An Empirical Study on ML DevOps Adoption Trends, Efforts, and Benefits Analysis

1st Dhia Elhaq Rzig
Computer and Information Science Department
University of Michigan - Dearborn
Dearborn, USA
dhiarzig@umich.edu

2nd Foyzul Hassan
Computer and Information Science Department
University of Michigan - Dearborn
Dearborn, USA
foyzul@umich.edu

3rd Marouane Kessentini
School Of Engineering And Computer Science
Oakland University
Rochester, USA
kessentini@oakland.edu

***Abstract*—Context: Machine Learning (ML), including Deep Learning(DL), based systems, have become ubiquitous in today's solutions to many real-world problems. ML-based approaches are being applied to solve complex problems such as autonomous driving, recommendation systems, etc. Objective: To improve the quality and deliverability of ML-based applications, the software development community is adopting state-of-the-art DevOps practices within them. However, we currently lack knowledge about the DevOps adoption trends, maintenance efforts and benefits among ML-based projects, and this work attempts to remedy this knowledge-gap. Method: In this research work, we conducted a large-scale empirical analysis on 4031 ML projects, including 1116 ML Tools and 2915 ML Applied projects to quantify DevOps adoption, maintenance effort and benefits. To characterize the development behaviors, we performed configuration-script analysis and commit-change-analysis on DevOps configuration files. To compare the characteristics of ML DevOps to those of traditional software projects, we performed the same analysis on 4076 non-ML projects. Results: Our analysis identified that ML projects, more specifically ML-Applied projects, have a slower, lower, and less efficient adoption of DevOps tools in general. DevOps configuration files in ML Applied projects tended to experience more frequent changes than ML-Tool projects and were less likely to occur in conjunction with build and bug fixes. It's also evident that adopting DevOps in ML projects correlates with an increase in development productivity, code quality, and a decrease in bug resolution time, especially in ML-Applied projects which have the most to gain by adopting these tools. Conclusion: We identified the characteristics and improvement scopes of ML DevOps, such as the slower adoption of DevOps in certain ML projects, and the need for automatic configuration synchronization tools for these projects. We also identified the improvements the productivity of ML teams and projects associated with DevOps adoption, including better code quality, more frequent code sharing and integration and faster issue resolution.**

***Index Terms*—machine learning, DevOps**

I. Introduction

Recently, Machine Learning (ML), including Deep Learning (DL), has become prevalent with many applications: Alzheimer's disease diagnosis [1], Blood glucose prediction in diabetics [2], Autonomous-driving cars [3], Loan approval prediction [4], etc. The Worldwide Developer Population and Demographic Study 2019 [5] estimates that approximately 7 million developers have used ML in their development activity, and expects another 9.5 million developers to use ML in the next twelve months. Although ML-based approaches are becoming widely adopted by the industry as well as the research community, one major challenge remains: the integration of ML components in complex production systems and processes while maintaining their reliability and efficiency in the context of continuously evolving ML projects.

To improve the software delivery process, a closer collaboration between the development and operations teams, known as DevOps [6] has become popular within the software engineering community. DevOps is a modern software engineering paradigm that brings changes to production processes with the approach of automating the building, testing, code analysis and deployment of software. A recent GitHub study [7] discovered that highly-performing DevOps teams recover from downtime 96 times faster, have a 5 times lower failure rate, and a 46 times more frequent deployment rate. While DevOps practices are slowly becoming more common and standardized for traditional software products [8], the state of DevOps within ML-based projects, the advantages, and the challenges it brings, still require more study within the research community.

Recently, there have been many works focused on ML DevOps support. MLFlow [9] and Amazon SageMaker [10] were designed to improve the workflow of ML project development, which involves the data collection, data preparation, model definition and training, and results-testing [11]. Package managers such as Spack [12] and EasyBuild [13] were conceived to allow the automatic rebuilding of ML models. Container-based technology such as Docker [14] and Kubernetes [15] has proven apt for shareable models. Aguilar et al. [16] proposed Ease.ml/CI for continuous integration (CI) and data management within ML projects. Fursin et al. [17] proposed CodeReef to perform benchmarking for ML projects and enable their reusable automation. However, the majority of these

tools are still premature, require an important development effort, and can only be used in conjunction with specific ML technologies or frameworks [17, 18, 19]. Prior research [17] also identifies that workflows using these solutions are not easy to put into practice. Moreover, very little is known about ML projects' DevOps adoption and the difficulty of maintaining correctly functioning DevOps tools within them. This motivates our large-scale study on DevOps tools' adoption within ML projects, their maintenance effort and goals, and the benefits they bring.

In order to obtain more information about these aspects, we defined the following research questions:

1) What are the current and historical adoption rates of DevOps Tools for ML and Non-ML projects?
2) What are the maintenance efforts and goals associated with DevOps tools across the different project categories ?
3) What are the advantages of adopting DevOps tools across the different project categories?

In this empirical study, we conducted a large scale analysis on 4031 ML projects that we manually curated from the dataset by Gonzalez et al. [20]. We also performed the same analysis on the 4076 Non-ML projects from the same dataset [20] for comparative purposes.

Our main contributions through this paper can be summarized as follows:

- Characterization of the current and historical adoption of DevOps tools within a subset of popular Open-source ML projects. Indeed, we found that ML Tool projects, which are general purpose projects meant for use by other developers, had similar current and historical DevOps tools' adoption to Non-ML projects, while ML Applied projects, which are specific-purpose projects meant for use by other developers and end-users, had a lower and slower DevOps tools adoption in comparison
- An empirical analysis of the development effort in regards to employing DevOps tools for different types of ML projects. We believe that more DevOps-related development effort is invested within ML Tool projects than ML Applied projects, and that the adoption of certain DevOps tools within these project categories is linked to a larger effort invested by their development teams.
- Characterization of the common goals behind the changes in DevOps configuration files and their other accompanying changes ML projects. We found that ML Tool and Non-ML projects achieve more Bug fixes than ML Applied projects. Both in ML Tool and Non-ML project, this increase in bug fixes is correlated with their adoption of DevOps tools such as Test and Code analysis tools, while this correlation was not found within ML Applied projects. A small percentage of DevOps-altering commits were found to have Build fixes as a goal, and the majority of them were concerned with other miscellaneous changes.
- An empirical analysis of the improvements in the development process resulting from the usage of DevOps tools within ML projects. Across all categories of projects, we found that the adoption of one or more DevOps tools was positively correlated with an increase in commit frequency, merge frequency, code quality, and a reduction of the average issue resolution duration.

The rest of this paper is organized as follows: We start by discussing related works in Section II. After that, in Section III, we discuss the methodology of our analysis, which includes data set selection, DevOps tools classification, and the methods of analysis we used to answer our Research Questions. Section IV presents the results of the empirical analysis within our study and Section V discusses the possible implications of our study. Finally, we discuss the threats to validity and our conclusion in Section VI and Section VII, respectively.

II. Related Work

As DevOps became a modern software engineering paradigm, it received growing attention from the research community [6, 21, 22, 23]. Luz et al. [24] compared different approaches of adopting DevOps and identified the main concerns of DevOps. They believe that collaboration is an important DevOps concern in addition to the more common and equally important tool usage. However, this work mainly focused on interview outcomes rather than an empirical analysis of DevOps as adopted by the software projects. Moving on to guidance on adopting DevOps, Leite et al. [6] analyzed DevOps within general-purpose software projects from a multitude of facets. They developed conceptual maps that described DevOps and linked them to engineering and management perspectives.

McIntosh et al. [25] analyzed Build files, a type of DevOps configuration files, in order to estimate the effort invested by developers to maintain functioning Build systems in 9 open-source and 1 closed source projects. They found that the level of correlation between source files and build files is linked to a project's programming languages. However, their work only covered a limited set of C and Java projects and a handful of build tools, such as Make and ANT. This means that their findings may not apply to projects with other programming languages and other Build and DevOps tools.

However, none of these aforementioned works focus specifically on ML projects or considered them as a specific project-category. We consider this an oversight due to the fundamental differences between ML and Non-ML software projects. While Non-ML projects are given specific solutions in the form of an algorithm designed by their developers to solve a specific problem or set of problems, ML projects are designed to come up with their own solutions, which may be unknown to these projects developers. Indeed, ML projects attempt to solve a problem by analyzing data, testing their findings, evaluating their results, and iterating on these phases. Furthermore, they require new development processes and practices such as data engineering and model management [11, 26, 27], follow different collaboration strategies between their collabora-

tors [20, 28], and may require different approaches to existing software development processes in comparison to traditional software, such as the example of Non-ML software testing being ineffective on ML projects [29].

Lwakatare et al. [11] outlined some of the problems teams face while attempting to integrate ML workflows within DevOps processes, such as the inadequacy of existing code versioning tools for ML artifacts management, and proposed alternative processes to employ DevOps in ML projects. Yet, their work relied on existing literature and expert knowledge when discussing DevOps adoption problems within ML projects, and did not perform empirical analysis to validate the actual factors behind ML projects' success or failure at adopting DevOps.

To analyze ML project development aspects, the work of Gonzalez et al. [20] conducted a large-scale empirical study of Open-source ML Tools (700) and Applications (4,524) hosted on GitHub. For comparative purposes, they also analyzed 4,101 Non-ML projects. Their work provided insight into collaboration and autonomy rates in development teams and identified ML Applied projects as the most autonomous, Non-ML projects as less autonomous, and ML Tool projects as the least autonomous. However, we uncovered problems with their data-set in regards to the selection and classification of ML projects. Furthermore, their work is more interested in analyzing development practices and collaboration aspects of the projects rather than analyzing their DevOps adoption and DevOps practices.

Focusing more on the intersection of DevOps and ML projects, Karlaš et al. [29] discussed the shortcomings and the lack of support of existing CI tools of ML projects in practice. Their work proposed implementation details that attempted to solidify and build on existing theoretical concepts concerning CI systems for ML projects. However, their work did not consider other aspects of DevOps processes such as Code Analyzers, Build systems, Deployment Automation, etc.

In contrast to existing works, our goal within this paper is to analyze the adoption rates and trends of all DevOps components such as Code Analyzers, Build systems, Continuous Integration systems, etc., within ML projects, to characterize their associated maintenance efforts, goals, as well as the advantages they bring to the projects that adopt them.

III. Methodology

In this section, we discuss the different steps of our analysis which includes: Data set selection, performed through a mix of automatic and manual steps, DevOps tool classification, performed via a study of existing research works around the types of these tools, and our methods of analysis, which relied on different phases of exploration and multi-pronged analyses to empower us to answer our different research questions.

A. Data Set Collection

For this work, our goal was to analyze DevOps tools' adoption within a set of active and currently developed Machine Learning (ML) software projects, referred to as *ML* projects, and a comparison set of Software Projects that do not use ML, referred to as *Non-ML* projects. However, preparing a data-set of ML projects and Non-ML projects is effort-intensive, and not the goal of this work. Initially, we opted for a recent dataset proposed by Gonzalez et al. [20] for our analysis. This dataset was supposed to contain 5224 ML projects and 4101 non-ML projects for comparative purposes. However, we found several problems with it such as the inclusion of toy projects, learning guides and other types of projects that were supposedly manually removed from it, as well as the misclassification between the two subsets of ML projects. To resolve this problem, two authors re-curated the ML projects by reading their descriptions on their main GitHub page, and any websites linked to by that page. The resulting new dataset we used within this work contained:

1) **1116 ML Tool** projects: frameworks and libraries such as Tensorflow, which can be used by developers to solve a variety of problems. These projects are generally only usable via an API.
2) **2915 ML Applied** projects: Applications and libraries that use ML components or libraries from the ML Tool projects, to solve a specific problem. FaceSwap is an example of an application and Document-Classifier-LSTM is an example of a library. These projects may offer a combination of a UI and an API.
3) **4076 Non-ML** projects: A comparison set of classic software projects that don't use ML. These projects may offer a combination of a UI and an API.

In addition, we used the GitHub API [30] in order to collect the following information about each project in our set: Age In days, Number of Stars, Number of Forks, Team size, Number of Pull Requests open, Number of Pull Requests merged, Number of Pull Requests rejected, Number of Core Pull Requests Open , Number of Core Pull Requests Merged, Number of Core Pull Requests Rejected, and Number of Issues open. The project properties with *Core* in their name refer to those managed by core developers and other project insiders, for example, *Number of Core Pull Requests Open* refers to the Number of PRs opened by project insiders. Vasilescu et al. [31] chose these data-points as representative characteristics of each project and its activity, and their works' validation by the research community indicate the validity of their variable selection.We especially note that the Age In days, Number of Stars, Number of Forks, Team size, are used as numerical estimators of the size of the projects in our work, similar to other works [31, 32, 33]. We collected these project properties to enrich the data-set and facilitate the statistical analyses within this work such as ANCOVA [34, 35].

B. DevOps Tools Classification

DevOps has many competing definitions, consequentially, there is no consensus on how to determine whether or not a project is employing DevOps. Prior research [24, 36, 37, 38] on DevOps and DevOps tools also identified the same challenge. To circumvent this problem, we used the adoption of DevOps tools as an indicator of the adoption of DevOps, and we focused on analyzing these tools and their usage within our

chosen project-set. DevOps tools are defined by Leite et al. [6] as the tools pursuing human collaboration across different departments, enabling continuous delivery, and maintaining software reliability. We opted for this definition as it is similar to those found within other research works concerning DevOps and DevOps tools [39, 40, 41]. Initially, we considered the list of DevOps tools determined by Leite et al. [6]. However, since this list was formed by analyzing traditional software, we wanted to expand the number of tools within our analysis to avoid missing any DevOps tools that are more popular with ML projects. To expand our list of DevOps tools to consider within this work, we followed the method outlined in Section III-C1, to discover new DevOps tools in-use within our projects but not described within previous works. We classified the different tools we found into 6 categories:

1) **Build Tools:** Responsible for generating packages meant for deployment, also referred to as builds. They are also generally responsible for generating other artifacts and providing feedback to developers using only the source code as input.
2) **Continuous Integration (CI) Tools:** Responsible for the orchestration of several steps that ensure the development pipeline and automation of development tasks such as package generation, automated test execution, and deployment to both development and production environments.
3) **Deployment Automation Tools:** Make use of certain outputs of the continuous delivery process. They are employed in the deployment stages in order to allow frequent and reliable deployment processes.
4) **Monitoring and Logging Tools:** Responsible for tracking non-functional properties, such as performance, availability, scalability, resilience, and reliability.
5) **Test Tools:** Validate the functionality of software, and identify possible errors, or missing requirements.
6) **Code Analysis Tools:** Static code analyzers that perform several operations, such as code coverage, static error detection, etc.

The Code Analysis category was proposed by Yin & Filkov et al. [42], and Leite et al. [6] coined the first 4 categories and while they considered Test tools as a part of the Build category, we opted to consider them as a separate category due to the difference in their respective goals, as detailed within the definitions above. We didn't consider Source code management tools in our analysis because the projects in our dataset were all collected from GitHub. Furthermore, our analysis in Section III-C1 did not uncover any ML-specific tools. To further verify the absence of usage of these tools, we performed an automatic search for the configuration files of some ML-specific tools such as MLFlow [9], Amazon SageMaker [10] and Spack [12], and we found no evidence of their usage within the two categories of ML projects we considered

C. Methods of Analysis

The overview of our analysis is illustrated in Figure 1.

1) Phase 1: File, Name and Import pattern collection: DevOps configuration files are written in a variety of domain specific languages (DSL). For example, the Maven build specification is written in an XML format, while the Gradle build specification is written in a Groovy-based DSL language. On the other hand, Docker uses a DSL that can only be parsed and recognized by the Docker tool. As a result, static program analysis techniques developed for certain programming languages or DSLs might not be sufficient to detect a large pool of DevOps tools. This lead us to use the configuration file name and path patterns to detect DevOps configuration files. We adapted this method from prior works which employed this approach for IaC and Build artifact files [25, 43, 44]. But first, in order to establish the set of DevOps tools to consider in this work, we considered the list of tools proposed by Leite et al. [6] as a starting point. However, upon realizing its limitations, as discussed within Section III-B, we performed a semi-automatic classification of DevOps configuration files on the top 1000 ML projects and 1000 Non-ML projects based on their GitHub project popularity[1]. First, performed an automatic classification of the files within the repositories of the aforementioned projects using the GitHub Linguist tool [45]. Then, a co-author manually verified the resulting classification, and extracted from it the possible DevOps configuration files by ignoring files with known extensions or names, such as source code and readme files. Libraries.io [46] was then consulted to find the tools corresponding to these configuration files and verify if they corresponded to DevOps tools. Finally, these tools' documentation were examined to extract configuration file name and path patterns that correspond to them. These patterns are then used within the phase described in Section III-C2. However, no such patterns were found for testing tools as they do not rely on specific configuration files. To detect these tools, we identified the testing files within the aforementioned repositories, using the name and path pattern-based method proposed by Zhu et al. [47] Then, the import or import-equivalent (e.g., include, using, etc.) statements within these files were manually checked by 2 co-authors and cross-referenced with the Libraries.io [46] dataset to determine if the modules being imported were testing tools and frameworks. These patterns are used within the phase described in Section III-C2. Overall, we identified 93 DevOps tools via this phase. Figure 2 presents a subset of the tools we identified and processes we used to identify them during our analysis, with a full list available at [48]

2) Phase 2: File System Analysis: Having extracted the file name and path patterns for Build, Continuous Integration, Deployment Automation, Code Analysis and Monitoring and Logging Tools, import-equivalent statements of the Test tools, we used these patterns to verify their adoption within a certain repository. We considered the existence of a configuration file

[1]The project popularity criteria used was a combination of the number of stars and number of watchers

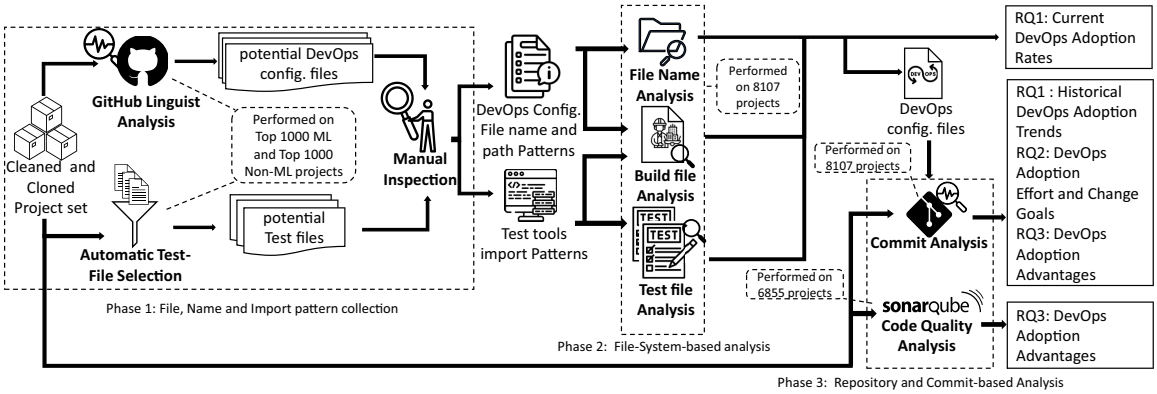

Fig. 1: Overview of our Approach

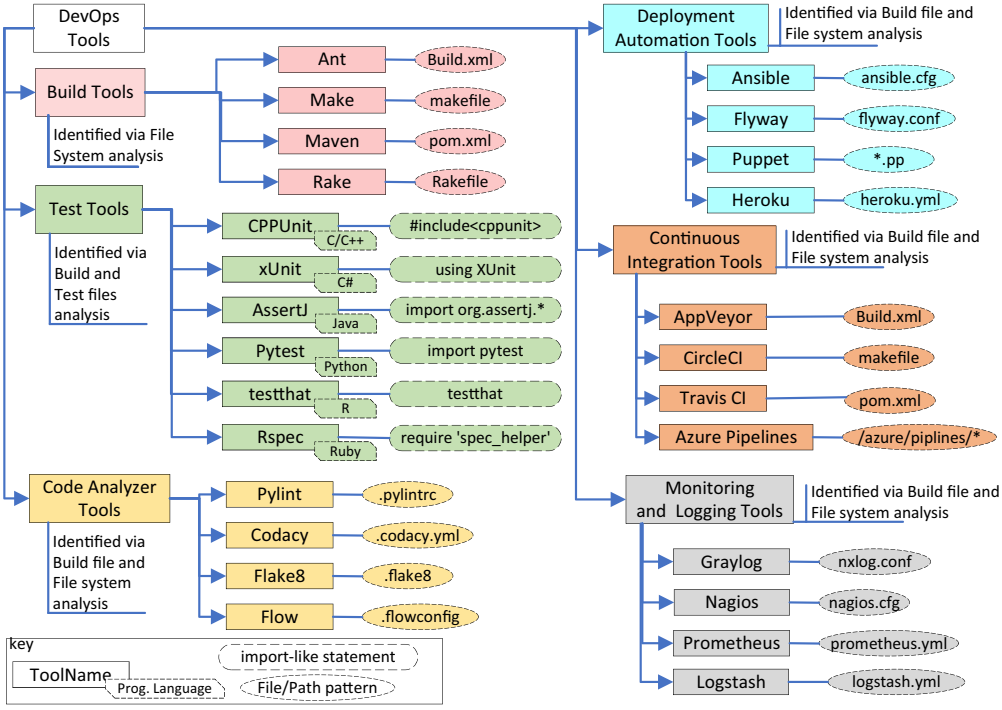

Fig. 2: Subset of DevOps tools, their categories, and their corresponding configuration file name patterns or import statements used to detect their usage.

matching the file name and path patterns of a specific DevOps tool as indicative of that tool's usage within the project. For example, a *pom.xml* file in the project repository indicates that Maven is being used as a Build tool within that project and a *.travis.yml* indicated that the project adopted Travis CI for Continuous Integration. Using the GitPython [49], and PyGitHub [30] libraries, we created a tool that allowed us to access and clone the remote source codes of these projects into a local file system. Then, we analyzed the files of each project and attempted to match them with the aforementioned patterns to detect if the tools corresponding to these patterns were adopted within each project. For the specific case of testing tools, we analyzed the test code files, detected per the method specified by Zhu et al. [47], for the import statements specific to the test file's possible testing tools, which are language specific. For example, if a test file has the `.py` extension, it is identified as a Python file. It is then scanned for the import statements of Python testing tools identified within Sec-

tion III-C1. For example, if the statement `import pytest` is found, the project that contains the test file is assumed to be using the `PyTest` tool. In a software system, a build script is responsible for collecting the necessary dependencies, thus analyzing build scripts can provide important information regarding their usage within a project. For example, Fan et al. [50] relied on build-script analysis to find dependency related errors related to building projects. In addition to the two previously described methods, we relied on the analysis of build scripts and considered a project's dependency on a tool to be indicative of its use within it. For example, if a project specified a dependency on Codecov within its Maven *pom.xml* file, we considered the project to be using the Codecov tool. We used this method to detect the usage of DevOps tools of all categories. The categories of DevOps tools and the methods we used to identify the tools of those categories, as well as a subset of the DevOps tools we considered, and their corresponding file name and path patterns or import patterns are illustrated in Figure 2. To determine the different variables that contribute to DevOps adoption within different project categories, We performed an ANCOVA [35] analysis, a type of GLM regression for models with categorical and continuous variables, using DevOps adoption as a dependent variable and the additional data we collected, detailed in Section III-A, as covariates. This phase allowed us to answer part of **RQ1** regarding the current adoption of DevOps of the different project categories, the project's properties linked to its DevOps adoption, and the most popular DevOps tools of each type in the different project categories we specified. We also used this phase to extract the different DevOps configuration files used within the different phases described in Section III-C3.

3) Phase 3: Repository and Commit-based Analysis: Repositories and their commits contain valuable information about a project's development and maintenance efforts [51]. DevOps tools are meant to be configured and updated via their configuration files, hence, commits affecting these files contain insight into the usage trends and practices of DevOps tools. We extracted the DevOps configuration files via the steps discussed in Section III-C2. We performed our analysis on the Main branch of the different repositories, using the PyDriller [52] tool and Github GraphQL API [53] to obtain additional data not stored in the Git repository, such as the CI status following a commit. While Test files were analyzed within Section III-C2 to extract information about a project's testing framework, which we considered a type of DevOps tool, test files are not considered DevOps configurations files within the scope of this analysis. This is because Test code is very similar to source code and test file changes are highly coupled with source-file changes [54, 55]. In contrast, DevOps configuration files are used to configure the different DevOps tools used within a project, such as Continuous Integration tools.We used this commit-based analysis to answer **RQ1** regarding DevOps historical adoption trends via analyzing our projects' commits, where we assumed the date of the first commit within a project to be the date of its creation, and the date of the addition of the first DevOps configuration file within a project to be an indicator of when it adopted DevOps. We also answered **RQ2** using commit-based analysis via the sub-phases detailed in Section III-C3a, Section III-C3b, and **RQ3** using commit-based analysis and repository-based analysis via the sub-phase Section III-C3c.

a) Phase 3-a: DevOps Adoption Effort: To obtain a better idea about the configuration and maintenance efforts of DevOps tools, we analyzed the commits that modified one or more DevOps configuration files. We calculated the *Commit Ratio* metric, which is similar to the amount of commits metric, used by a number of works to estimate activity within a project [56], but adapted to the context of a specific type of files, to estimate the portion of commits that affect DevOps configuration files. This metric is defined as follows:

Commit Ratio:

$$CommitRatio = \frac{NofC_{DevOps}}{NofC}$$

$NofC_{DevOps}$ is the total number of commits that involved DevOps configuration file(s) and $NofC$ is the total number of commits.

To estimate the size of an update per-file-type within a project, We calculated the *Average Normalized Code Churn* of Source and DevOps configuration files, a commonly used metric [56] that was also previously used in the context of build artifacts [25], and that is superior to other metrics such as Lines of Code (LOC) [57]. This metric is defined as follows:

Average Normalized Code Churn:

$$AvgNormal.CodeCh.(Type, Project) = \frac{\sum_{i=1}^{n} \frac{NBFilesChanged(Type, Project)}{NBFilesExist(Type, Project)}}{NbOfDevMonths}$$

$NbOfDevMonths$ is the number of development months[2]. $NBFilesChanged(Type, Project)$ is the number of either Source code or DevOps configuration files of that changed during a development period, $NBFilesExisted(Type, Project)$ is the number of files of a certain type, source code file or DevOps configuration file, that existed during a development period.

For each project category, we performed 2 ANCOVA [35] analyses, using Commit Ratio as a dependent variable for the first analysis and Normalized Code Churn for the second analysis, and using the covariates presented within Section III-A. In total, this was 6 ANCOVA analyses. We also performed 2 ANOVA analyses to detect any statistical differences concerning these metrics between the different project categories. We used this sub-phase and its associated analyses and metrics to partially answer **RQ2** regarding DevOps adoption efforts.

b) Phase 3-b: DevOps Change Goals: While the Normalized Code Churn and Commit Ratio metrics inform us on the properties of DevOps configuration files changes, they do not reveal the underlying causes of the changes occurring to these DevOps configuration files. To approximate the change goals of DevOps configuration files, we selected the projects that adopted at least a Build and a CI tool, then analyzed their

[2]We considered a development month to be 30 days within this work

commits that affected their DevOps configuration file(s). We analyzed commits from 851 ML Applied projects, 586 ML Tool projects, and 1942 Non-ML projects. We classified the commits' main change goal between 4 different alternates:

- **Bug Fix:** A bug fix is done to remedy a programming bug or error. However, identifying bug-fixing commits in a Git commit-history is a challenging task [58]. To identify this type of commit, we adopted the approach proposed by Ray et al. by scanning commit messages for the keywords ("error","bug","fix","issue","mistake","incorrect","fault","defect","flaw","type") [59].
- **CI Build Fix:** CI Build fix refers to code changes that aim to fix integration failures such as compilation failures, dependency issues, unit test failures, etc. that are reported by CI systems, and also referred to as Build Breakages. To detect these commits, we adopted an approach proposed by Hassan & Wang et al. [60], and that's similar to the approach used by Hyunmin et al. [61] to detect a build-failure resolution. Based on this approach, if a commit changes the CI build status from *Build failure* or *Build error* to *Build success*, we consider the commit a CI-fixing commit. We used the GraphQL Github API [53] to detect the CI build status.
- **Bug and CI Fix:** A commit that meets the criteria of a Bug Fix commits and CI fix commit is considered to be attempting to fix both types of problems.
- **Other changes:** We considered commits that contain neither a bug fix nor a CI fix as commits with the main goal of other miscellaneous changes. These commits may add new functionality, refactor existing code, etc.

Finally, in order to make these measures project-specific, we calculated the percentage of each of the aforementioned commit types out of all the commits of a project.

For each project category, we performed an ANCOVA [35] analysis, using the four goals, Bug and CI fix, Bug fix, CI fix, and Other changes, as dependent variables for the analysis, and using the same covariates as the ANCOVA analysis done within Section III-C1 in addition to the adoption of different DevOps Tool types, such as Build Tool Adoption, CI Tool adoption, etc. In total, this was 3 ANCOVA analyses. We used this sub-phase and its associated analyses to partially answer **RQ2** regarding DevOps change goals

c) Phase 3-c: DevOps Adoption Advantages: Having gained an idea about the properties and goals of the changes performed on DevOps configuration files, we wanted to develop an understanding of the advantages associated with adopting DevOps Tools. To achieve this, we used the metrics of Commit Frequency, Merge Frequency, and Average Issue duration, which also rely on commit-based analysis , and Code Quality, through the widely-used tool SonarQube [62] which relies on repository-based analysis.

DevOps encourages more code sharing via frequent commits and merges, hence Average Commit Frequency and Average Merging Commits Frequency are correlated diretctly to is principles of DevOps. These two metrics are calculated as follows:

Average Commit Frequency:

$$AverageCommitFrequency = \frac{NBofCommits}{NBofDevMonths}$$

$NBofCommits$ is the total number of commits within a project and $NBofDevMonths$ is the total number of development months within a project.

Average Merging Commits Frequency:

$$AverageMergingCommitFrequency = \frac{NBofMergingCommits}{NBofDevMonths}$$

$NBofMergingCommits$ is the total number of merging commits within a project and $NBofDevMonths$ is the total number of development months within a project.

A reduced issue duration is also an expected result of adopting DevOps, since it is claimed to increase the speed and productivity of teams in relation to resolving software issues, making Average Issue Duration a good metric to evaluate this claim. This metric is calculated as :

Average Issue Duration :

$$AvgIssueDuration(Project_A) = \frac{\sum_{i=1}^{n} Duration(Issue_i, Project_A)}{TotalNBIssues(Project_A)}$$

$Duration(Issue_i, Project_A)$ is the duration of an issue i for a project A, $n\ TotalNBIssues(Project_A)$ indicates the number of issues for that project.

Finally, DevOps is associated with an improvement in the quality of the development process, and possibly that of the code-base as well. We used the Maintainability and Reliability code quality metrics as generated by SonarQube to evaluate the quality of the projects within our set.

Prior works [63, 64, 65] used similar metrics and tools to analyze the effectiveness of adopting CI within a number of projects, giving confidence to their effectiveness.

For each project category, we performed 4 ANCOVA [35] analyses, using Average Commit Frequency as a dependent variable for the first analysis, Average Merging Commits Frequency for the second analysis, Average Issue Duration for the third analysis, and Code Quality for the fourth analysis. We used the same covariates as the ANCOVA analysis done within Section III-C1. In total, this was 12 ANCOVA analyses. We also performed 4 ANOVA analyses to detect any statistical differences concerning these metrics between the different project categories. We used this sub-phase and its associated analyses to partially answer **RQ3** regarding DevOps adoption advantages.

IV. Results

A. Adoption rates of DevOps Tools

> ***Research Question 1:*** What are the current and historical adoption rates of DevOps Tools for ML and Non-ML projects?

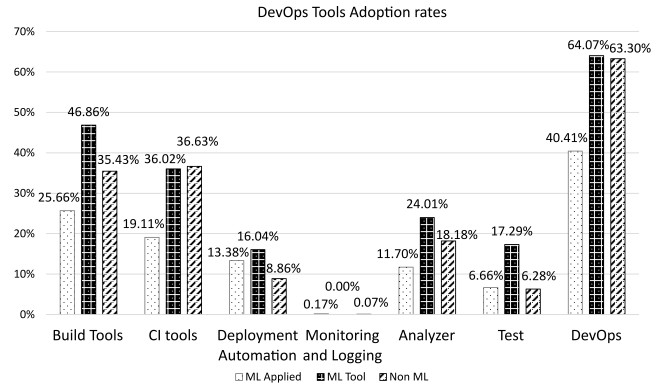

Fig. 3: DevOps Tools Current Adoption Rates

1) DevOps' current Adoption Rates: Adopting DevOps tools and practices within software projects has numerous advantages to the productivity of a development team and the quality of their processes. Since their growth in popularity, DevOps tools are progressively being embraced by independent developers and companies alike. Following our analysis, we were able to confirm this with the high adoption rates of 63.30% for Non-ML projects, and 64.07% for the ML Tool projects. However, ML Applied projects have shown a lower adoption rate of only 40.41%. Focusing on the different DevOps tools categories, ML Tool projects generally had the highest adoption rates across the majority of tool types, with Non-ML projects following as a close second, and Applied projects trailing as the third.

To identify the factors behind adoption of DevOps, we performed an ANCOVA [35] analysis, a type of GLM regression for models with categorical and continuous variables, for each project category. We used DevOps adoption as a dependent variable and the additional data we collected concerning each project, detailed in Section III-A, as covariates. The project-specific data points were: Age In days, Number of Stars, Number of Forks, Team size, Number of Pull Requests open, Number of Pull Requests merged, Number of Pull Requests rejected, Number of Core Pull Requests Open, Number of Core Pull Requests Merged, Number of Core, Pull Requests Rejected, and Number of Issues open.

Using the results of the ANCOVA analysis illustrated within Table I, we found that for ML Applied projects, the most important statistically-significant factors that contribute to DevOps adoption within them were the Age of a project and its Team size. This is indicated via the Partial Eta Square statistic which informs us which variables have the largest effect on the dependent variable, which is a project's adoption of DevOps in our case. Hence, older and larger ML Applied projects are more likely to adopt DevOps. Similar results were found when performing ANCOVA on ML Applied projects while considering as dependent variables each DevOps tool category, except Analyzer and Test tools where only Team size was a determining variable of their adoption of DevOps.

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.007	Intercept of the model
Age In Days	<.001	.027	A project's Age
Team Size	<.001	.008	A project's team-size
N_Pr_Merged	<.001	.008	Number of Pull requests merged
N_Pr_Core_Merged	<.001	.006	Number of Pull requests by core developers merged

R Squared = .119 (Adjusted R Squared = .116)

TABLE I: ANCOVA analysis of DevOps adoption within Applied projects (Only statistically significant† variables are shown)

† Statistically significant variables have a Sig.(P-value) less than 0.05

No statistically significant contributor was determined behind the adoption of monitoring and logging tools by ML Applied projects, most likely due to their low adoption by this project category.

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.131	Intercept of the model
Age In Days	<.001	.023	Age of the project
N_Stars	.036	.004	Number of Stars of project
N_Forks	.016	.006	Number of Forks of project
N_Pr_Open	.020	.005	Number of pull requests open of project

R Squared = .096 (Adjusted R Squared = .087)

TABLE II: ANCOVA analysis of DevOps adoption within Tool projects (Only statistically significant variables are shown)

For ML Tool projects, as illustrated within Table II, Age was statistically significant correlated to their DevOps adoption. Furthermore, the Number of stars and Number of forks also had a significant correlation to their DevOps adoption. It's important to note that while Team size was significantly correlated to DevOps adoption of ML Applied projects, this correlation was not found within Tool projects. Around 50% of ML Tool projects before our re-categorization process were backed by major organization such as Microsoft and IBM [20], and after this process, and we estimate that 30% of these projects are backed by such organizations. This makes all the more surprising the lack of correlation between team-size and DevOps adoption for ML Tool projects, especially considering these organization are more likely to have larger resource and to adopt best practices such as DevOps in comparison to independent developers. It's also important to note that ML Tool projects show more variance within their team sizes than their ML Applied counterparts, as illustrated within Figure 4, signaling that a lack of correlation between Team size and DevOps adoption is not due to limitations related to sample size, but rather the properties of ML Tool projects. Focusing on the different categories of DevOps tools, Age was also a key variable in determining whether an ML Tool project adopts Build, CI or Deployment tools, while surprisingly, Team size

was the key predictor of Code Analysis tools adoption. Finally, no predictors of Monitoring tools' adoption by ML Tools projects was found.

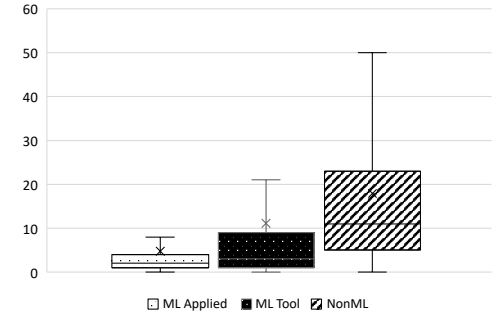

Fig. 4: Variance of Team size (Outliers removed)

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.106	Intercept of the model
Team Size	.003	.002	Size of the project's team
Age In Days	.006	.002	Age of the project
N_Forks	.010	.002	Number of Forks of projects
N_Pr_Open	.951	.000	Number of Pull Requests open of project

R Squared = .060 (Adjusted R Squared = .057)

TABLE III: ANCOVA analysis of DevOps adoption within Non-ML projects (Only statistically significant variables are shown)

Considering Non-ML projects, it's clear through Table III that they show similar results regarding the factors contributing to DevOps adoption to those of ML Tool projects. A Non-ML project's age, pull-request based development activity, popularity as measured by its number of forks, and its team size are significant contributing factors to its adoption of DevOps. Focusing on the different categories of DevOps tools, two or more of the aforementioned projects' characteristics were among the main predictors of the adoption of a specific DevOps tools category, indicating no major difference between the predictors of DevOps adoption in-general and the adoption of a specific category of DevOps tools by Non-ML projects.

A summary of our findings is illustrated in Table IV. We found that an ML Applied project's age and team size are more likely to affect its' DevOps adoption more so than that of a Tool or Non-ML project. Similar characteristics related to a project's size, popularity, as measured by its number of stars and forks, and its reliance on PR-based development, are the important factors that affect whether or not it adopts DevOps, regardless of whether or not it's an ML project. One important outlier is that an ML Tool projects' team size does not affect its DevOps adoption outcome. Based on observations by Karlaš et al. [29], Renggli et al. [16], Lwakatare et al. [11, 66], Amershi et al. [67], and Arpteg et al. [68], we attribute the lower adoption of DevOps by ML Applied projects to the differences between traditional software projects and ML projects, and a lack of DevOps tools that were specifically designed for small scale ML projects.

Category	Most important variables affecting DevOps adoption	Interpretation
ML Applied	Age In Days, Team Size, N_Pr _Merged, N_Pr_Core _Merged	An ML Applied projects' DevOps adoption is linked to its age, team size and reliance on PR-based development as measured through its number of pull requests merged
ML Tool	Age In Days, N_Stars, N_Forks, N_Pr_Open	An ML Tool projects' DevOps adoption is linked to its age, popularity as measured with its number of stars and forks, and its Number of PRs open
Non-ML	Team Size, Age In Days, N_Forks, N_Pr _Open	A Non-ML projects' DevOps adoption is linked to its team size, age, popularity as measured with its number of forks and reliance on PR-based development as measured through its number of pull requests open

TABLE IV: Summary of ANCOVA analyses results for DevOps Adoption

2) Most popular DevOps tools: In addition to exploring the adoption rates of DevOps tools and the factors affecting their adoption, we were interested in exploring which tools were currently popular across the different types of projects. tables V to IX illustrate the adoption rates for the DevOps tools that have at least 1% adoption rate by one or more categories of projects. We believe knowing which tools are popular for each project category can help guide future research regarding DevOps practices within them. For example, research on Code analysis within ML projects should focus on the Coverage and Pylint tools since they are the most popular Code analysis tools within them.

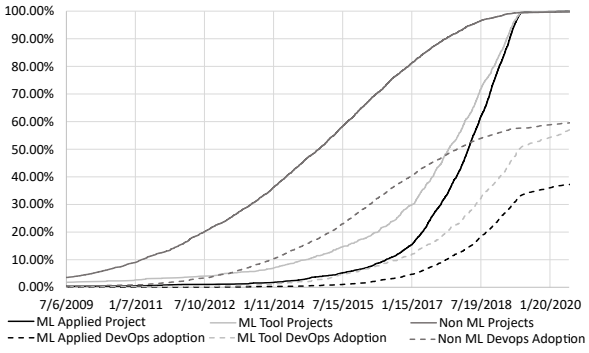

Fig. 5: Historical project amounts and their DevOps tools' adoption (normalized to percentages)

3) DevOps' historical Adoption Rates: We analyzed the historical adoption trends of DevOps tools to get a better understanding of the evolution of their adoption rates over time. The results of our analysis are illustrated in Figure 5.

Tool Name	Project Category	Adoption Percentage
setuptools	ML Applied	9.88%
	ML Tool	18.91%
	Non-ML	0.74%
Rake	ML Applied	0.41%
	ML Tool	1.34%
	Non-ML	1.72%
QMake	ML Applied	1.30%
	ML Tool	1.25%
	Non-ML	2.21%
Maven	ML Applied	2.78%
	ML Tool	5.38%
	Non-ML	1.67%
MakeFile	ML Applied	16.78%
	ML Tool	30.73%
	Non-ML	3.68%
JUnit	ML Applied	2.81%
	ML Tool	3.76%
	Non-ML	1.64%
Gradle	ML Applied	2.02%
	ML Tool	2.06%
	Non-ML	2.77%
Clang	ML Applied	2.06%
	ML Tool	6.63%
	Non-ML	0.65%
Ant	ML Applied	1.72%
	ML Tool	5.02%
	Non-ML	0.54%

TABLE V: Usage rates of Build Tools (Tools with 1% or more usage rates)

Tool Name	Project Category	Adoption Percentage
Pylint	ML Applied	2.02%
	ML Tool	4.75%
	Non-ML	0.17%
Flow	ML Applied	0.96%
	ML Tool	1.88%
	Non-ML	0.39%
Flake8	ML Applied	1.78%
	ML Tool	3.32%
	Non-ML	0.05
ESLint	ML Applied	1.58%
	ML Tool	1.34%
	Non-ML	2.55%
Coverage	ML Applied	3.50%
	ML Tool	7.89%
	Non-ML	0.29%
Codecov	ML Applied	2.64%
	ML Tool	5.38%
	Non-ML	0.25%
CodeClimate	ML Applied	0.48%
	ML Tool	1.08%
	Non-ML	0.27%
Clang	ML Applied	1.58%
	ML Tool	6.00%
	Non-ML	0.37

TABLE VI: Usage rates of Code Analysis Tools (Tools with 1% or more usage rates)

Tool Name	Project Category	Adoption Percentage
testthat	ML Applied	1.03%
	ML Tool	2.78%
	Non-ML	0.12%
Pytest	ML Applied	2.81%
	ML Tool	6.45%
	Non-ML	0.39%
JUnit	ML Applied	1.34%
	ML Tool	2.42%
	Non-ML	2.04%
Cassert	ML Applied	0.34%
	ML Tool	1.08%
	Non-ML	0.37%

TABLE VII: Usage rates of Test Tools (Tools with 1% or more usage rates)

Tool Name	Project Category	Adoption Percentage
Travis	ML Applied	17.94%
	ML Tool	33.24%
	Non-ML	10.30%
Jenkins	ML Applied	0.58%
	ML Tool	1.97%
	Non-ML	0.05%
AppVeyor	ML Applied	2.44%
	ML Tool	6.09%
	Non-ML	0.76%

TABLE VIII: Usage rates of Continuous Integration Tools (Tools with 1% or more usage rates)

Tool Name	Project Category	Adoption Percentage
Docker	ML Applied	13.17%
	ML Tool	15.59%
	Non-ML	1.67%
Chef	ML Applied	0.10%
	ML Tool	0.36%
	Non-ML	0.64%

TABLE IX: Usage rates of Deployment Automation Tools (Tools with 1% or more usage rates)

When analyzing the growth of Non-ML projects overall in comparison to that of Non-ML projects with one or more DevOps tools, it's clear that they both have similar trends over time, signaling a healthy adoption growth of DevOps among this type of projects.

Focusing on ML project types, both ML Tool projects' growth and ML Applied projects' seen a near exponential increase starting from 2017.The explosion in the projects' total amount can be attributed to the advances in ML fields and gains in their popularity. Focusing on the amount of ML Tool projects with DevOps tools, it shows similar growth trends as the total number of the ML Tool projects. This similarity in growth trends is also observed for Non-ML projects growth. However, while ML Applied projects have seen a similar in amount to ML Tool projects due to analogous reasons, their DevOps adoption growth has stalled in comparison. DevOps tools' in ML Applied projects had a slower and lower adoption rate overall in comparison to both Non-ML and ML Tool projects, and we were able to partially link them to the smaller team sizes of these projects in Section IV-A1 to their lower DevOps adoption. Overall, these results indicate that the current adoption rates are consistent with the historical rates

across project categories, and there are no abrupt changes of DevOps adoption.

Finding 1: *ML Tool projects and Non-ML projects have significantly higher current and historical DevOps tools' adoption rates than ML Applied projects. This adoption is most influenced by a project's age, team-size or both factors, depending on the project's category.*

B. DevOps Maintenance Efforts and Goals

Research Question 2: What are the maintenance efforts and goals associated with DevOps tools across the different categories of projects ?

Having determined the historical and current adoption rates, we wanted to investigate the differences in the effort that developers are putting into maintaining their DevOps configuration files and the correct functioning of DevOps tools within their repositories, and to explore the different goals of updates to DevOps configuration files.

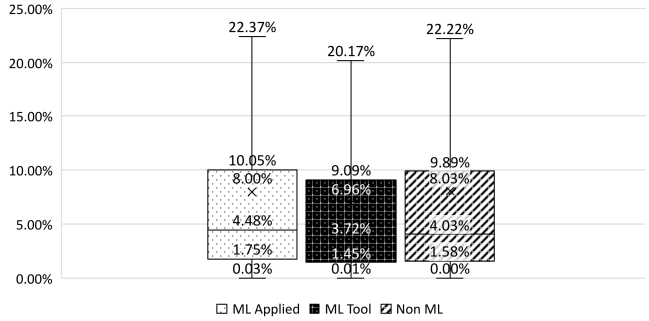

Fig. 6: Commit Ratios of DevOps configuration files

1) Ratio of DevOps configuration files' updates: We used the Commit Ratio metric to estimate the share of updates that affect DevOps tools out of all the updates that affect a repository. As illustrated by Figure 6, Tool projects tend to update their DevOps configuration files less overall, while Applied and Non-ML projects had higher and similar ratios of updates. The projects with the highest DevOps commits ratio are generally those with the majority of their updates affecting their Build, CI or Deployment automation files. One such example is the ML Applied project ROSETTE-API/ROSETTE-ELASTICSEARCH-PLUGIN, with 78.46% of its commits modifying its Maven and Travis file. The majority of these updates are comprised of version or dependency and configuration changes for the project overall or its docker image and the plug-ins it provides. Another example is the CLARITYCAFE/IVY repo, which has frequent commits which almost always change its Travis CI and Docker file. Upon closer inspection, we identified that this project's Docker and Travis files are mostly changed to fix CI and Deployment problems. These examples and our statistical findings stand in contrast with the concept of "write-once-and-forget-it" for DevOps configuration files and indicate that they evolve frequently for different aspects of software maintenance.

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.022	Intercept of the model
CI	.006	.007	Adoption of CI tool(s)
Build * CI * Analyzer	.007	.007	Adoption of Build, CI and CA tool(s)
Build * Deployment * Test	.024	.005	Adoption of Build, DA and Test tool(s)
Build * Analyzer * Test	.035	.004	Adoption of Build, Code Analysis and Test tool(s)
CI * Deployment * Analyzer	.045	.004	Adoption of CI, DA and CA tool(s)
CI * Deployment * Analyzer * Test	.045	.004	Adoption of CI, DA , CA and Test tool(s)

R Squared = .098 (Adjusted R Squared = .063)

TABLE X: ANCOVA analysis of Commit Ratio for Applied projects (Only statistically significant variables are shown)

To further investigate whether these project-specific trends are a widespread phenomenon, we performed, the ANCOVA analysis illustrated in Table X, we found that CI adoption, and the adoption of CI, Build and Test tools at the same time to be among the strongest factors leading to a higher commit ratio in Applied projects.

However, after performing the same analysis on the other two project categories, we found no statistically significant link between the adoption of specific DevOps tool categories and the commit ratio in ML Tool and Non-ML projects. This allows us to deduce that specific categories of DevOps tools, such as CI, Build and Testing tools in ML Applied projects need more frequent updates in comparison to other types of tools. Yet, ML Tool and Non-ML projects do not show this correlation. A summary of our ANCOVA analyses is found within Table XI

Category	Most important variables affecting DevOps churn	Interpretation
ML Applied	CI, Build * CI * Analyzer, Build * Deployment * Test, Build * Analyzer * Test, CI * Deployment * Analyzer, CI * Deployment * Analyzer * Test	An ML Applied projects' adoption of certain DevOps tool categories or a combination of these categories is linked to an increase in its DevOps configuration files commit-ratio
ML Tool	None	An ML Tool projects' DevOps configuration files commit-ratio is not linked to its adoption of a tool of a certain DevOps category.
Non-ML	None	A Non-ML projects' DevOps configuration files commit-ratio is not linked to its adoption of a tool of a certain DevOps category.

TABLE XI: Summary of ANCOVA analyses results for DevOps Commit-ratio

Finally, we were able verify the statistical dissimilarity between the different projects categories via the one-way

ANOVA test [69], a test developed to allow the comparison of the means of three or more different groups based on one property. The p-value obtained was 0.032 implying significant statistical difference between the three groups regarding their Commit Ratios.

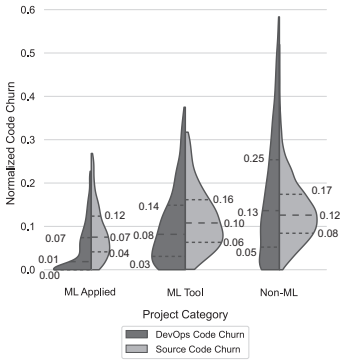

Fig. 7: Average Normalized Code Churn(Outliers removed with IQR [70])

2) DevOps Coding Efforts: To estimate the effort that developers put into DevOps configuration files in comparison to Source files between different commits, we used the Average Normalized Code Churn metric. As illustrated by the results in Figure 7, a comparatively higher relative churn of DevOps configuration files is noted in ML Tool projects in comparison to ML Applied projects. This is made clearer with the higher quartiles and median values of this metric for ML Tool DevOps churn in comparison to those of ML Applied projects. Non-ML projects had a bigger churn overall on both file-types, yet its DevOps churn shows a more even distribution across its value range, reflecting more diverse DevOps maintenance practices within these projects. With a more detailed analysis, we identified that both Source and DevOps churn values are generally high at the beginning of a project's history, matching the intuition regarding changes being done to a large number of files as the project's initial code and configuration are being defined across a variety of them. These rates tended to quickly drop in value during the following months. Regarding DevOps Churn specifically, it tended to increase across all project categories whenever a new DevOps tool was added to a project, and it can take several development periods to drop again. This signifies a possible adoption barrier due to the time and effort required to establish and configure correctly working DevOps tools in a project.

Focusing on some interesting cases, the ML Applied project with the highest Avg. Normalized DevOps Churn and Source code churn was the INDIX/WHATTHELANG project with the respective values of 1.0 and 0.43. This project provides a language prediction application usable via a CLI or an API. It employs Travis CI For continuous integration. Within this repository, 23 total commits over the period of one month were made. The only DevOps file within this project was a .travis.yml file, and it was updated more than once during that month, but not all of the source files were updated during this period following their creation.

The ML Tool project with the highest Avg. Normalized DevOps Churn and Source code churn was the YINCHUANDONG/SENTIMENT-ANALYSIS project with values of 1.0 and 0.2 respectively. It is a Deep Learning Workflow for Sentiment Analysis, and the only DevOps tool it uses is Docker for Deployment Automation. It also has a relatively low activity with 36 commits over the duration of one month, during which the Docker file was frequently updated. These two specific cases aside, ML projects of both types had DevOps churn values close to their Source churns. This implies that DevOps configuration files require development effort similar to that of Source files, along with the accompanying time and resource investments. Our intuition is confirmed within the ANCOVA analyses of DevOps code churn across the different project categories, which are illustrated and discussed in the following paragraphs.

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.022	Intercept of the model
Team Size	<.001	.021	Project's team size
Age In Days	<.001	.011	Project's age
N_Pr_Merged	.021	.005	Number of Pull requests merged
CI	.031	.004	Adoption of CI tool(s)
Deployment * Analyzer * Test	.032	.004	Adoption of DA, CA and Test tool(s)
Build * Deployment * Analyzer * Test	.037	.004	Adoption of Build, DA, CA and Test tool(s)
Analyzer * Test	.041	.004	Adoption of CA and Test tool(s)
N_Pr_Core_Merged	.048	.004	Number of Pull requests by core developers merged

R Squared = .213 (Adjusted R Squared = .182)

TABLE XII: ANCOVA analysis of DevOps Code Churn for Applied projects (Only statistically significant variables are shown)

Focusing on ML Applied projects, the results of which are illustrated in Table XII, we found that their adoption of a DevOps tool, or a combination of tools, such as Build or CI tools, is strongly correlated with an increase in their DevOps churn. Furthermore, the varying effect size values (represented by the Partial Eta Square) imply that different DevOps tools have different effort-requirements, with CI Tools being the ones that are most effort-intensive for ML Applied projects.

Moving on to ML Tool projects, the ANCOVA of which is illustrated in Table XIII, we also found that their adoption of one or more DevOps tools is correlated with an increase in their DevOps churn. In their case, the adoption of Build, Deployment Automation, Continuous Integration, and Code Analysis tools at the same time had the largest effect size, and thus the highest consequential increase in DevOps Churn. This implies that an ML Tool project's adoption of multiple DevOps tools categories at the same time is more likely to result in an increase of its DevOps configuration files churn

Source	Sig.	Partial Eta Squared	Details
Age In Day	<.001	.019	Age of the project
Intercept	<.001	.018	Intercept of the model
Build * CI * Deployment * Analyzer	<.001	.005	Adoption of Build, DA, CI, and CA tools
CI	.002	.004	Adoption of CI Tools
Build * CI * Analyzer	.002	.004	Adoption of Build, CI, and CA Tools
Deployment * Test	.006	.003	Adoption of DA and Test tools
N_issues_Open	.007	.003	Number of Issues open
CI * Analyzer	.020	.002	Adoption of CI and CA tools
Build * CI * Deployment	.021	.002	Adoption of Build, CI and DA tools
Build * Test	.042	.002	Adoption of Build and Test tools

R Squared = .106 (Adjusted R Squared = .090)

TABLE XIII: ANCOVA analysis of DevOps Code Churn for Tool projects (Only statistically significant variables are shown)

and this increase is likely to be more substantial than that resultant of the adoption of DevOps tools of one category.

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.082	Intercept of the model
Build * Analyzer * Test	.009	.011	Adoption of Build, CA and Test tools
Age In Days	.010	.011	Age of the project
Build	.031	.008	Adoption of Build Tools
N_Pr_Open	.040	.007	Number of Pull requests opened
Team Size	.043	.007	Size of the project's team
CI	.049	.006	Adoption of CI Tools

R Squared = .148 (Adjusted R Squared = .091)

TABLE XIV: ANCOVA analysis of DevOps Code Churn for Non-ML projects (Only statistically significant variables are shown)

Finally, focusing on Non-ML projects' ANCOVA, illustrated in Table XIV, we find similar results to those of ML Applied and ML Tool projects, establishing that the phenomena of increased DevOps configuration files Churn is true across project categories. It's interesting to note that the adoption of a different mix of DevOps categories, more specifically Build, Code Analysis and Test tools, which is different from that of ML Tool projects', is the variable with the largest effect size and hence the biggest effect on DevOps Churn of Non-ML projects. It is especially interesting that Test tools are within this group of category, as they do not rely on any specific configuration file. As mentioned in Section III-C3, we do not consider Test files as DevOps configuration files.,

The summary of our ANCOVA analyses in relation to DevOps churn is within Table XV. Notably, across all project categories, the number of issues does not seem to affect DevOps code churn, signaling a lack of correlation between the reporting of issues within a project and the churn of DevOps configuration files. Applying the one-way ANOVA

Category	Most important variables affecting DevOps churn	Interpretation
ML Applied	Team Size, Age In Days, N_Pr_Merged, CI, Deployment * Analyzer * Test, Build * Deployment * Analyzer * Test, Analyzer * Test, N_Pr_Core_Merged,	An ML Applied projects' Team Size, Age, reliance on PR-based development, and its adoption of certain DevOps tool categories or a combination of these categories are linked to an increase in its DevOps configuration files churn
ML Tool	Build * CI * Deployment * Analyzer, Build * CI * Analyzer , Deployment * Test, N_Issues_Open, CI * Analyzer, Build * CI * Deployment, Build * Test	An ML Tool projects' DevOps configuration files churn is not linked to its adoption of certain DevOps tool categories, and its number of issues open.
Non-ML	Build * Analyzer * Test, Age In Days, Build, N_Pr_Open, Team Size, CI	A Non-ML projects' DevOps configuration files churn is linked to its adoption of certain DevOps tool categories, its age, its reliance on PR-based development, and its team size.

TABLE XV: Summary of ANCOVA analyses results for DevOps Churn

test across the different categories, we obtain a p-value of $3.61\text{e}{-18}$ for the Source Code Churn and $6.49\text{e}{-18}$ for the DevOps Code Churn, implying significant statistical difference between the three groups of projects.

3) DevOps Change goals: After uncovering the efforts invested by developers in DevOps configuration files, we wanted to explore the goals developers were trying to achieve by changing one or multiple DevOps configuration files. To achieve this, we analyzed the different commits that affect DevOps configuration files and determined the commits' main goals, within 1437 ML projects and 1942 Non-ML projects which adopted Build and CI Tools, via a process detailed in Section III-C3b. The results are illustrated in Figure 8.

In a typical development cycle, bugs and problems may be detected directly by the developer through local unit testing, or be reported externally by either customers or testers. In a project that adopts CI tools, program bugs, test failures, DevOps tools' misconfigurations and other problems may be detected and reported by the CI system.

Source	Sig.	Partial Eta Squared	Details
CI * Deployment * Analyzer	.068❄	.004	Adoption of CI, DA and CA Tools
Intercept	.087❄	.004	Intercept of the model
Build * CI	.112❄	.003	Adoption of Build and CI Tools

R Squared = .059 (Adjusted R Squared = .007)

TABLE XVI: ANCOVA analysis of bug-fix commit goal for Applied projects (❄ marks statistically non-significant variables, table is shown for illustrative purposes)

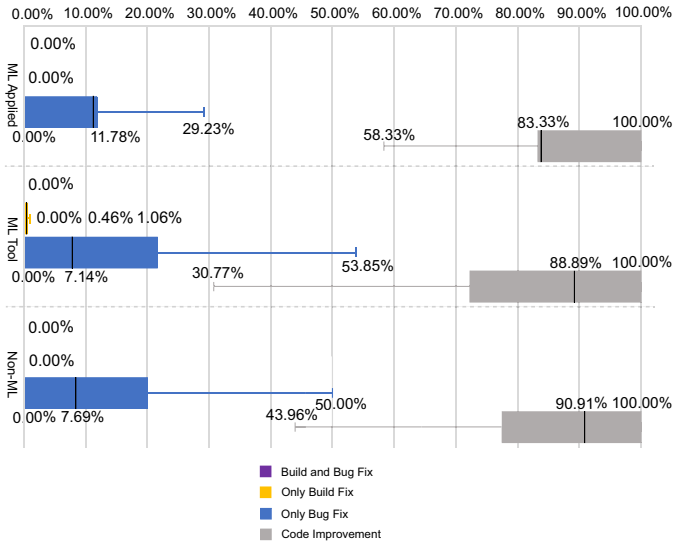

Fig. 8: Goals of DevOps-changing Commits (Outliers points hidden, 3 quartile-values shown if different)

For ML Applied projects, the lower percentages of bug-fixes shown in Figure 8 may imply that these projects are experiencing less build breakages and bugs. But in reality, the ANCOVA analysis for ML Applied projects in Table XVI indicates that there is no correlation between the adoption of Test and Code Analysis tools and a reduction in the percentages of these fixes. This indicates that ML Applied projects are not using these tools efficiently in order to remedy the bugs that may arise in their code. In addition, we did not find any correlation between team size or other covariates considered and bug-fixes. This implies that this misuse of Test and Code Analysis tools is present within the majority ML Applied projects, regardless of a project's properties.

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.060	Intercept of the model
Build * Analyzer * Test	.016	.012	Adoption of Build, CA and Test tools

R Squared = .136 (Adjusted R Squared = .062)

TABLE XVII: ANCOVA analysis of bug-fix commit goal for Tool projects (Only statistically significant variables are shown)

Moving on to ML Tool projects, a clear correlation is found between the adoption of Build, Code Analysis and Test tools within a project and bug-fixing commits being performed within it. Since the goals of Code Analysis and Test tools is to allow developers to find bugs and issues with their codebase, we interpret the increase of bug-fixing commits of ML Tool projects that adopted them as a sign of efficient use of these tools by these projects.

Concerning Non-ML projects, a correlation is found between the adoption of Build, CI, Code analysis, Test and Deployment Automation tools within a project and bug-fixing commits being performed within it. Similar to ML Tool projects, we interpret the increase of bug-fixing commits of Non-ML projects that adopted the different categories of DevOps tools, especially those designed to allow bug-detection as a sign of efficient use of these tools by these projects.

Source	Sig.	Partial Eta Squared	Details
Analyzer	<.001	.006	Adoption of CA tool(s)
Intercept	.005	.004	Intercept of the model
Deployment * Test	.020	.003	Adoption of DA tool(s)
Build * CI	.025	.003	Adoption of Build,CI tool(s)
Build * Analyzer	.033	.003	Adoption of Build,CA tool(s)
Build * Test	.047	.00	Adoption of Build, Test tool(s)

R Squared = .045 (Adjusted R Squared = .023)

TABLE XVIII: ANCOVA analysis of bug-fix commit goal for Non-ML projects (Only statistically significant variables are shown)

Across all projects categories, no correlation between Build fix percentage and Code Analysis tool adoption or any other variable was found within the ANCOVA analysis. This indicates that Build failures and the corresponding Build fixes are not affected by variability within projects or project categories, and that there is no evidence that the adoption of a specific tool or tool type such as code analyzers will influence build failures and subsequent build-fixes. A summary of the analyses we performed for DevOps change goals is within Table XIX.

Category	Most important variables affecting DevOps bug-fix commit	Interpretation
ML Applied	None	An ML Applied projects' adoption of certain DevOps tool categories or a combination of these categories is not linked to an increase in its Bug fixes
ML Tool	Build * Analyzer * Test	An ML Tool projects' commits which modify DevOps-files and fix bugs increase when Build, Code Analysis, and Test tools are adopted by them. This implies that these tools are being efficiently used to find and subsequently fix bugs.
Non-ML	Deployment * Test, Build * CI, Build * Analyzer, Build * Test	A Non-ML projects' commits which modify DevOps-files and fix bugs increase when combination of Build, Code Analysis, Test, Deployment, CI tools are adopted by them. This implies that the tools from these categories which facilitate bug-locating are being efficiently used to find and subsequently fix bugs.

TABLE XIX: Summary of ANCOVA analyses results for DevOps change goals

4) Interpretation of results: Using these findings, it's evident that developers working on ML Applied projects make numerous updates to their DevOps configuration files that are also smaller than those of ML Tools project. By comparison,

developers behind ML Tool projects overall did a smaller number of updates to their DevOps configuration files, that were larger in size. Non-ML projects had frequencies of DevOps-files updates similar to those of ML Applier projects, with a bigger variance in update-size in comparison to both ML categories. The frequency and size of updates, measured through the commit-ratio and DevOps code churn of ML Applied DevOps updates was linked to their adoption of certain DevOps tools categories, while no such correlations were found for ML Tool and Non-ML projects. The majority of DevOps updating commits of all projects categories had concerns that are not immediately related to the CI infrastructure which are in turn configured by DevOps configuration files. However, through the ANCOVA analyses we performed, we found that the adoption of Code Analysis, Test and other DevOps tools by ML Tool and Non-ML projects correlates with an increase in their bug-fixes. This signals that these tools are being efficiently used within these projects to detect bugs and the large effect size in the ANCOVA model signify this effect has important consequences on the number of bug-fixing commits. However, while adopting these tools is linked with larger and more frequent updates to DevOps configuration files within ML Applied projects, it is not linked with an increase in bug-fixing commits. This hints at a less efficient adoption of these tools which requires more frequent updates with more effort but no noticeable results on bug-fixes within ML Applied projects

Finding 2: *While ML Applied DevOps configuration files updates are more frequent, they are smaller in size than those of ML Tool DevOps configuration files, are less concerned with CI Build fixes, and imply that DevOps tools are being used less efficiently within these projects.*

C. DevOps Adoption Advantages

Research Question 3: What are the advantages of adopting DevOps tools across the different types of projects?

1) Commit Frequency: Among the goals of the adoption of DevOps tools and practices within software projects is to increase the rate at which developers share their code with other stakeholders within their teams, which in-turn is measured with the frequency of commits that developers make during a specific development period. As illustrated in Figure 9, the projects that adopted 1 or more specific types of DevOps tools had generally higher monthly commit frequencies. This was especially true for projects that adopted CI, Deployment Automation and Testing tools, where the increase in commit frequencies was significant across all types of projects. In addition, ML Tool projects tend to see more frequent commits than ML Applied projects, which in turn have more frequent commits than Non-ML projects.

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.043	Intercept of the model
DevOps	<.001	.013	DevOps tool(s) adoption
N_Pr_Rejected	<.001	.004	Number of Pull requests rejected
N_Pr_Core_Rejected	.002	.003	Number of Pull requests by core developers rejected
Age In Days	.003	.003	Project's age
Team Size	.004	.003	Project's team size
N_Stars	.013	.002	Number of stars
N_Pr_Core_Open	.036	.002	Number of Pull requests by core developers opened
N_issues_Open	.044	.001	Number of issues opened

R Squared = .185 (Adjusted R Squared = .182)

TABLE XX: ANCOVA analysis of Commit frequency for ML Applied projects (Only statistically significant variables are shown)

When statistically analyzing the Commit frequency through ANCOVA for ML Applied projects, as illustrated in Table XX, it's clear that DevOps tool adoption has a significant and important effect on the increase of monthly commit averages, especially since DevOps adoption is the variable with the largest effect size within the ANCOVA model.

Source	Sig.	Partial Eta Squared	Details
Team Size	<.001	.061	Size of the project's team
Intercept	<.001	.037	Intercept of the model
N_issues_Open	<.001	.011	Number of Pull requests opened
DevOps	.018	.005	Adoption of DevOps tool(s)
N_Forks	.029	.005	Number of Forks

R Squared = .198 (Adjusted R Squared = .189)

TABLE XXI: ANCOVA analysis of Commit frequency for ML Tool projects (Only statistically significant variables are shown)

For ML Tool projects, the ANCOVA analysis in table XXI, shows that DevOps tools adoption by these projects also has an important effect on the increase of their monthly commit averages. However, the size of a project's team and the number of open issues it has seem to have a larger effect than its DevOps adoption on its commit averages.

For Non-ML projects, the ANCOVA analysis in table XXII, shows that DevOps tools adoption by Non-ML projects positively affects its monthly commit averages. However, the size of a project's team and other variables related to its pull requests have a larger effect than its DevOps adoption on its commit averages.

The summary of our findings through the ANCOVA analysis linked to the average monthly Commits metric is illustrated in Table XXIII. Applying the one-way ANOVA test on this metric across the different categories, we obtain a p-value of s $7.29\mathrm{e}{-13}$, implying significant statistical difference regarding the average monthly commit frequency metric between the three groups of projects.

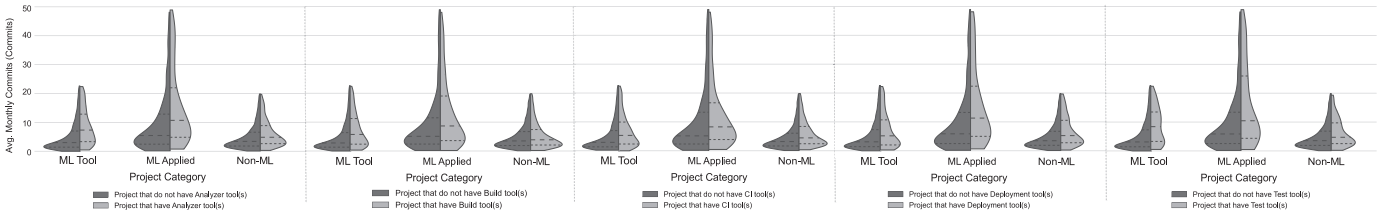

Fig. 9: Commit Frequency in correlation to Project Type and DevOps tool adoption (Outliers removed with IQR [70])

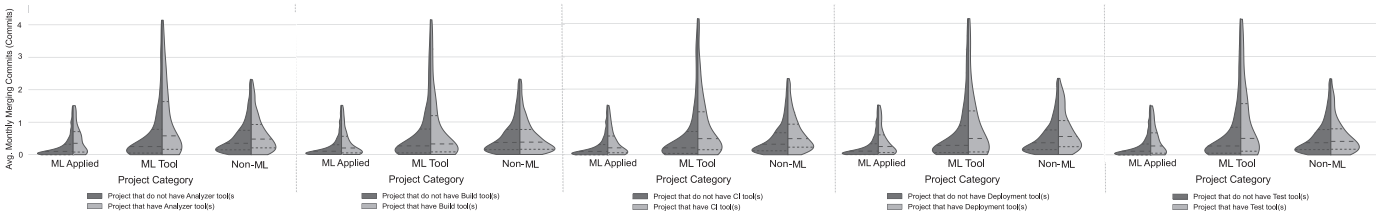

Fig. 10: Merging Commit Frequency in correlation to Project Type and DevOps tool adoption(Outliers removed with IQR [70])

Source	Sig.	Partial Eta Squared	Details
Team Size	<.001	.015	Project's team size
Intercept	<.001	.007	Intercept of the model
N_Pr_Core_Rejected	<.001	.005	Number of Pull requests by core developers rejected
N_Pr_Rejected	<.001	.003	Number of Pull requests rejected
N_Pr_Merged	.004	.002	Number of Pull requests merged
DevOps	.006	.002	Adoption of DevOps tool(s)
N_Pr_Core_Merged	.016	.002	Number of Pull requests by core developers merged

R Squared = .057 (Adjusted R Squared = .054)

TABLE XXII: ANCOVA analysis of Commit frequency for Non-ML projects (Only statistically significant variables are shown)

Category	Most important variables affecting Commit Frequency	Interpretation
ML Applied	DevOps, N_Pr_Rejected, N_Pr_Core_Rejected, Age In Days, Team Size, N_Stars, N_Pr_Core_Open, N_issues_Open	An ML Applied projects' adoption of DevOps has the largest effect on its monthly commits. Other factors such as its Number of rejects PRs and Team-size also affect this metric.
ML Tool	Team Size, N_issues_Open, DevOps, N_Forks	An ML Tool project's adoption of DevOps has an important effect on its monthly commits, however, other factors such as its Team-size have a larger effect on this metric.
Non-ML	Team Size, N_Pr_Core_Rejected, N_Pr_Rejected, N_Pr_Merged, DevOps, N_Pr_Core_Merged	A Non-ML project's adoption of DevOps has an important effect on its monthly commits, however, other factors such as its Team-size have a larger effect this metric.

TABLE XXIII: Summary of ANCOVA analysis results of Commit frequency

2) Merging Frequency: Increasing the rate at which developers merge their code with other code branches, thus increasing their code integration, is also a crucial goal of DevOps practices and tools. Merges are represented with merging commits in a Git repository, and the frequency of branch merges is measured with the frequency of merging commits that developers make within a specific development period. As represented in Figure 10, the projects that adopted a specific type or more of DevOps tools had generally higher monthly merge commit frequencies. This was especially true for projects that adopted Analyzer, CI, and Deployment Automation tools, where the increase in merge frequencies was significant across all types of projects. ML Tool projects tend to have more frequent merges than Applied projects, which in turn have more frequent commits than Non-ML projects.

Source	Sig.	Partial Eta Squared	Details
Intercept	<.001	.018	Intercept of the model
DevOps	<.001	.011	Adoption of DevOps tool(s)
Age In Days	.005	.003	Age of the project
N_Stars	.016	.002	Number of stars
N_Pr_Merged	.019	.002	Number of Pull Requests merged

R Squared = .258 (Adjusted R Squared = .255)

TABLE XXIV: ANCOVA analysis of Merge Commit frequency for Applied projects (Only statistically significant variables are shown)

To examine the relationship between DevOps tools' adoption and the frequency of merge commits, we built ANCOVA models for the different project categories. For ML Applied projects, this model is represent in Table XXIV. Similar to the results found within Section IV-C1 it's clear that adopting DevOps tools has a statistically-significant and important effect on the increase of monthly merge averages for ML Applied projects. DevOps adoption is the variable with the largest effect size within the ANCOVA model, indicating that

DevOps adoption has the highest positive influence on Merge commit rates within ML Applied projects.

Source	Sig.	Partial Eta Squared	Details
Team Size	<.001	.047	Size of the project's team
Intercept	<.001	.022	Intercept of the model
DevOps	.021	.005	Adoption of DevOps tool(s)

R Squared = .143 (Adjusted R Squared = .133)

TABLE XXV: ANCOVA analysis of Merge Commit frequency for Tool projects (Only statistically significant variables are shown)

Moving on to ML Tool projects, Table XXV shows that adopting DevOps tools also has a statistically-significant and important effect on the increase of monthly merge averages for ML Tool projects. However, it's important to note that an ML Tool project's team-size has a much larger effect on Merge commit rates within ML Tool projects.

Source	Sig.	Partial Eta Squared	Details
Team Size	<.001	.052	Size of the project's team
Age In Days	<.001	.006	Age of the project
Intercept	<.001	.004	Intercept of the model
N_Forks	<.001	.003	Number of forks
N_Stars	<.001	.003	Number of stars
N_Pr_Rejected	<.001	.003	Number of pull requests rejected
N_Pr_Core_Rejected	.003	.002	Number of pull requests by core developers rejected

R Squared = .086 (Adjusted R Squared = .083)

TABLE XXVI: ANCOVA analysis of Merge Commit frequency for Non-ML projects (Only statistically significant variables are shown)

Through the ANCOVA analysis on Non-ML projects, shown in Table XXVI, it seems that DevOps adoption has no effect on Non-ML merge rates. To better investigate this contradiction with existing findings regarding DevOps tool adoption on merge frequency [7], we performed a detailed analysis on the effects of the adoption of the different categories of DevOps tool categories, such as Build Tools, CI Tools, etc., on Non-ML merge frequency, which is illustrated within Table XXVII.

Source	Sig.	Partial Eta Squared	Details
Team Size	<.001	.042	Size of the project's team
CI * Analyzer * Test	<.001	.012	Adoption of CI, CA and Test tools
Build * CI * Deployment * Analyzer	<.001	.006	Adoption of Build, CI, DA and CA tools
Build * CI * Analyzer	<.001	.006	Adoption of Build, CI and CA Tools

R Squared = .144 (Adjusted R Squared = .135)

TABLE XXVII: Detailed ANCOVA analysis of Merge Commit frequency for Non-ML projects (Only statistically significant variables are shown)

In this model, the statistically significant variable with the second largest effect size is the adoption of CI tools, Analyzer tools and Test tools, implying that these specific tool categories are more likely to increase the merge frequency of Non-ML projects, versus the adoption of any combination of tools, which apparently has no effect on the number of monthly merges.

Category	Most important variables affecting Merging Commit Frequency	Interpretation
ML Applied	DevOps, Age In Days, N_Stars, N_Pr_Merged	An ML Applied projects' adoption of DevOps has the largest effect on its monthly merging commits. Other factors such as its number of stars and Number of Pull requests merged also affect this metric.
ML Tool	Team Size, DevOps	An ML Tool project's adoption of DevOps has an important effect on its monthly commits, however, Team-size has a larger effect this metric.
Non-ML	Team Size, CI * Analyzer * Test, Build * CI * Deployment * Analyzer, Build * CI * Analyzer	An Non-ML project's adoption of certain DevOps tool categories at the same time, such as adoption CI, Code Analysis and Test tools, has an important effect on its monthly merging commits. However, its Team-size has a larger effect this metric.

TABLE XXVIII: Summary of ANCOVA analyses results for Merging Commits frequency

The summary of our ANCOVA analyses in relation to the Average Monthly Merging Commits metric is detailed in Table XXVIII. Applying the one-way ANOVA test on the Average Monthly Merging Commits metric across the different project categories, we obtain a p-value of s $1.61\mathrm{e}{-13}$, implying that there is a significant statistical difference between the three groups of projects.

3) Issue Duration: Allowing the quick resolution of problems and shortening down-time are also some of the purported goals of adopting DevOps within a software project. To measure the effectiveness of teams at resolving such problems, we used the average issue duration metric to approximate the duration an issue takes to be resolved after it's opened within a specific project, in accordance to a project's category and its adoption of one or more types of DevOps tools. As illustrated in figure 11, adopting any type of DevOps tools corresponds to a quicker resolution of issues, especially the adoption of Analyzer, CI, Deployment Automation and Testing tools. Furthermore, ML Tool projects tend to have quicker resolution of issues than Applied projects, which in turn have a quicker resolution then Non-ML projects.

When analyzing the effect of the adoption of DevOps tools on issue durations of ML Applied projects, as illustrated in the ANCOVA analyses in Table XXIX, it's clear that it has a statistically significant and important effect on decreasing the

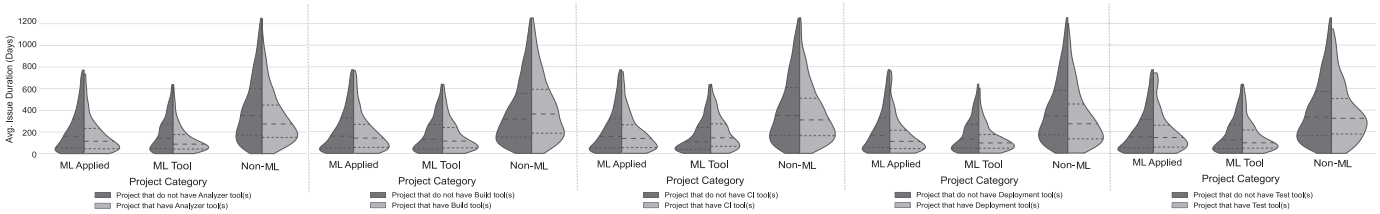

Fig. 11: Average Issue Duration in correlation to Project Type and DevOps tool adoption(Outliers removed with IQR [70])

Source	Sig.	Partial Eta Squared	Details
N_issues_Open	<.001	.033	Number of issues open
Intercept	<.001	.032	Intercept of the model
Age In Days	<.001	.021	Project's age
DevOps	<.001	.012	Adoption of DevOps tool(s)
N_Pr_Rejected	<.001	.007	Number of pull requests rejected
N_Pr_Core_Rejected	<.001	.006	Number of pull requests by core developers rejected
Team Size	.003	.003	Project's team size
N_Pr_Core_Open	.004	.003	Number of pull requests opened by core developers
N_Pr_Merged	.009	.003	Number of pull requests merged
N_Pr_Core_Merged	.033	.002	Number of pull requests by core developers merged

R Squared = .096 (Adjusted R Squared = .092)

TABLE XXIX: ANCOVA analysis of Average Issue duration for Applied projects (Only statistically significant variables are shown)

average issue durations across all project categories. However, the number of issues open and the age of the project seem to have larger effects than DevOps adoption.

Source	Sig.	Partial Eta Eta	Details
Intercept	<.001	.095	Intercept of the model
N_issues_Open	<.001	.022	Number of issues open
Age In Days	<.001	.017	Age of the project
N_Pr_Core_Open	.019	.006	Number of pull requests opened by core developers
DevOps	.045	.004	Adoption of DevOps tool(s)
N_Pr_Open	.046	.004	Number of pull requests open

R Squared = .094 (Adjusted R Squared = .083)

TABLE XXX: ANCOVA analysis of Average Issue duration for Tool projects (Only statistically significant variables are shown)

Moving on to the ANCOVA analysis regarding issue durations of ML Tool projects illustrated in Table XXX, it's clear that it has an important effect on decreasing the average issue durations. However, similar to ML Applied projects, the number of issues open and the age of the project seem to have larger effects than DevOps adoption.

By observing the ANCOVA analysis of the issue durations of Non-ML projects illustrated in Table XXXI, it's clear that it has an important effect on decreasing average issue durations. However, other factors, such as the number of pull requests

Source	Sig.	Partial Eta Squared	Details
N_Pr_Open	<.001	.101	Number of Pull requests opened
Age In Days	<.001	.076	Age of the project
N_issues_Open	<.001	.057	Number of Issues opened by core developers
Intercept	<.001	.049	Intercept of the model
N_Pr_Core_Open	<.001	.043	Number of Pull requests opened by core developers
Team Size	<.001	.027	Size of the project's team
N_Pr_Rejected	<.001	.012	Number of Pull requests rejected
DevOps	<.001	.008	Adoption of DevOps tool(s)
N_Stars	<.001	.007	Number of stars of project
N_Pr_Core_Rejected	<.001	.006	Number of Pull requests by core developers rejected
N_Forks	.001	.003	Number of forks of a project

R Squared = .327 (Adjusted R Squared = .325)

TABLE XXXI: ANCOVA analysis of Average Issue duration for Non-ML projects (Only statistically significant variables are shown)

open and the age of the project seem to have larger effects than DevOps adoption.

Category	Most important variables affecting Issue Duration	Interpretation
ML Applied	N_issues_Open, Age In Days, DevOps, N_Pr_Rejected,N_Pr _Core_Rejected, Team Size, N_Pr_Core_Open, N_Pr_Merged, N_Pr_Core_Merged	An ML Applied projects' DevOps adoption helps it reduce its issue duration, however, other factors such as its numbers of issues open and its age have a larger effect on these durations.
ML Tool	N_issues_Open, Age In Days, N_Pr_Core_Open, DevOps, N_Pr_Open	An ML Tool project' DevOps adoption helps it reduce its issue duration, however, other factors such as its numbers of issues open and number of PRs open have a larger effect on these durations.
Non-ML	N_Pr_Open, Age In Days, N_issues_Open, N_Pr_Core_Open, Team Size, N_Pr_Rejected, DevOps, N_Stars, N_Pr_Core_Rejected, N_Forks	A Non-ML project' DevOps adoption helps it reduce its issue duration, however, other factors such as its age and number of PRs open have a larger effect on these durations.

TABLE XXXII: Summary of ANCOVA analyses results for Average Issue Duration

A summary regarding the ANCOVA analyses linked to the average issue duration metric is illustrated in Table XXXII. Applying the one-way ANOVA test on the Average Monthly Merging Commit metric across the different categories, we obtain a p-value of s $1.02\mathrm{e}{-174}$, implying significant statistical difference between the three groups of projects.

4) Code Quality: In addition to positively influencing the code sharing rates and issue resolution durations, DevOps is also posed as a method of improving the quality of development processes of a project as well as its code base. To evaluate the validity of this claim, we used the state-of-the-art tool SonarQube [62] via the method described in Section III-C3c in order to evaluate the quality of the projects within our dataset. We were able to successfully generate code quality reports for 2566 ML Applied projects, 969 ML Tool projects and 3320 Non-ML projects, forming respectively 88.02%, 86.82% and 81.45% of the total number of projects from their categories. SonarQube was unable to process some projects due to problems such as software incompatibility, as the free version is not compatible with C and C++ projects, missing dependencies, internal memory management issues, among other reasons.

Source	Sig.	Partial Eta Squared	Details
Intercept	0.000	0.435	Intercept of the model
DevOps	<0.001	0.08	Adoption of DevOps
Age In Days	0.002	0.004	Age of a project
N_issues_Open	0.006	0.006	Number of Issues Open
Team Size	0.011	0.003	Age of a project

R Squared = .065 (Adjusted R Squared = .059)

TABLE XXXIII: ANCOVA analysis of Reliability for ML Applied projects

Source	Sig.	Partial Eta Squared	Details
Intercept	0.000	0.992	Intercept of the model
DevOps	<0.001	0.004	Adoption of DevOps

R Squared = .011 (Adjusted R Squared = .004)

TABLE XXXIV: ANCOVA analysis of Maintainability for ML Applied projects

Through the ANCOVA analyses within Table XXXIII, it's clear that an ML Applied project's reliability is correlated and most improved by its DevOps adoption. It's also interesting to note that a project's age, team size, and number of issues have a significant effect on improving a project's reliability. Longer-lived projects with larger teams, who are more capable at keeping track of bugs, are more likely to have better Reliability metrics. Focusing on ML Applied project's Maintainability, it's clear through Table XXXIV that DevOps adoption is the only project property that is statistically correlated to this quality metric. Overall, through these two analyses, it's clear that DevOps adoption is the number one factor influencing an ML Applied project's code quality.

Source	Sig.	Partial Eta Squared	Details
Intercept	0.000	0.479	Intercept of the model
DevOps	0.001	0.013	Adoption of DevOps

R Squared = .089 (Adjusted R Squared = .077)

TABLE XXXV: ANCOVA analysis of Reliability for ML Tool projects

Moving on to ML Tool projects, it's clear through Table XXXV that DevOps is the only statistically significant variable that affects these projects Reliability metric. However, no such correlation was found concerning the Maintainability metric, as no statistically significant variables were found within its ANCOVA analysis. This allows us to deduce that DevOps adoption only affects certain aspect of an ML Tool project's code quality, yet it is the only variable that seems to affect it, regardless of an ML Tool project's team size, age, etc.

Source	Sig.	Partial Eta Squared	Details
Intercept	0.000	0.476	Intercept of the model
DevOps	0.000	0.010	Adoption of DevOps
N_issues_Open	0.000	0.009	Number of Issues Open
Age In Days	0.000	0.005	Age of a project
NBForks	0.013	0.002	Number of Forks
Team size	0.005	0.002	Size of project's team

R Squared = .059 (Adjusted R Squared = .055)

TABLE XXXVI: ANCOVA analysis of Reliability for Non-ML projects

Concerning Non-ML projects, it's clear through Table XXXVI that DevOps is the single biggest contributor to a project's improved Reliability metric. In addition, a project's number of issues open, age, number of forks and Team size all correlate to this metric, signaling that multiple factors can influence a Non-ML project's reliability. However, it's also important to note that no statistically significant variables were found within the ANCOVA analyses of the Maintainability metric.

A summary regarding the ANCOVA analyses linked to the reliability and maintainability metrics is illustrated in Table XXXVII. Applying the one-way ANOVA test on these two metrics across the different categories, we obtain a p-value of $5.22\mathrm{e}{-6}$ for Reliability, and 0.98 for Maintainability. This is surprising as it implies significant statistical difference between the three groups of projects for the first metric, but similarity regarding the second metric, even though both are code quality metrics.

5) Interpretation of results: Using these five metrics and their associated statistical analyses, it's evident that employing DevOps tools of different categories has mostly correlated with an increase in the frequency of code commits, an increase in the merges across different branches, a reduced duration leading up to issue resolution, and an increase in code quality across the three different types of projects. These advantages are especially prevalent when using CI and

Category	Most important variables affecting Reliability	Most important variables affecting Maintainability	Interpretation
ML Applied	DevOps, Age In Days, N_issues _Open, Team Size	DevOps	An ML Applied project's DevOps adoption, age, and Number of issues open are the most important factors that affect its code quality
ML Tool	DevOps	None	An ML Tool project's DevOps adoption is the only statistically significant factors affecting its code quality
Non-ML	DevOps, N_issues _Open, Age In Days, NBForks, Team Size	None	A Non-ML project's DevOps adoption, Number of issues open, age, Number of forks and Team size are the most important factors that influence its code quality

TABLE XXXVII: Summary of ANCOVA analyses results for Reliability and Maintainability

Deployment automation tools across all categories of projects. Focusing more on ML Applied projects, it's evident that employing DevOps tools has an important and generally positive effect on the development activities, issue resolution, and code quality within these projects, thus signaling that while these projects may have a harder time employing DevOps tools, as per the findings in Section IV-B, they also have the most to gain from using DevOps tools within their code bases.

ML Tool and Non-ML projects that employ DevOps show mostly similar improvements in comparison to their non-DevOps counterparts, however, the improvements are not as drastic as those of the Applied ML projects.

Finding 3: *All categories of projects that employ DevOps show improvements in their development, code quality and issue resolution metrics in comparison to their non-DevOps counterparts, especially in the case of ML Applied projects, supporting the claim that DevOps tools can improve the development processes of most projects they are used in.*

V. Implications of the Proposed Study

In this section, we discuss the implications of our empirical analysis. The following is a list of actionable items we identified:

- Our analysis on DevOps adoption rates and trends, detailed in section IV-A, identified that ML Applied projects were slow in adopting DevOps. They also had a lower adoption across different DevOps tool categories such as Build, CI and Code Analyzer. While analyzing the exact reasons behind the barriers to adoption of DevOps tools is by ML projects is not within this work's scope, our results shed a light on the necessity for researchers to study the barriers to adopting DevOps in ML projects and identify possible improvement scopes. These may include ML DevOps task automation, DevOps tools for ML models evaluation and monitoring, etc. On the other hand, tool developers can employ program analysis [71] techniques to automatically generate ML DevOps configuration files which can lower the barriers of entry for data scientists who might be unfamiliar with DevOps concepts and practices.
- Our DevOps tool maintenance effort analysis, detailed within sections IV-B1 and IV-B2, reveals that even though ML Applied projects much less adoption of DevOps than the other two categories (ML Tools and Non-ML projects), their developers are changing DevOps configuration files more frequently. This highlights the necessity of working on support for automatic synchronization of DevOps configuration files. This may be provided via change recommendation tools [72], safe refactoring tools [73], and others. These tools can help reduce maintenance overhead, and can provide technical support to developers and data scientists who may not be very familiar with DevOps tools.
- Our analysis on events that trigger DevOps file changes, within section IV-B3, identified that bug-fixing commits within Tool project that alter DevOps configuration files were much more prevalent in comparison to ML Applied and Non-ML projects. This indicates that the software maintenance research community should invest more heavily in co-evolution analysis [74] of functional code and DevOps configuration files to facilitate early bug-detection. In turn, this will save both time and resources and allow teams to invest them in improving their software product's quality and reputation, rather than resolving problems within it.
- Our analysis on DevOps adoption advantages, within section IV-C, identified that for all project types, adopting DevOps has positive consequences on the code sharing and code integration speed and frequency and helped decrease the duration necessary for issue resolution and improve its quality. Even though using DevOps tools for all types of projects, including ML projects, introduced adoption and maintenance overhead, it appears that the benefits of DevOps outweigh the associated costs. Thus, data scientists and ML developers should adopt DevOps tools within their projects. Furthermore, we believe that adopting DevOps tools present these benefits for all ML projects, even for those with smaller teams. This is especially prevalent in the case of ML Applied projects, which had smaller team sizes overall but generally saw larger improvements resultant of DevOps adoption than ML Tool projects.
- Software engineering educators lack concrete ideas on

ML DevOps integration trends, benefits, and tools, preventing them from training students with ML DevOps skills that would allow them to build industry-ready ML-based systems. This study helps educators understand the current trends, benefits, and tools of ML DevOps in order to include up-to-date pedagogical material on ML DevOps.

VI. Threats to Validity

Our empirical analysis has some limitations that we would like to discuss:

Construct validity: We used the code churn and commit ratio metrics to estimate DevOps configuration files maintenance efforts. However, while these metrics may not reflect maintenance effort 100% correctly, they remain representative work items for maintaining source code and other files.

Internal validity: During DevOps tools detection, we used a file name patterns list which we manually constructed. To mitigate bias, one of the co-authors performed a manual checking of DevOps configuration files and file naming patterns in both ML projects and Non-ML projects. In addition, most tools have highly specific naming conventions, so the probability of false positives is minimal. Some tools, such as logging tools, may be hosted on third-party servers and do not need to have any configuration files within a repository, but they remain the minority among DevOps tools. Furthermore, DevOps tools that do not leave traces in files within the code repository, such as communication tools, can not be detected via our approach.

External validity: Our analysis is based on public repositories on GitHub. These results might differ for private GitHub repositories and closed repositories, including projects developed by companies. However, our project set does contain projects developed by companies, such as `tensorflow/tensor2tensor` which is backed by Google. We also estimate that at least 30% of ML Tool projects are backed by major organizations such as Microsoft and IBM. Furthermore, since we used popular organization and user-managed projects within our analysis, we expect many similarities of behavior.

VII. Conclusion

In this study, we conducted an empirical study on 4031 ML projects and a comparative set of 4076 Non-ML projects hosted in GitHub for ML DevOps adoption, maintenance effort and benefit analysis. Through our analysis, we found evidence of a lower adoption of DevOps tools within ML Applied projects, as well as different development practices and efforts in relation to these files that tended to be less efficient than those of ML Tool and Non-ML projects. In contrast, this type of projects has the most to gain from adopting these tools, and with similar advantages for both ML Tool and Non-ML projects. To the best of our knowledge, this is the first large scale empirical study on ML DevOps adoption, maintenance effort and benefit analysis. This exploratory work lays the foundation for future works, where we plan to investigate the roadblocks developers encounter when adopting different DevOps tools and the features they need to adopt to ease their adoption by ML developers. Our data and code are available at [48].

Acknowledgments

The UofM-Dearborn authors are supported in part by UofM-Dearborn Research Support and NSF Award NSF-2152819.

References

[1] S. Liu, S. Liu, W. Cai, S. Pujol, R. Kikinis, and D. Feng, "Early diagnosis of alzheimer's disease with deep learning," in *2014 IEEE 11th International Symposium on Biomedical Imaging (ISBI)*, April 2014, pp. 1015–1018.

[2] H. N. Mhaskar, S. V. Pereverzyev, and M. D. van der Walt, "A deep learning approach to diabetic blood glucose prediction," *CoRR*, vol. abs/1707.05828, 2017. [Online]. Available: http://arxiv.org/abs/1707.05828

[3] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.

[4] "The algorithm that beats your bank manager," https://www.forbes.com/sites/parmyolson/2011/03/15/the-algorithm-thatbeats-your-bank-manager/#15da2651ae99/, accessed: 2020-12-29.

[5] "Evans data corporation. 2019. global developer population and demographic study." https://evansdata.com/reports/viewRelease.php?reportID=9/, accessed: 2019-12-01.

[6] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of devops concepts and challenges," *ACM Comput. Surv.*, vol. 52, no. 6, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3359981

[7] "Seven devops tips for faster app development." https://resources.github.com/downloads/GitHub-Top-7-tips-for-faster-application-development-with-DevOps.pdf/, accessed: 2020-12-30.

[8] S. M. Brown Allana, Kersten Nigel, "2020 state of devops report," 2020.

[9] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym *et al.*, "Developments in mlflow: A system to accelerate the machine learning lifecycle," in *Proceedings of the fourth international workshop on data management for end-to-end machine learning*, 2020, pp. 1–4.

[10] "Amazon sagemaker," https://aws.amazon.com/sagemaker/, accessed: 2020-12-30.

[11] L. E. Lwakatare, I. Crnkovic, and J. Bosch, "Devops for ai – challenges in development of ai-enabled applications," in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, Sep 2020, p. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9238323/

[12] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral, "The

spack package manager: Bringing order to hpc software chaos," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2807591.2807623

[13] K. Hoste, J. Timmerman, A. Georges, and S. De Weirdt, "Easybuild: building software with ease," in *High Performance Computing, Networking, Storage and Analysis, Proceedings*. IEEE, 2012, pp. 572–582. [Online]. Available: http://dx.doi.org/10.1109/SC.Companion.2012.81

[14] "Docker," https://www.docker.com/, accessed: 2020-12-20.

[15] "Kubernetes," https://kubernetes.io/, accessed: 2020-12-20.

[16] C. Renggli, F. A. Hubis, B. Karlaš, K. Schawinski, W. Wu, and C. Zhang, "Ease.ml/ci and ease.ml/meter in action: Towards data management for statistical generalization," *Proc. VLDB Endow.*, vol. 12, no. 12, p. 1962–1965, Aug. 2019. [Online]. Available: https://doi.org/10.14778/3352063.3352110

[17] G. Fursin, H. Guillou, and N. Essayan, "Codereef: an open platform for portable mlops, reusable automation actions and reproducible benchmarking," 2020.

[18] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar, "Accelerating the machine learning lifecycle with mlflow," *IEEE Data Eng. Bull.*, vol. 41, pp. 39–45, 2018.

[19] L. E. Lwakatare, I. Crnkovic, and J. Bosch, "Devops for ai – challenges in development of ai-enabled applications," in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2020, pp. 1–6.

[20] D. Gonzalez, T. Zimmermann, and N. Nagappan, "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 (MSR)*, May 2020.

[21] D. Teixeira, R. Pereira, T. A. Henriques, M. Silva, and J. Faustino, "A systematic literature review on devops capabilities and areas:," *International Journal of Human Capital and Information Technology Professionals*, vol. 11, no. 2, p. 1–22, Apr 2020.

[22] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What is devops?: A systematic mapping study on definitions and practices," in *Proceedings of the Scientific Workshop Proceedings of XP2016*. ACM, May 2016, p. 1–11. [Online]. Available: https://dl.acm.org/doi/10.1145/2962695.2962707

[23] F. Erich, C. Amrit, and M. Daneva, "A mapping study on cooperation between information system development and operations," in *Product-Focused Software Process Improvement*, A. Jedlitschka, P. Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, and M. Raatikainen, Eds. Springer International Publishing, 2014, p. 277–280.

[24] W. P. Luz, G. Pinto, and R. Bonifácio, "Building a collaborative culture: A grounded theory of well succeeded devops adoption in practice," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3239235.3240299

[25] S. McIntosh, B. Adams, T. H. Nguyen, Y. Kamei, and A. E. Hassan, "An empirical study of build maintenance effort," ser. ICSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 141–150. [Online]. Available: https://doi.org/10.1145/1985793.1985813

[26] R. M. Shukla and J. Cartlidge, "Agileml: A machine learning project development pipeline incorporating active consumer engagement," in *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. Brisbane, Australia: IEEE, Dec 2021, p. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/9718470/

[27] I. Karamitsos, S. Albarhami, and C. Apostolopoulos, "Applying devops practices of continuous automation for machine learning," *Information*, vol. 11, no. 7, p. 363, Jul 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/7/363

[28] N. Nahar, S. Zhou, G. Lewis, and C. Kästner, "Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process," no. arXiv:2110.10234, Feb 2022, arXiv:2110.10234 [cs]. [Online]. Available: http://arxiv.org/abs/2110.10234

[29] B. Karlaš, M. Interlandi, C. Renggli, W. Wu, C. Zhang, D. Mukunthu Iyappan Babu, J. Edwards, C. Lauren, A. Xu, and M. Weimer, "Building continuous integration services for machine learning," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2407–2415. [Online]. Available: https://doi.org/10.1145/3394486.3403290

[30] PyGithub, "Pygithub/pygithub." [Online]. Available: https://github.com/PyGithub/PyGithub

[31] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, Aug 2015, p. 805–816. [Online]. Available: https://dl.acm.org/doi/10.1145/2786805.2786850

[32] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu, "The impact of continuous integration on other software development practices: A large-scale empirical study," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Oct 2017, p. 60–71.

[33] J. H. Bernardo, D. A. da Costa, and U. Kulesza, "Studying the impact of adopting continuous integration

on the delivery time of pull requests," in *Proceedings of the 15th International Conference on Mining Software Repositories*. Gothenburg Sweden: ACM, May 2018, p. 131–141. [Online]. Available: https://dl.acm.org/doi/10.1145/3196398.3196421

[34] H. J. Keselman, C. J. Huberty, L. M. Lix, S. Olejnik, R. A. Cribbie, B. Donahue, R. K. Kowalchuk, L. L. Lowman, M. D. Petoskey, J. C. Keselman *et al.*, "Statistical practices of educational researchers: An analysis of their anova, manova, and ancova analyses," *Review of educational research*, vol. 68, no. 3, pp. 350–386, 1998.

[35] A. Rutherford, *ANOVA and ANCOVA: A GLM Approach*. Wiley, 2011. [Online]. Available: https://books.google.com/books?id=c5aOZEniMqwC

[36] S. Rafi, W. Yu, and M. A. Akbar, "Rmdevops: A road map for improvement in devops activities in context of software organizations," *Proceedings of the Evaluation and Assessment in Software Engineering*, 2020.

[37] B. B. N. França, H. Jeronimo, and G. Travassos, "Characterizing devops by hearing multiple voices," in *SBES '16*, 2016.

[38] L. E. Lwakatare, T. Kilamo, T. Karvonen, T. Sauvola, V. Heikkilä, J. Itkonen, P. Kuvaja, T. Mikkonen, M. Oivo, and C. Lassenius, "Devops in practice: A multiple case study of five companies," *Information and Software Technology*, vol. 114, p. 217–230, Oct 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584917302793

[39] G. B. Ghantous and A. Gill, "Devops: Concepts, practices, tools, benefits and challenges," *Pacific-Asia Conference On Information Systems PACIS 2017 Proceedings*, 2017. [Online]. Available: https://aisel.aisnet.org/pacis2017/96

[40] I. Bucena and M. Kirikova, "Simplifying the devops adoption process," in *BIR Workshops*, 2017.

[41] A. H. L., N. J. S., V. J., and V. K., "A basic introduction to devops tools," 2015.

[42] L. Yin and V. Filkov, "Team discussions and dynamics during devops tool adoptions in oss projects," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, pp. 697–708.

[43] S. Mcintosh, B. Adams, and A. E. Hassan, "The evolution of java build systems," *Empirical Softw. Engg.*, vol. 17, no. 4–5, p. 578–608, Aug. 2012. [Online]. Available: https://doi.org/10.1007/s10664-011-9169-5

[44] Y. Jiang and B. Adams, "Co-evolution of infrastructure and source code: An empirical study," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. IEEE Press, 2015, p. 45–55.

[45] Github, "github/linguist." [Online]. Available: https://github.com/github/linguist

[46] J. Katz, "Libraries.io Open Source Repository and Dependency Metadata," Jan. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3626071

[47] J. Zhu, M. Zhou, and A. Mockus, "Patterns of folder use and project popularity: a case study of github repositories," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*. ACM Press, 2014, p. 1–4. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2652524.2652564

[48] "Replication package," https://figshare.com/s/0c4b685d4ab04f7f15af.

[49] Gitpython-Developers, "gitpython-developers/gitpython." [Online]. Available: https://github.com/gitpython-developers/GitPython

[50] G. Fan, C. Wang, R. Wu, X. Xiao, Q. Shi, and C. Zhang, "Escaping dependency hell: finding build dependency errors with the unified dependency graph," *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020.

[51] G. Robles, J. M. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar, "Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack," in *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. ACM Press, 2014, p. 222–231. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2597073.2597107

[52] D. Spadini, M. Aniche, and A. Bacchelli, "PyDriller: Python framework for mining software repositories," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*. New York, New York, USA: ACM Press, 2018, pp. 908–911. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3236024.3264598

[53] [Online]. Available: https://graphql.github.com/

[54] Z. Lubsen, A. Zaidman, and M. Pinzger, "Using association rules to study the co-evolution of production test code," in *2009 6th IEEE International Working Conference on Mining Software Repositories*, 2009, pp. 151–154.

[55] A. Zaidman, A. Zaidman, B. Van Rompaey, B. Van Rompaey, A. van Deursen, A. van Deursen, S. Demeyer, and S. Demeyer, "Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining," *Empirical software engineering : an international journal*, vol. 16, no. 3, pp. 325–364, 2011.

[56] H. Wu, L. Shi, C. Chen, Q. Wang, and B. Boehm, "Maintenance effort estimation for open source software: A systematic literature review," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Oct 2016, p. 32–43. [Online]. Available: http://ieeexplore.ieee.org/document/7816452/

[57] D. H. Martin and J. R. Cordy, "On the maintenance complexity of makefiles," in *Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics - WETSoM '16*. ACM Press, 2016, p. 50–56. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2897695.2897703

[58] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: Bugs and bug-fix commits," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 97–106. [Online]. Available: https://doi.org/10.1145/1882291.1882308

[59] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu, "On the naturalness of buggy code," *Proceedings of the 38th International Conference on Software Engineering*, 2016.

[60] F. Hassan and X. Wang, "Hirebuild: An automatic approach to history-driven repair of build scripts," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1078–1089. [Online]. Available: https://doi.org/10.1145/3180155.3180181

[61] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge, "Programmers' build errors: a case study (at google)," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, May 2014, p. 724–734. [Online]. Available: https://dl.acm.org/doi/10.1145/2568225.2568255

[62] "Code quality and code security | sonarqube." [Online]. Available: https://www.sonarqube.org/

[63] A. Rahman, A. Agrawal, R. Krishna, and A. Sobran, "Characterizing the influence of continuous integration. empirical results from 250+ open source and proprietary projects," *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics*, p. 8–14, Nov 2018, arXiv: 1711.03933.

[64] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 426–437. [Online]. Available: https://doi.org/10.1145/2970276.2970358

[65] C. Vassallo, F. Palomba, A. Bacchelli, and H. C. Gall, "Continuous code quality: are we (really) doing that?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. Montpellier France: ACM, Sep 2018, p. 790–795. [Online]. Available: https://dl.acm.org/doi/10.1145/3238147.3240729

[66] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic, *A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation*, ser. Lecture Notes in Business Information Processing. Springer International Publishing, 2019, vol. 355, p. 227–243. [Online]. Available: http://link.springer.com/10.1007/978-3-030-19034-7_14

[67] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "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, May 2019, p. 291–300. [Online]. Available: https://ieeexplore.ieee.org/document/8804457/

[68] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, Aug 2018, p. 50–59. [Online]. Available: https://ieeexplore.ieee.org/document/8498185/

[69] H.-Y. Kim, "Analysis of variance (anova) comparing means of more than two groups," *Restorative Dentistry& Endodontics*, vol. 39, no. 1, p. 74, 2014.

[70] P. J. Rousseeuw and M. Hubert, "Robust statistics for outlier detection," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 1, p. 73–79, 2011.

[71] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *Acm sigplan notices*, vol. 40, no. 6, pp. 190–200, 2005.

[72] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004.

[73] H. K. Wright, D. Jasper, M. Klimek, C. Carruth, and Z. Wan, "Large-scale automated refactoring using clangmr," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 548–551.

[74] Y. Jiang and B. Adams, "Co-evolution of infrastructure and source code-an empirical study," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 45–55.