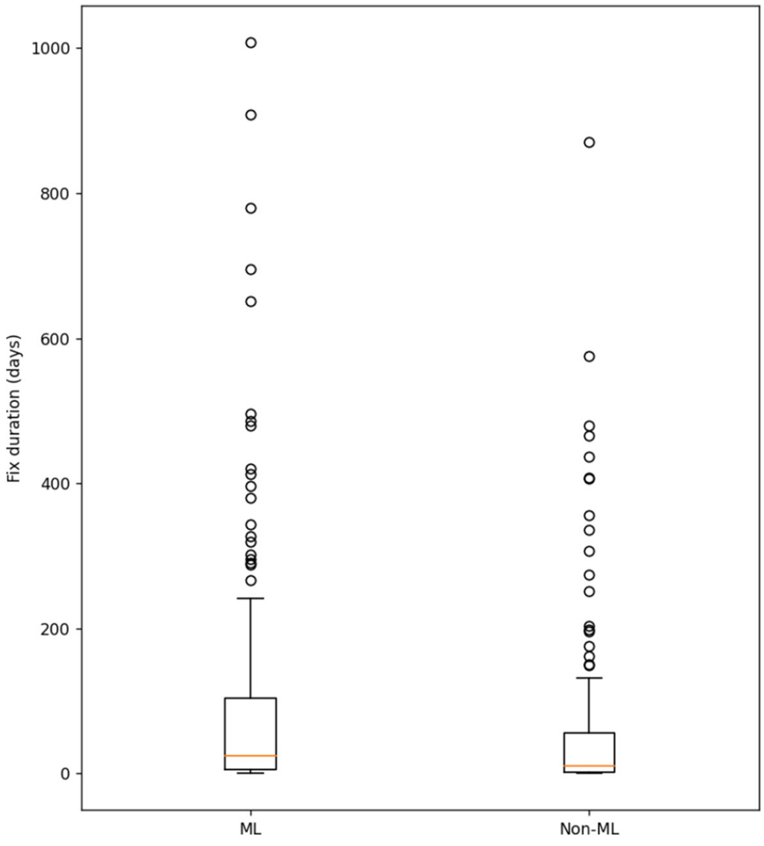


Fig. 6 ML and non-ML issues using boxplot

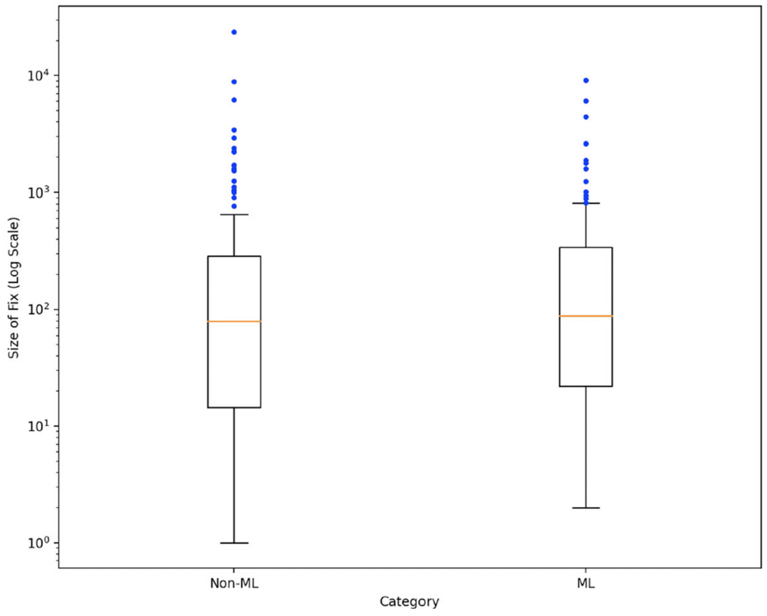


Fig. 7 Comparison of Size of Fix using Log Scale: ML vs non-ML issues

5.3 RQ3. How Does the Distribution of Different ML Issue Categories Compare in Terms of Resolution Time and Size of Fix?

Overall, we analysed 147 ML issues across 6 categories, we found that there are no significant differences between the ML categories for size of fix or resolution time.

Table 7 shows the number of issues per category, GPU Usage is the category with only 2 issues found. The Kruskal-Wallis H-test (MacFarland et al. 2016) (a non-parametric statistical test that compares three or more unpaired groups to determine if there are significant differences between them) was used to test the null hypothesis (H0) that all groups come from populations with the same median.

For resolution time between ML categories, we fail to reject the null hypothesis (p=0.47, α=0.05). This means that based on the data, there's no significant evidence to conclude that the median fix duration differs across the categories of machine learning issues.

Table 8 Descriptive Statistics for Size of Fix

Descriptive Statistics	ML Issues	Non-ML Issues
Count	147	147
Mean	484 lines	557 lines
Standard Deviation	1340 lines	2186 lines
Minimum	2 lines	1 line
25th Percentile	22 lines	15 lines
Median (50th %ile)	88 lines	80 lines
75th Percentile	342 lines	285 lines
Maximum	9126 lines	23767 lines

For size of fix, again, we fail to reject the null hypothesis (p=0.27, α=0.05). This means that there is no significant evidence to conclude that the median size of fix (number of lines of code changed) differs across the categories of machine learning issues.

6 Discussion

Our findings suggest a nuanced approach for engineering teams to adapt their practices for applied ML projects. First, our results show that the fix duration of ML issues is longer than non-ML issues. Project teams should allocate additional time when allocating ML issues although whether ML issues are harder to estimate remains future work. The reason why ML issues take longer could be that debugging in ML issues is performance-based and not solely to locate and fix bugs (Wan et al. 2019). Experimentation is also needed to discover where the ML bug is located as per the updated taxonomy (training process, model, GPU usage, or tensor and input).

Second, our results show no statistically significant differences in the size of fix between ML and non-ML issues. This shows that the fix duration is not directly proportional to the scale of the code modifications. This further adds weight to the hypothesis that training time is where the additional time is spent. An alternative explanation is that understanding and locating the defect is where the time is spent. Our research finding confirms our hypothesis that the ML life cycle is stretched due to the new problems that arise in each stage of the issue life cycle. In future research, we look further into each stage of the issue lifecycle and identify the challenges in each stage that caused the life cycle to be prolonged.

Another possible explanation for the significant difference in resolution time and not size of fix between ML and non-ML issues is the longer issue reporting process and the information evolution. If ML issues take longer to resolve than non-ML issues, but the size of fix is not different, that means the extra time is spent elsewhere and not writing code. We predict that the extra components in ML systems, i.e. data processing pipeline, training and inferences pipeline, model and monitoring have caused the developer to not report sufficient information in the initial issue reports. This leads to time-consuming and iterative interactions between the issue fixers and the reporters to gather and build up information. To tackle this issue, in future research, we plan to empirically investigate the attributes required in issue templates used by applied AI project owners. Future research will take interactions, i.e. comments in the issue timeline, into consideration.

The finding that there were "Other" types of ML bugs not mentioned in existing taxonomies by Humbatova et al. (2020) is empirical evidence for the need for a further research direction into understanding the types of issues and assist with triaging (categorising and assigning developers) for ML-specific issues.

In contrast with previous studies in understanding ML issues (Sun et al. 2017; Humbatova et al. 2020), our study focused on issues in applied ML applications, not ML frameworks. Our experience from applying a taxonomy for ML issues and the questions raised from our study shows that further research is needed to better understand ML issues. Furthermore, our project filtering criteria have substantially cut down the number of projects included in the analysis. In future work, we plan to loosen up the criteria and conduct a large-scale analysis of ML vs non-ML issues to further validate our results.

Research on understanding software issues in open-source projects focused on building and testing models for resolution time prediction (Vieira et al. 2022; Ardimento and Boffoli 2022). However, with our new finding that ML issues take longer time to resolve than non-ML issues, future work is needed to validate these prediction models on applied ML projects. Also, we believe that the findings indicate a lack of tooling catered to ML-specific applications. The resolution time of issues in open-source software has been studied before through empirical analysis, a previous study suggested that issues with fewer stakeholders are resolved faster than those with more stakeholders Nguyen Duc et al. (2011). Software quality and code maintainability ratings are correlated with faster issue resolution of defects and enhancement Bijlsma et al. (2012). Furthermore, based on an analysis of 14000 issue reports taken from 34 open source projects, issue resolution time was found to depend on the maintenance types, i.e. corrective, adaptive, perfective or preventive maintenance Murgia et al. (2014). However, we do not think these results will bias our research findings because we sampled an equal number of ML and non-ML issues per project in our analysis. Additionally, our selection of ML issues per project is independent of the issue types.

Developers have different expertise levels when it comes to bug fixing based on their unique skill set and specific domain knowledge. However, we do not think this will bias the research results because the pool of developers who fix issues in each project is the same. Thus, our study results will only be biased if open-source applied AI projects have different sub-teams for fixing ML-specific issues vs other types of issues and these sub-teams differ based on expertise. We do not have any evidence to suggest that there are different sub-teams for fixing ML-specific issues in open-source applied AI projects or that people working on ML-specific issues would be slower or faster than people working on other types of issues as a result of expertise differences, we don't investigate this aspect in the study, though accept that it could be an alternative explanation for the longer resolution time of non-ML issues.

Code can vary significantly between developers, even when they are working on the same problem, leading to differences in time and code size. It is true that developers do not always write identical code, we do not think this will bias our result because we are considering the mean size of fix. Also, the sample size is not small, so we can generalise the result. In future work, we plan to examine the interrelationship of the expertise of the developer with ML and non-ML resolution time and code size is an interesting idea for future work.

In future work, we plan to scale the investigation by studying a larger dataset to see whether the results generalise beyond the sample investigated. Understand the human element further through surveys and interviews, i.e. what do developers find different between fixing ML vs non-ml issues, what about the data preparation effort that is not going to be reflected in the commit history, is there data repositories and other repositories for an ML that is not public. Furthermore, we plan to do an investigation into how the types of issues influence the resolution time, there are insufficient data from our study to be conclusive.

7 Threats to Validity

In this section, we will discuss the validity threats in three key areas: internal validity, which examines the accuracy of our findings within the dataset; external validity, concerning the generalizability of our results; and construct validity, which assesses the suitability of our measurement and interpretation methods. Addressing these threats is essential to ensure the credibility and relevance of our study in the broader context of ML and software development.

7.1 Internal Validity

The size of fix may be misleading in some cases, such as a project refactor. In particular, past research identified cases of repositories that encoded data as Python files, leading to source files with over 50,000 lines of code (Simmons et al. 2020). To lessen the impact of this risk, we make use of non-parametric tests (e.g., that test for a difference of the median rather than the mean, and are thus less sensitive to outliers). Furthermore, our analysis involves visual inspection of the distributions, which allows identifying outliers and investigation of possible causes.

Some of the non-ML issues could still be ML-related but didn't make use of one of the three deep learning libraries we checked for. However, our selection methodology ensures that projects that didn't make use of Tensorflow, Keras, or PyTorch at all would have had 0 ML issues and therefore no non-ML issues would be selected either.

7.2 External Validity

Our analysis assumes that ML issues are reported using the issue tracker, that the pull request fixing the issue is linked to the relevant issue report, and that the issue is closed after resolution. However, ML development may follow a less formal process than traditional software, for example, may be conducted in notebooks using informal versioning practices (Rule et al. 2018), in which case ML issues with linked pull requests containing fixes may not be present in the issue tracker. Future work will be to investigate whether ML issues are being reported and fixed through informal processes.

The scope of our study is limited to ML projects written in Python. This was necessary so that we could identify ML code modules in a consistent manner (based on the libraries they import). Python is the most popular language for open-source ML projects on Github (Braiek et al. 2018), but restricting the scope to a single language may still limit the generality of our findings.

Furthermore, we only analyse open-source projects, and thus may not capture the issues faced by companies running proprietary ML pipelines in production. In particular, ML issues such as data shift are inevitable for companies running ML in production, but may not occur in open-source ML projects that serve as a library or are demonstrated on static datasets.

7.3 Construct Validity

Our study measures the resolution time and size of fix. However, the interpretation of these constructs is nuanced. A long resolution time could be because the issue was difficult to resolve, or it could simply be because the issue was considered low priority by developers. Similarly, an issue that is of a large size may indicate that it was more complex to resolve, or could simply point to poor coding practices.

8 Conclusion

In conclusion, our analysis revealed that ML issues take longer to resolve than non-ML issues by a median of 14 days while there is no significant difference between the size of fix, which is associated with the developer's effort. These results suggest that the time difference is spent elsewhere and is yet to be found and confirmed. Although we have pointed out the challenges that arise in each stage of an issue's life cycle due to the unique characteristics of ML, further investigation is required to identify the causes. Our study provides empirical evidence that existing SE resource allocation tools and effort prediction models need to be re-evaluated before applying in ML applications. Furthermore, our study highlighted the need for future work to investigate what stages in the issue life cycle are stretched causing the resolution time to prolong. Because the size of fix between ML and non-ML has no significant difference, while the median difference in resolution time is 14 days, with ML issues taking longer to resolve. We predict the information evolution in the ML issue reporting process is what caused the resolving time to increase, developers reporting ML issues require more back-and-forth interactions with the fixers to gather sufficient information to resolve the issue.

Our analysis reveals that there is a statistically significant association between our study and Humbatova et al. (2020) study in terms of the frequency of ML issues. Although the level of association is modest, this led us to continue using the taxonomy to investigate the differences between the 6 different ML issue categories in terms of resolution time and size of fix. In total, we have manually labelled 147 ML issues; however, no significant differences are found among the 6 categories. Notably, our analysis revealed that GPU Usage and Third-party Usage exhibited the lowest occurrence of issues within the ML context.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10664-024-10467-3.

Acknowledgements The first author of this paper would like to acknowledge that the research activity is funded by a postgraduate scholarship provided by the Applied Artificial Intelligence Institute (A2I2), Deakin University, Melbourne Australia.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

Data Availability The datasets generated during and/or analysed during the current study are available in the https://github.com/DungLai/EMSE_dataset.

Declarations

Conflicts of interests The authors declare that there are no conflicts of interest to disclose in relation to this research.

References

Akbarinasaji S, Caglayan B, Bener A (2018) Predicting bug-fixing time: A replication study using an open source software project. J Syst Softw 136:173–186

Al-Zubaidi WHA, Dam HK, Ghose A, Li X (2017) Multi-objective search-based approach to estimate issue resolution time. In: Proceedings of the 13th international conference on predictive models and data analytics in software engineering, pp 53–62

Amershi S, Begel A, Bird C, DeLine R, Gall H, Kamar E, Nagappan N, Nushi B, Zimmermann T (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, pp 291–300

Ardimento P, Boffoli N (2022) A supervised generative topic model to predict bug-fixing time on open source software projects. In: ENASE, pp 233–240

Arya D, Wang W, Guo JL, Cheng J (2019) Analysis and detection of information types of open source software issue discussions. In: 2019 IEEE/ACM 41st International conference on software engineering (ICSE), IEEE, pp 454–464

Baskaran A, Kautz EJ, Chowdhary A, Ma W, Yener B, Lewis DJ (2021) Adoption of image-driven machine learning for microstructure characterization and materials design: A perspective. Jom 73:3639–3657

Bhattacharya P, Ulanova L, Neamtiu I, Koduru SC (2013) An empirical analysis of bug reports and bug fixing in open source android apps. In: 2013 17th European conference on software maintenance and reengineering, IEEE, pp 133–143

Bijlsma D, Ferreira MA, Luijten B, Visser J (2012) Faster issue resolution with higher technical quality of software. Softw Qual J 20:265–285

Biswas S, Islam MJ, Huang Y, Rajan H (2019) Boa meets python: A boa dataset of data science software in python language. In: 2019 IEEE/ACM 16th international conference on mining software repositories (MSR), IEEE, pp 577–581

Braiek HB, Khomh F (2020) On testing machine learning programs. J Syst Softw 164:110542

Braiek HB, Khomh F, Adams B (2018) The open-closed principle of modern machine learning frameworks. In: Proceedings of the 15th international conference on mining software repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018, ACM, pp 353–363. https://doi.org/10.1145/3196398.3196445

Chen J, Liang Y, Shen Q, Jiang J (2022) Toward understanding deep learning framework bugs. arXiv:2203.04026

Chou A, Yang J, Chelf B, Hallem S, Engler D (2001) An empirical study of operating systems errors. In: Proceedings of the eighteenth ACM symposium on Operating systems principles, pp 73–88

Cramér H (1999) Mathematical methods of statistics, vol 26. Princeton University Press

Cureton EE (1956) Rank-biserial correlation. Psychometrika 21(3):287–290

Davies S, Roper M (2014) What's in a bug report? In: Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement, ESEM '14. https://doi.org/10.1145/2652524.2652541,

Du J, Ren X, Li H, Jiang F, Yu X (2022) Prediction of bug-fixing time based on distinguishable sequences fusion in open source software. J Softw Evol Process e2443

El Asri I, Kerzazi N, Uddin G, Khomh F, Idrissi MJ (2019) An empirical study of sentiments in code reviews. Inf Softw Technol 114:37–54

Galin D (2004) Software quality assurance: from theory to implementation. Pearson education

Gegick M, Rotella P, Xie T (2010) Identifying security bug reports via text mining: an industrial case study. In: 2010 7th IEEE working conference on mining software repositories (MSR 2010), IEEE, pp 11–20

Ghanavati M, Costa D, Seboek J, Lo D, Andrzejak A (2020) Memory and resource leak defects and their repairs in java projects. Empir Softw Eng 25(1):678–718

Gonzalez D, Zimmermann T, Nagappan N (2020) The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on github. In: Proceedings of the 17th international conference on mining software repositories, pp 431–442

Humbatova N, Jahangirova G, Bavota G, Riccio V, Stocco A, Tonella P (2020) Taxonomy of real faults in deep learning systems. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering, pp 1110–1121

Islam MJ, Nguyen G, Pan R, Rajan H (2019) A comprehensive study on deep learning bug characteristics. In: Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp 510–520

Janssen A, Bennis FC, Mathôt RA (2022) Adoption of machine learning in pharmacometrics: an overview of recent implementations and their considerations. Pharmaceutics 14(9):1814

Jia L, Zhong H, Wang X, Huang L, Lu X (2020) An empirical study on bugs inside tensorflow. In: Database systems for advanced applications: 25th international conference, DASFAA 2020, Jeju, South Korea, September 24–27, 2020, Proceedings, Part I 25, Springer, pp 604–620

Kim M, Kim Y, Lee E (2021) Denchmark: a bug benchmark of deep learning-related software. In: 2021 IEEE/ACM 18th international conference on mining software repositories (MSR), IEEE, pp 540–544

Lai TD, Simmons A, Barnett S, Schneider JG, Vasa R (2022) Comparative analysis of real bugs in open-source machine learning projects–a registered report. arXiv:2209.09932

Lal S, Sureka A (2012) Comparison of seven bug report types: A case-study of google chrome browser project. In: 2012 19th Asia-Pacific software engineering conference, IEEE, vol 1, pp 517–526

Li Z, Tan L, Wang X, Lu S, Zhou Y, Zhai C (2006) Have things changed now? an empirical study of bug characteristics in modern open source software. In: Proceedings of the 1st workshop on architectural and system support for improving software dependability, pp 25–33

Light RJ (1971) Measures of response agreement for qualitative data: some generalizations and alternatives. Psychol Bull 76(5):365

Liu C, Lu J, Li G, Yuan T, Li L, Tan F, Yang J, You L, Xue J (2021) Detecting tensorflow program bugs in real-world industrial environment. In: 2021 36th IEEE/ACM International conference on automated software engineering (ASE), IEEE, pp 55–66

Lourenço R, Freire J, Shasha D (2019) Debugging machine learning pipelines. In: Proceedings of the 3rd International workshop on data management for end-to-end machine learning, pp 1–10

Lu J, Liu A, Dong F, Gu F, Gama J, Zhang G (2018) Learning under concept drift: A review. IEEE Trans Knowl Data Eng 31(12):2346–2363

MacFarland TW, Yates JM, MacFarland TW, Yates JM (2016) Kruskal–wallis h-test for oneway analysis of variance (anova) by ranks. Introduction to nonparametric statistics for the biological sciences using R, pp 177–211

Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 50–60

McKnight PE, Najab J (2010) Mann-whitney u test. The Corsini encyclopedia of psychology, pp 1

Mehrabi N, Morstatter F, Saxena N, Lerman K, Galstyan A (2021) A survey on bias and fairness in machine learning. ACM Comput Surv (CSUR) 54(6):1–35

Morovati MM, Nikanjam A, Khomh F, Jiang ZM (2023) Bugs in machine learning-based systems: a faultload benchmark. Empir Softw Eng 28(3):62

Murgia A, Concas G, Tonelli R, Ortu M, Demeyer S, Marchesi M (2014) On the influence of maintenance activity types on the issue resolution time. In: Proceedings of the 10th international conference on predictive models in software engineering, pp 12–21

Nguyen Duc A, Cruzes DS, Ayala C, Conradi R (2011) Impact of stakeholder type and collaboration on issue resolution time in oss projects. In: IFIP International conference on open source systems, Springer, pp 1–16

Nikanjam A, Braiek HB, Morovati MM, Khomh F (2021) Automatic fault detection for deep learning programs using graph transformations. ACM Trans Softw Eng Methodol (TOSEM) 31(1):1–27

Parker B, Khan L (2015) Detecting and tracking concept class drift and emergence in non-stationary fast data streams. In: Proceedings of the AAAI conference on artificial intelligence, vol 29

Rawindaran N, Jayal A, Prakash E (2021) Machine learning cybersecurity adoption in small and medium enterprises in developed countries. Computers 10(11):150

Rule A, Tabard A, Hollan JD (2018) Exploration and explanation in Computational notebooks. Conference on human factors in computing systems - proceedings 2018-April:1–12. https://doi.org/10.1145/3173574.3173606

Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, Chaudhary V, Young M (2014) Machine learning: The high interest credit card of technical debt. Softw Eng Mach Learn

Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, Chaudhary V, Young M, Crespo JF, Dennison D (2015) Hidden technical debt in machine learning systems. Adv Neural Inf Process Syst 28:2503–2511

Seymoens T, Ongenae F, Jacobs A, Verstichel S, Ackaert A (2018) A methodology to involve domain experts and machine learning techniques in the design of human-centered algorithms. In: IFIP working conference on human work interaction design, Springer, pp 200–214

Shen Q, Ma H, Chen J, Tian Y, Cheung SC, Chen X (2021) A comprehensive study of deep learning compiler bugs. In: Proceedings of the 29th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp 968–980

Shihab E, Kamei Y, Adams B, Hassan AE (2013) Is lines of code a good measure of effort in effort-aware models? Inf Softw Technol 55(11):1981–1993

Simmons AJ, Barnett S, Rivera-Villicana J, Bajaj A, Vasa R (2020) A large-scale comparative analysis of Coding Standard conformance in Open-Source Data Science projects. In: International symposium on empirical software engineering and measurement. https://doi.org/10.1145/3382494.3410680

Sun X, Zhou T, Li G, Hu J, Yang H, Li B (2017) An empirical study on real bugs for machine learning programs. In: 2017 24th Asia-Pacific software engineering conference (APSEC), IEEE, pp 348–357

Tambon F, Nikanjam A, An L, Khomh F, Antoniol G (2021a) Silent bugs in deep learning frameworks: An empirical study of keras and tensorflow. arXiv:2112.13314

Tambon F, Nikanjam A, An L, Khomh F, Antoniol G (2021b) Silent bugs in deep learning frameworks: an empirical study of keras and tensorflow. arXiv:2112.13314

Tan L, Liu C, Li Z, Wang X, Zhou Y, Zhai C (2014) Bug characteristics in open source software. Empir Softw Eng 19(6):1665–1705

Thung F, Wang S, Lo D, Jiang L (2012) An empirical study of bugs in machine learning systems. In: 2012 IEEE 23rd international symposium on software reliability engineering, IEEE, pp 271–280

Vieira R, Mesquita D, Mattos CL, Britto R, Rocha L, Gomes J (2022) Bayesian analysis of bug-fixing time using report data. In: Proceedings of the 16th ACM/IEEE international symposium on empirical software engineering and measurement, pp 57–68

Wan Z, Xia X, Lo D, Murphy GC (2019) How does machine learning change software development practices? IEEE Trans Softw Eng 47(9):1857–1871
Wang S, Minku LL, Yao X (2018) A Systematic Study of Online Class Imbalance Learning with Concept Drift. IEEE Trans Neural Netw Learn Syst 29(10):4802–4821. https://doi.org/10.1109/TNNLS.2017.2771290
Wardat M, Le W, Rajan H (2021) Deeplocalize: fault localization for deep neural networks. In: 2021 IEEE/ACM 43rd international conference on software engineering (ICSE), IEEE, pp 251–262
Wardat M, Cruz BD, Le W, Rajan H (2022) Deepdiagnosis: automatically diagnosing faults and recommending actionable fixes in deep learning programs. In: 2022 IEEE/ACM 44th international conference on software engineering (ICSE), IEEE, pp 561–572
Xiang W, Musau P, Wild AA, Lopez DM, Hamilton N, Yang X, Rosenfeld J, Johnson TT (2018) Verification for machine learning, autonomy, and neural networks survey. arXiv:1810.01989
Yan M, Chen J, Zhang X, Tan L, Wang G, Wang Z (2021) Exposing numerical bugs in deep learning via gradient back-propagation. In: Proceedings of the 29th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp 627–638
Zhang R, Xiao W, Zhang H, Liu Y, Lin H, Yang M (2020) An empirical study on program failures of deep learning jobs. In: 2020 IEEE/ACM 42nd international conference on software engineering (ICSE), IEEE, pp 1159–1170
Zhang T, Gao C, Ma L, Lyu M, Kim M (2019) An empirical study of common challenges in developing deep learning applications. In: 2019 IEEE 30th international symposium on software reliability engineering (ISSRE), IEEE, pp 104–115
Zhang Y, Chen Y, Cheung SC, Xiong Y, Zhang L (2018) An empirical study on tensorflow program bugs. In: Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis, pp 129–140
Zhou L, Pan S, Wang J, Vasilakos AV (2017) Machine learning on big data: Opportunities and challenges. Neurocomputing 237:350–361
Zhu W, Godfrey MW (2021) Mea culpa: How developers fix their own simple bugs differently from other developers. In: 2021 IEEE/ACM 18th international conference on mining software repositories (MSR), IEEE, pp 515–519

Authors and Affiliations

Tuan Dung Lai[1] · Anj Simmons[1] · Scott Barnett[1] · Jean-Guy Schneider[2] · Rajesh Vasa[1]

Anj Simmons
a.simmons@deakin.edu.au

Scott Barnett
scott.barnett@deakin.edu.au

Jean-Guy Schneider
Jean-Guy.Schneider@monash.edu

Rajesh Vasa
rajesh.vasa@deakin.edu.au

[1] Applied Artificial Intelligence Institute, Deakin University, Geelong, Australia

[2] Faculty of Information Technology, Monash University, Clayton, Australia