On the Impact of Multi-language Development in Machine Learning Frameworks

Manel Grichi
GIGL department
Polytechnique Montreal
Quebec, Canada
manel.grichi@polymtl.ca

Ellis E. Eghan
GIGL department
Polytechnique Montreal
Quebec, Canada
ellis.eghan@polymtl.ca

Bram Adams
GIGL department
Polytechnique Montreal
Quebec, Canada
bram.adams@polymtl.ca

***Abstract*—The role of machine learning frameworks in software applications has exploded in recent years. Similar to non-machine learning frameworks, those frameworks need to evolve to incorporate new features, optimizations, etc., yet their evolution is impacted by the interdisciplinary development teams needed to develop them: scientists and developers. One concrete way in which this shows is through the use of multiple programming languages in their code base, enabling the scientists to write optimized low-level code while developers can integrate the latter into a robust framework. Since multi-language code bases have been shown to impact the development process, this paper empirically compares ten large open-source multi-language machine learning frameworks and ten large open-source multi-language traditional systems in terms of the volume of pull requests, their acceptance ratio *i.e.,* the percentage of accepted pull requests among all the received pull requests, review process duration *i.e.,* period taken to accept or reject a pull request, and bug-proneness. We find that multi-language pull request contributions present a challenge for both machine learning and traditional systems. Our main findings show that in both machine learning and traditional systems, multi-language pull requests are likely to be less accepted than mono-language pull requests; it also takes longer for both multi- and mono-language pull requests to be rejected than accepted. Machine learning frameworks take longer to accept/reject a multi-language pull request than traditional systems. Finally, we find that mono-language pull requests in machine learning frameworks are more bug-prone than traditional systems.**

***Keywords*-Machine learning, Framework, Open Source, Software engineering, Multi-language, Traditional systems.**

I. Introduction

Modern software is no longer developed in a single programming language. Instead, programmers tend to exploit the strengths of different programming languages, thus developing multi-language systems [1], [2].

In the context of this study, we consider a multi-language system as any software system where its own codebase contains source code executed at run-time that is developed in at least two programming languages. This definition excludes Makefiles, shell scripts, etc. Furthermore, it excludes cases where a third party library is developed in a different language, since the code of that library does not belong to the system's own codebase.

Programmers also opt for this practice to keep using legacy implementations of their existing libraries and systems, while still benefiting from code reuse of modern software components [3], [4]. Typical combinations of programming languages are Python C extension [5] and Java Native Interface [4]. For example, the Python C extension allows a Python program to interact with native C/C++ modules through a native extension module (FFI) [6]. The extension module provides a set of native functions (written in C) that can be imported and used within Python code.

Given the complex nature of machine learning (ML) and artificial intelligence (AI) frameworks, the AI community has been taking advantage of multiple languages in a single machine learning framework. In particular, while Python has evolved as the most commonly used language for developing machine learning frameworks due to its large range of powerful features [7], there are some demerits of using Python alone *i.e.,* it lacks computational performance needed for high-frequency real-time predictions [8], it takes significant CPU time for interpretation, etc. Hence, the Python C extension is often used as a solution to interface with highly performant C code for frequently executed low-level algorithms, as required, for example, by the gaming industry [9], multi-agents [10], and so on.

Despite the benefits of using multi-languages in developing machine learning frameworks, there are also several challenges associated with it. The major issue concerning this paradigm is that multi-language programs do not necessary obey any of the semantics of the combined languages [10] and it is the developer's responsibility to deal with the different programming calling conventions to avoid introduction of diverse issues that can harm the software.

Such concerns are not necessarily new, since multi-language development has been used for a long time for developing more traditional systems as discussed by several works in the literature [11]–[13]. However, in the case of ML frameworks, the traditional issues of multi-language development are further corroborated by the inherent complexity of ML frameworks. For example, they implement highly specialized mathematical operations that are challenging to test and debug, and even require interdisciplinary collaboration between scientists and developers [14], [15].

Hence, how "open" is open source ML framework development? Are these frameworks following the multi-language

trend? Does the practice of multi-language development increase the difficulty of dealing with machine learning frameworks? To the best of our knowledge, in this paper we present the first study that investigates the prevalence and the impact of multi-language development on the development of machine learning frameworks in terms of their ability to solicit high-quality open source contributions. More specifically, we study in this paper the impact of multi-language development in terms of the volume, acceptance ratio (percentage of accepted pull requests out of the total), review process duration (the period spent by the developers to accept or reject a pull request), and bug-proneness of pull requests (PR).

We empirically analyze the ten largest open source multi-language machine learning frameworks (Cat-I) and the ten largest open source traditional systems (Cat-II). In addition, we consider a set of seven mono-language open source machine learning frameworks (Cat-III) that serves as a control group for the comparison between Cat-I and Cat-II. We address the following research questions:

- **RQ1.** What is the prevalence of multi-language development in machine learning frameworks?
- **RQ2.** What is the impact of multi-language development on pull request acceptance ratio in machine learning frameworks?
- **RQ3.** What is the impact of multi-language development on the period taken to accept pull requests in machine learning frameworks?
- **RQ4.** Are multi-language pull requests more bug-prone than mono-language pull requests in machine learning frameworks?

Our main results show that:

- ML frameworks and traditional systems are comparable in terms of the proportion of multi-language development and multi-language pull requests.
- Multi-language PRs in ML frameworks have a lower acceptance ratio than mono-language PRs.
- Multi-language PRs in ML frameworks take longer to be accepted than mono-language PRs, and ML frameworks take longer to accept/reject a multi-PR than traditional systems.
- Mono-language PRs of ML frameworks are more bug-prone than in traditional systems.

The remainder of this paper is organized as follows. Section II describes the methodology and the design of our study. Section III presents our findings. Section IV discusses the lessons learned and the implications of the findings. We summarize the threats to the validity of our conducted study in Section V and present related work in Section VI. Finally, Section VII concludes the paper and outlines avenues for future work.

II. Methodology

This section discusses our methodology to empirically analyze the impact of multi-language development on open-source machine learning frameworks.

A. Project selection and cloning

In this empirical study, we analyze a total of 27 open source projects hosted on GitHub. Our selected projects include the ten largest multi-language machine learning frameworks and seven mono-language machine learning frameworks identified by Braiek *et al.* 's study [7], as well as the ten largest multi-language traditional systems from Grichi *et al.* 's study [4]. The seven mono-language machine learning frameworks serve to control for bias and any confounding factors in our comparison of multi-language machine learning and multi-language traditional systems.

We clone each project from GitHub and extract the following information: total number of lines of code, all pull requests (PR), all commits, and the percentage of programming languages used. All the used scripts in this study are available online[1] for replication.

B. Project categorisation

For clarity, the selected projects are grouped into 3 categories: Cat-I constitutes the ten largest multi-language open source **machine learning** frameworks, Cat-II constitutes the ten largest multi-language **traditional systems**, and Cat-III constitutes the seven **mono-language** open source machine learning frameworks. Table I gives an overview of the three categories.

C. Preprocessing and filtering

Accepted and Rejected Pull requests — We categorize pull requests as either accepted or rejected based on the pull request status (*i.e.,* Merged, Closed). Pull requests with both closed and merged status are classified as accepted pull requests. We identify a rejected pull request as being closed but not merged. We do not consider open pull requests in this study as they are still under review. For the rest of paper, we call accepted pull requests "accept-PR" and rejected pull requests "reject-PR".

Multi- and Mono-language Pull requests — We identify the set of changed files for each commit linked to a pull request, as well as the programming language(s) used in each file. A pull request that has at least one multi-language commit is considered as a multi-language pull request. Conversely, a pull request with no multi-language commit is considered a mono-language pull request. A commit is tagged as multi-language if it has files written in more than one programming language, while it is tagged as mono-language when it has files written in only one programming language. For example, a pull request *P1* that contains two commits *C1 (file1.java, file2.c)* and *C2 (file3.c, file4.c)* is considered as a multi-language pull request. A pull request *P2* that contains two commits *C3 (file5.java, file6.java)* and *C4 (file7.c, file8.c)* is considered as a mono-language pull request.

An alternative definition of multi-language PR would have been "any PR for which the union of changed files covers

[1] https://github.com/ICSME-2020/ICSME20.git, https://doi.org/10.6084/m9.figshare.12812159.v1

at least two programming languages". However, according to the rules for inter-language dependencies (*e.g.,* , between Java and C), the (multi-language) dependant files should change together in order to compile and run, hence our commit-level definition is more realistic. For the rest of the paper, we call multi-language pull requests "multi-PR" and mono-language pull requests "mono-PR".

D. Pull request analysis

Pull request acceptance period — We calculate the period spent (in hours) for a pull request to be accepted or rejected based on the difference between the pull request submission date and when the pull request was closed.

Bug-inducing pull requests — We collect the log messages of all pull requests and their contained commits. We split each message into words, the search for keywords and references to bug reports. Examples of the common keywords used were: '"fix", "correct", "bug", "error", "issue", "mistake", "blunder", "incorrect", "fault", "defect", "flaw", "bugfix", "bugfix:"'. Each identified buggy pull request was checked automatically to identify any reference to a #Bug_number. As soon as a pull request containing one of the keywords was also found to refer to a bug report, it was considered to be a bug fix.

Then, once we identify the pull requests that contain a fix to a bug, we apply the SZZ algorithm [16] to determine the initial bug-introducing pull request. The SZZ algorithm uses git-blame on the revision history to identify commits that are likely to have introduced bugs to determine first what changed in the bug-fix, then to locate the origins of the deleted or modified source code change that introduced this bug [17]. Finally, all identified bug-introducing pull requests are tagged automatically and assigned to the right group (multi-PR or mono-PR). We statistically compare the bug-introducing multi-PR to the bug-introducing mono-PR.

E. Statistical tests

We use the non-parametric Mann-Whitney U statistical test [18] with a 95% confidence level (*i.e.,* α = 0.05). We consider Bonferroni correction [19] to control the family-wise error rate when we perform more than one comparison on the same data. According to this correction, we divide the confidence level α by the number of tests. We also compute the Cliff's Delta effect size [20] if a significant difference is obtained. An effect size, r, is classified as "negligible" if r<0.2, as "medium" if 0.2<r<0.5, and as "large" if 0.5<r<0.8. The larger the effect size the stronger the relationship between the two variables.

Regarding the statistical tests of proportional metrics (*i.e.,* acceptance ratio and bug-proneness), we used Pearson's Chi-Square test of independence [21] to test if there is a difference in acceptance ratio or bug-proneness between (1) multi and mono changes in Cat-I, (2) multi and mono changes in Cat-II, (3) multi changes in Cat-I and Cat-II projects, or (4) mono changes in Cat-I, Cat-II, and Cat-III. We considered as well a 95% confidence level (*i.e.,* α = 0.05).

TABLE I: Selected case study projects, grouped by category and, per category, ordered from largest to smallest in terms of total number of lines code.

	Project	#Code lines	#Commits	#Pull Requests	Versions
Cat-I	Spacy	6,02M	10382	1057	v2.2.4
	Tensorflow	2,49M	61240	12393	v2.1.1
	Pytorch	817K	19559	16999	v1.5.0
	Incubator-mxnet	414K	9869	7965	v1.6.0
	CNTK	327K	16108	547	v2.7
	Paddle	290K	24724	10858	v1.8.1
	Caffe2	275K	3680	1260	v0.8.1
	Theano	155K	28081	4094	v1.0.4
	Scikit-learn	153K	24299	7971	v0.23.1
	Caffe	76,3K	4154	2204	v1.0
Cat-II	NativeScript	1,93M	16150	2435	v6.5.2
	Openj9	1,47M	8239	4519	v0.20.0
	Godot	1,15M	19898	12057	v3.2.1
	Libgdx	830K	13580	2779	v1.9.10
	RethinkDB	486K	33402	363	v2.4.0
	Mapbox GL	399K	14976	7707	v1.11.0
	React-native	395K	18038	8623	v0.63.0
	Play!framework	394K	14059	1943	v2.8.2
	RocksDB	346K	8341	4012	v6.11
	VLC	196,1K	11866	1884	v4.0.0
Cat-III	Nltk	228K	13884	1128	v3.5
	Keras	50,8K	5342	3918	v2.3.0
	Neon	49,4K	1118	88	v0.4.0
	Torch7	29K	1337	510	v1.5.0
	Pattern	23,6K	1433	118	v2.6
	Tflearn	10,4K	605	247	v0.3.2
	Sonnet	7,42K	764	39	v2.0.0

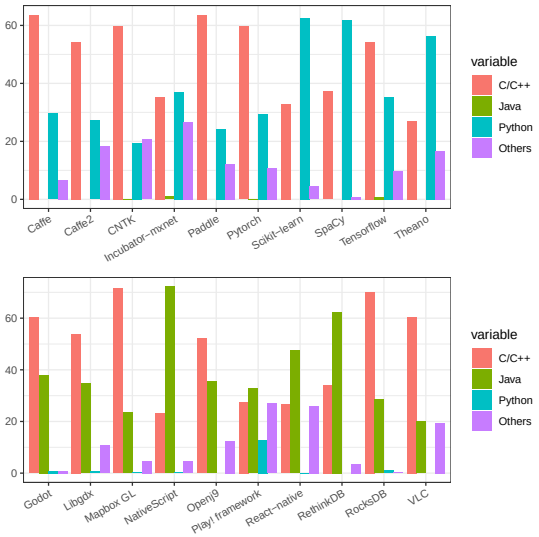

Fig. 1: Percentage of the programming languages used.

III. Results

The following section presents our results and summarizes them per research question.

RQ1. *What is the prevalence of multi-language development in machine learning frameworks?*

Motivation: In recent years, multi-language development has been adopted massively in the domain of AI-based software systems. In particular, many (open-source)

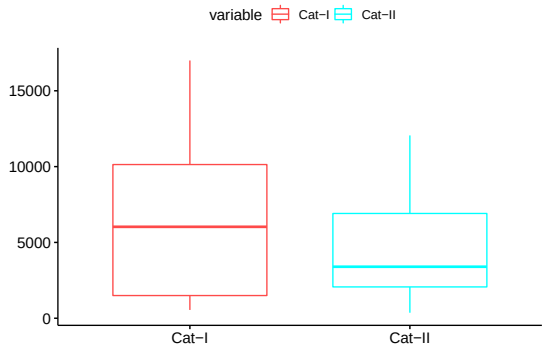

Fig. 2: Total number of pull requests.

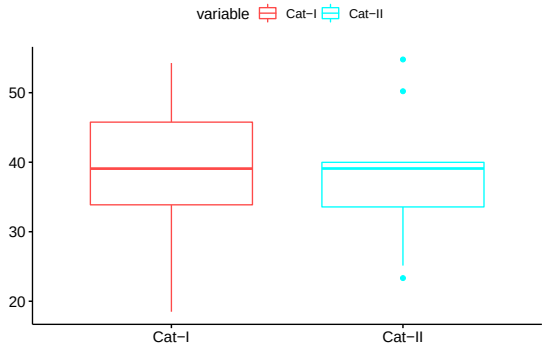

Fig. 3: Distribution of the percentage of multi-language pull requests

frameworks for machine learning have been engineered using multi-language practices, typically to integrate a low-level language for efficient computations with a high-level language for building robust software frameworks [22]. This research question aims to identify the presence/absence of the practice of multi-language development in ML frameworks. We investigate the different languages used and the prevalence of multi-language contributions (pull requests) in machine learning frameworks, then compare the results with those of multi-language traditional software systems.

Results: Both Cat-I and Cat-II projects have similar percentages of main programming languages involved, *i.e.,* the two sets of projects are comparable. Figure 1 shows that the distribution of programming languages involved in the studied Cat-I (Figure 1a) and Cat-II (Figure 1b) projects are similar. We find that regarding Cat-I, the main languages are Python and C, while Java and C/C++ are the main languages for Cat-II. Other languages, especially Objective-C and Perl are the least common for both categories.

Cat-I and Cat-II projects are also comparable in terms of the total number of PRs and the number of multi-language PRs.

Figure 2 presents the total number of pull requests (PR) in the studied Cat-I and Cat-II projects, respectively. The number of pull requests of Cat-I is not significantly different from Cat-II (p-value=0.6305) *i.e.,* the categories are similar to each other. As shown in Figure 3, we observe that Cat-I and Cat-II projects have a similar proportion of multi-language pull requests (multi-PR): both Cat-I and Cat-II have the same median (39,08 for Cat-I and 39,09 for Cat-II), while the variance of Cat-I is larger.

Discussion: We observe from the results that machine learning frameworks follow the same multi-language programming trend as traditional projects. Out of the top 20 open source machine learning frameworks identified by Braiek *et al.* 's study [7], we analyze their source code and defined the different languages used. We find that only 35% (seven frameworks) are mono-language frameworks, while 65% (13 frameworks, of which we studied the largest 10 frameworks for our study) are multi-language. In other words, machine learning developers are generally aware of the benefits of multi-language development, and are equally able to attract open-source contributions just like the traditional (*i.e.,* non-ML) open-source projects. The next RQs analyze to what extent those contributions are successful in getting accepted and of high quality (bug-free).

The sets of Cat-I and Cat-II are comparable according to the usage of multi-language development (Figure 1), the number of pull requests (Figure 2), and the percentage of multi-language pull requests (Figure 3).

RQ2. *What is the impact of multi-language development on pull request acceptance ratio in machine learning frameworks?*

Motivation: Existing research shows that multi-language development requires substantial additional effort from software developers [4]. Rahman and Roy [23] also report that programming languages involved in pull requests can influence the success and failure rates of the pull requests.

Since pull requests represent the process through which a collaborator contributes in a software project, this research question aims to study the impact of multi-PR and mono-PR on the pull request acceptance ratio of Cat-I projects. We compare our results to those of Cat-II and Cat-III (mono-PR only) projects.

Results: We did not find a significant difference between the proportion of accepted pull requests in both Cat-I and Cat-II. Figure 4 shows the total percentage of all accepted pull requests (both multi-PR and mono-PR) in the two project categories. While the Figure shows that the acceptance ratio in Cat-II generally exceeds the acceptance ratio in Cat-I, the chi-square test shows an insignificant difference with a p-value = 0.08. Hence, both Cat-I and Cat-II are, in general, equally likely to accept PRs.

Given this inconclusive result, we perform further analysis to compare the acceptance ratio of multi-PR and mono-PR (relative to the totality of pull requests).

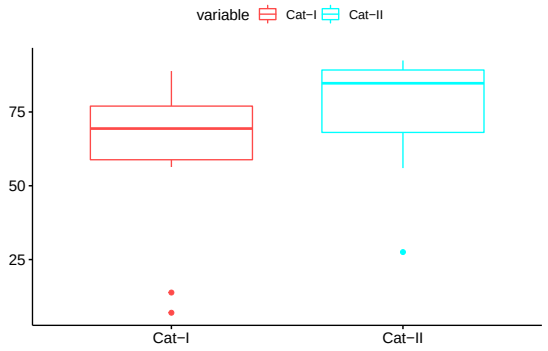

Fig. 4: %Accepted pull requests.

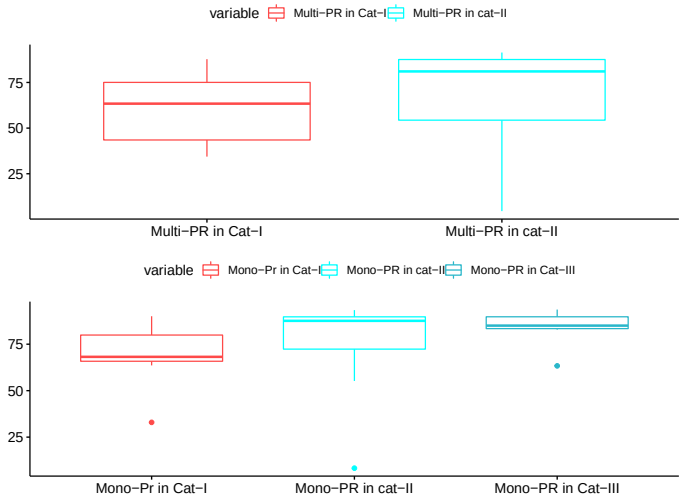

Fig. 5: Acceptance ratio in multi-/mono-language pull requests.

We did not find a significant difference between the multi-language acceptance ratio between Cat-I and Cat-II (same for the mono-language acceptance ratio). We observe from Figure 5a that Cat-II projects have a generally higher multi-PR acceptance ratio than Cat-I projects. Also, mono-PR in Cat-II seems to have a higher acceptance ratio than mono-PR in Cat-I, as shown in Figure 5b. However, the Chi-Square test did not show a significant difference in either the multi-PR acceptance ratio (p-value=0.315) or the mono-PR acceptance ratio (p-value=0.5993) comparisons between Cat-I and Cat-II projects. This finding shows that ML (Cat-I) and non-ML (Cat-II) projects have similar acceptance ratio even at the finer granularity of mono- and multi-PRs.

In Figure 5b, we further compare the acceptance ratio of mono-PR in Cat-III with the mono-PRs in Cat-I and Cat-II. We observe that the median percentage of accepted mono-PR of Cat-III projects is lower than that of Cat-II projects, but higher than the median accepted mono-PR of Cat-I projects. The Chi-Square test shows that these differences are significant, with p-values of 0.002269 and 0.01169, respectively.

Mono-PR have a significantly higher acceptance ratio than multi-PR in Cat-I projects, while there is no such difference in Cat-II projects. Multi-PR in Cat-I (Figure 5a) and mono-PR in Cat-I (Figure 5b) show a significant difference with a p-value of 0.02313. However, multi-PR (Figure 5a) and mono-PR (Figure 5b) in Cat-II did not show a significant difference (p-value = 0.227).

Discussion: Multi-language programming has been presented as a solution for diverse problems, but, at the same time, it represents a difficult practice that needs to be used carefully. ML is a relatively new domain and the development of ML systems requires competences of both software developers (experience in programming languages) and data scientists (experience in ML algorithms and the involved data). Pull request reviewers could be either data scientists or software developers, and either volunteers or employees, as discussed by Braiek *et al.* [7]. Thus, the acceptance ratio of a multi-PR could vary depending on the difference in the expertise of the reviewers involved. In other words, the findings in this RQ highlight that the interaction between the complexity of the ML domain and of multi-language development can have an impact on the contribution review process. Future work should consider this issue in order to support ML framework developers and reviewers in dealing with multi-PRs.

Multi-language pull requests are significantly harder to get accepted than in traditional projects in ML frameworks.

RQ3. *What is the impact of multi-language development on the period taken to accept pull requests in machine learning frameworks?*

Motivation: While the previous RQ's observations in terms of PR acceptance ratio are able to provide some insights, they do not tell the full story, since a multi-language PR that took a lot of time and effort to be accepted might still indicate a kind of overhead imposed by multi-language development. This motivate us to investigate the period taken for a multi- or mono-PR to be accepted or rejected. Since the period until a PR is accepted/rejected can be impacted by several factors, we control the time required with (1) the effort required to review each specific PR (approximated by the number of changed files in the PR), as well as (2) the size of the developer community.

Results: Pull requests take longer to be rejected than accepted in both Cat-I and Cat-II, for both mono- and multi-PR.

Figure 6 shows the period taken (in hours) by a developer to accept or reject a mono-PR/multi-PR in all three project categories. From the figure, we can see that the period taken by the reviewers to reject a mono-PR (multi-PR) in Cat-I or to reject a mono-PR (multi-PR) in Cat-II is higher than the period taken to accept them. These observations are confirmed by the Mann-Whitney U tests, which yield p-values of 0.00033 (0.0068) and 0.00032 (0.00049), respectively. The effect size shows a large effect in all cases of r=0.87 (r=0.65) and r=0.87 (r=0.84), respectively.

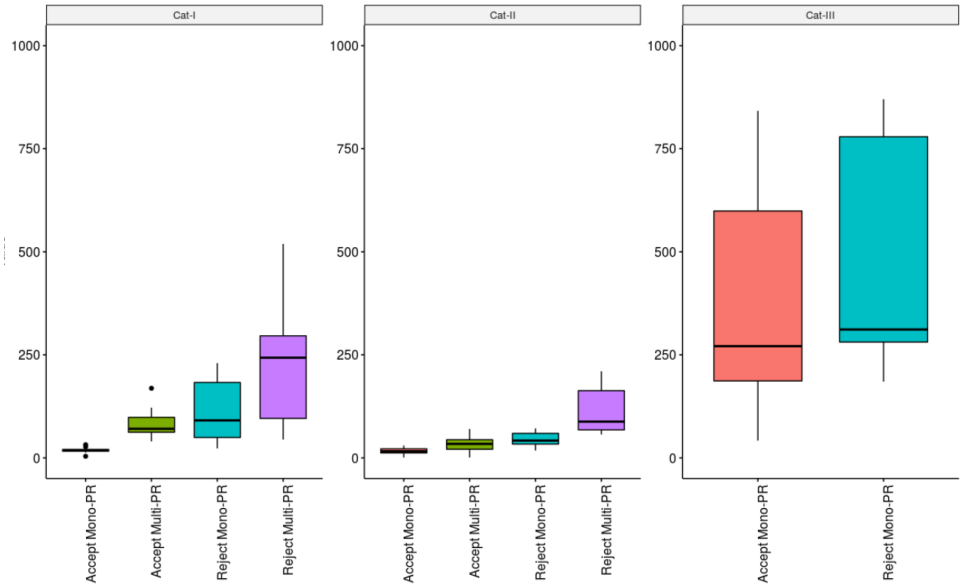

Fig. 6: Period taken (in hours) to accept/reject a multi-/mono-PR. (It should be noted that two outliers were removed from Cat-III's Accept Mono-PR (value = 1352.5) and Reject Mono-PR (value = 1552.5) to improve the presentation of the figure.)

Rejecting a multi-PR takes longer than rejecting a mono-PR in both Cat-I and Cat-II. Figure 6 shows that reviewers spend a significantly longer amount of period to reject a multi-PR in Cat-I than to reject a mono-PR in Cat-I , with a p-value of 0.016 and a medium effect size of 0.54. A similar observation can be made for Cat-II projects: it take longer for reviewers to reject multi-PR in Cat-II than to reject a mono-PR in Cat-II, with p-values of 0.00073 and a large effect size of 0.75.

Only in ML frameworks, multi-PR take significantly longer to accept than mono-PR. While we obtained a p-value of 0.00018 (and a large effect size of 0.83) for the comparison of multi-PR and mono-PR acceptance period in ML frameworks (Cat-I), the p-value for the corresponding comparison in traditional projects (Cat-II) do not show a significant difference (p = 0.037) due to the Bonferroni correction, as we divided the α by the number of tests (3) as discussed in Section II-E; thus, α became 0.05 / 3 = 0.017, which is lower than the p-value of 0.037. This finding shows that not only do multi-PRs have it harder to get accepted (as reported in RQ2), the PRs that are accepted generally take more time to do so as well.

We did not find significant differences between Cat-I and Cat-II in terms of the period taken to accept/reject mono-PRs. Hence, even though our comparisons within both categories showed some differences, the categories again are similar to each other, just like in RQ1.

ML frameworks (Cat-I) take longer to accept and reject a multi-PR than traditional systems (Cat-II). The period to accept (Reject) multi-PR in Cat-I is higher than Cat-II, with significant p-values: 0.0052 (0.023) with a large effect size of 0.62 and a medium effect size of 0.5, respectively.

The acceptance (rejection) of mono-PRs in Cat-III projects take equally long (and significantly longer) than the corresponding periods of Cat-I and Cat-II, respectively. We perform further analysis based on the mono-PR of Cat-III projects to understand whether earlier findings in this RQ apply across all ML project (since Cat-III contains only mono-language pull requests). The Mann-Whitney U tests comparing the mono-PR acceptance/rejection periods of Cat-III to the corresponding Cat-I and Cat-II periods all are significant. However, there is no significant difference between the mono-PR acceptance and rejection periods within Cat-III (p-value of 0.41).

These findings suggest that the observed differences in terms of the period to accept multi-PR compared to mono-PR for Cat-I are not necessarily due to the fact that the ML domain is more complex, since Cat-III projects seem to suffer much more from longer accept/reject periods than Cat-I/II. One possible confounding factor might be that the Cat-III projects receive larger pull requests than Cat-I/II, potentially explaining their slower review process. Hence, we discuss this confounding factor next.

Discussion: The more file changes in a PR, the longer its acceptance period and the shorter its rejection period (in both Cat-I and Cat-II). In Figure 7(a), we compare the relationship between the median number of changed files in multi-PRs (discussed in Section II-C) and the accepted (rejected) periods of multi-PR in both Cat-I and Cat-II projects. We can see that when the median number of multi-language files increases (X axis), the period spent (Y axis) to accept a multi-PR in Cat-I (red dots) and multi-PR in Cat-II (blue dots) increases as well. Spearman rank correlation showed a strong positive relationship with the following respective coefficient, p=0.88 and p=0.84.

However, the period spent to reject a multi-PR in Cat-I (yellow dots) and a multi-PR in Cat-II (gray dots) decreases

when the number of multi-language files increases. Spearman rank correlation showed a strong relationship (p=-0.90) only for Cat-I (p=-0.57 for Cat-II). This may be because when a pull request has many files, the reviewer does not invest much time to review it and instead asks to slice the large pull request in more manageable chunks [24].

For mono-PRs, including Cat-III, there is no correlation between the size of a PR and its acceptance/rejection period. In Figure 7(b), we compare the relationship between mono-language files and the accepted (rejected) periods of mono-PR in all three project categories. Spearman rank correlation showed a significant correlation only between the median of mono-language changed files (X axis) and the period taken for reject mono-PR in Cat-III (p=0.95). As future work, we will conduct more in-depth investigations to understand the reason behind this finding.

Cat-III projects have significantly fewer contributors than Cat-I and Cat-II. As an alternative explanation to the different PR acceptance/rejection periods between Cat-III and Cat-I/Cat-II (see Figure 6), we consider the size of a project's community *i.e.,* the number of contributors involved in each project. For example, a project with 100 contributors would have a larger pool of reviewers than a project with only 50 contributors, and hence might be more effective in reviewing, regardless of multi- or mono-PRs.

Figure 8 presents the distribution of the number of contributors per studied system. It shows that Cat-III projects have the least number of contributors compared to contributors in Cat-I and Cat-II. Our Mann-Whitney U tests show significant differences between both Cat-III vs. Cat-I (p-value= 0.014, effect size = 0.26) and Cat-III vs. Cat-II (p-value = 0.025, effect size = 0.51) comparisons. We conclude that the differences between the ML projects (Cat-III and Cat-I) in terms of mono-PR periods are not specific to the complexity or other characteristics of ML code, but rather due to the size of the developer community. A low number of contributors could cause delays in the revision process because there are not enough contributors for all the pull requests and this is what may causes pull requests to remain under revision for a long time before being accepted or rejected.

In ML frameworks, multi-language PRs take longer to be accepted than mono-language PRs and ML frameworks take longer to accept/reject a multi-PR than traditional systems.

RQ4. *Are multi-language pull requests more bug-prone than mono-language pull requests in machine learning frameworks?*

Motivation: Despite the diverse advantages of multi-language development, it presents some challenges to developers such as decreasing the quality and security of software systems [13]. In this research question, we aim to understand the correlation between multi-language development and the introduction of bugs in ML frameworks, compared to traditional systems. Our analysis focuses only on the accepted pull requests (rejected pull requests are incapable of introducing bugs since they are never merged into the code base). We compare our results across all three project categories.

Results: We only found a significant difference between the bug-proneness of mono-PRs of ML frameworks (Cat-I/III) and Cat-II. Figure 9 shows the percentage of bug-introducing multi-PRs and mono-PRs (relative to the total number of pull requests) in the studied project categories. As shown in Figure 9a, the median percentage of the bug-introducing multi-PR (mono-PR) of Cat-I projects is generally higher than the percentage of bug-introducing multi-PR (mono-PR) of Cat-II projects. However, only a significant difference was found between mono-PR of Cat-I and mono-PR of Cat-II (p-value = 0.0006976).

We did not find a significant difference between both Cat-I's buggy multi-PR vs. mono-PR (p-value = 0.2799) and Cat-II's buggy multi-PR vs. mono-PR (p-value = 0.1051) comparisons. Furthermore, statistical tests show a significant difference between the buggy mono-PRs in Cat-II and Cat-III (p-value = 0.0004885), but no significant difference was found between the buggy mono-PRs in Cat-I and Cat-III (p-value = 0.062.

Discussion: The results show a correlation between mono-PR and the introduction of bugs in ML frameworks. The lack of correlation between multi-PR and bug introduction can be explained by the fact that multi-PR in ML systems are less accepted than mono-PR as shown in (RQ2). Also, the longer time spent by reviewers before rejecting a multi-PR (as shown in RQ3) shows that reviewers pay more attention when reviewing multi-PR due to its complexity (*i.e.,* inter-language dependencies between the multi-language changed files) and the potential risks (bug introducing) it can cause. Since we analyze only the accepted PRs in this research question, we argue that a big percentage of risky multi-PR may already been cleaned in the review process.

Despite the longer acceptance period and lower acceptance ratio of multi-PR, no difference was found between the bug-proneness of Cat-I and Cat-II multi-PR. However, mono-PR in ML frameworks seem to be more bug-prone.

IV. Lessons learned and implications

RQ1. *What is the prevalence of multi-language development in machine learning frameworks?*

Nowadays, multi-language development is a fact in many machine learning frameworks, as we show in Table I where the most used ML frameworks in the industry such as Tensorflow, PyTorch, Theano, etc, all are multi-language. Furthermore,Ben Braiek *et al.* [7] showed that Python is the most commonly used language for developing machine learning frameworks, making the Python C extension a heavily used FFI in practice [10]. In contrast, recent research papers [4], [11]–[13] showed that in literature the most studied multi-language systems are the traditional ones, *i.e.,* non-machine learning frameworks, and that the most used combination of languages are Java and C/C++ interfaced via the FFI, *i.e.,* Java Native Interface.

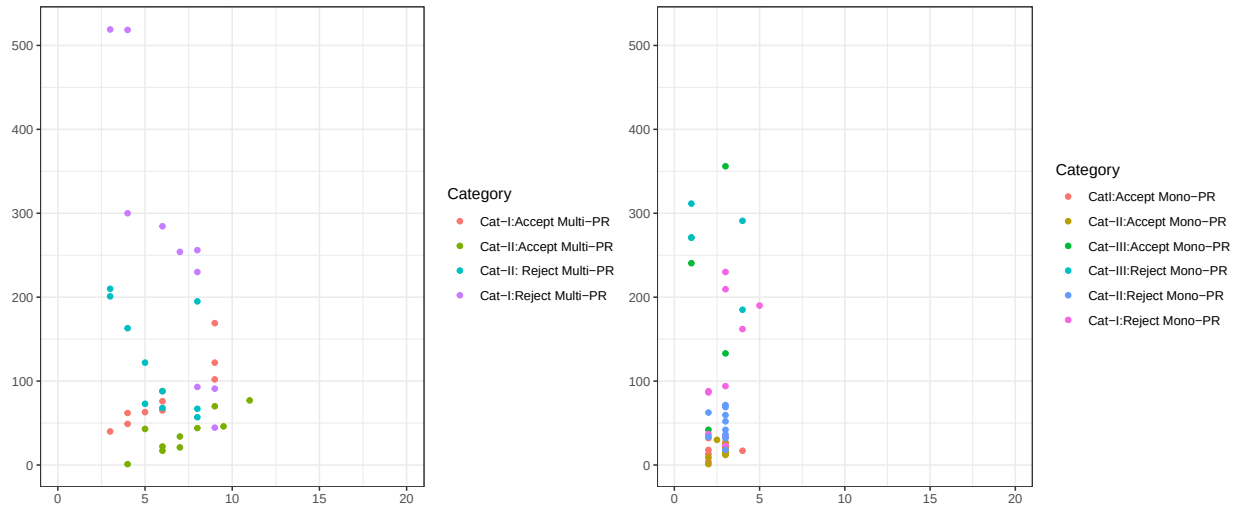

Fig. 7: Comparing period to accept/reject a multi-/mono-PR according to the changed files.

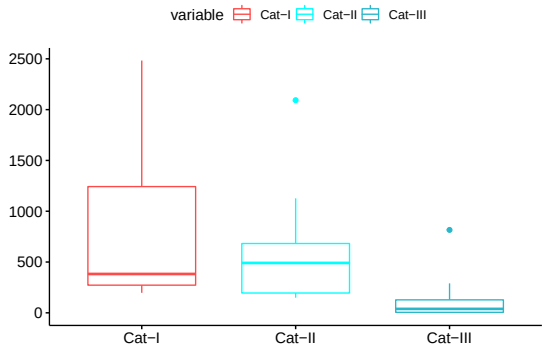

Fig. 8: Number of contributors per project.

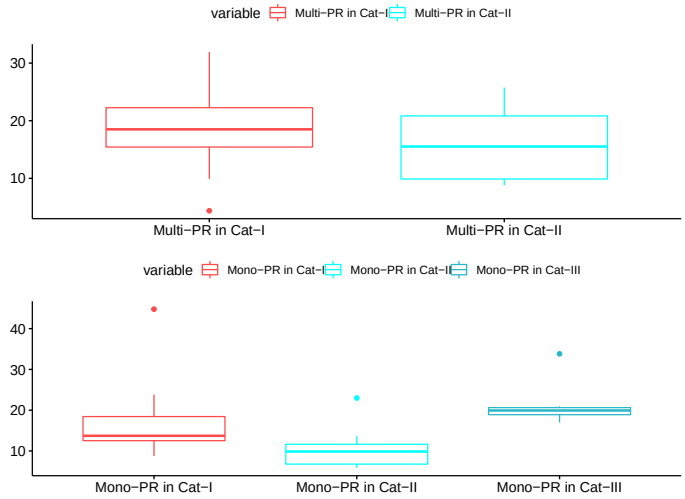

Fig. 9: %Bug-inducing pull requests

Hence, we suggest researchers to direct their focus towards machine learning frameworks as an important new class of multi-language systems that not only offers a new kind of large multi-language data-set, but also opens up a wide range of new challenges involving software maintenance, open source contributions, quality assurance, collaboration, etc.

RQ2. *What is the impact of multi-language development on pull request acceptance ratio in machine learning frameworks?*

Our results show that in machine learning frameworks, mono-language pull requests are more likely to be accepted than multi-language pull requests, while we do not observe such differences in the case of traditional systems. Based on this finding, we can assume that the machine learning domain presents more challenges for developers than traditional systems. For one, machine learning frameworks are a relatively new domain that requires developers to collaborate with data scientists and other roles in order to understand the complex implementation of ML algorithms.

We suggest that future research should further explore the challenges and issues involving program comprehension (*e.g.,* debugging) and maintenance of multi-language machine learning frameworks. Of particular importance will be qualitative studies of the kinds of changes made by machine learning experts, as well as to understand their development and comprehension needs. Furthermore, since machine learning is a multi-disciplinary domain involving multiple roles, our results presented in Section III will be beneficial for all roles involved.

RQ3. *What is the impact of multi-language development on the period taken to accept pull requests in machine learning frameworks?*

Our findings for RQ3 corroborate our earlier findings, since we find that in ML frameworks, multi-language PRs take longer to be accepted than mono-language PRs in ML frameworks, in addition to having a lower acceptance rate (cf. RQ2), however, no difference was found in traditional systems.

Thus, the challenge do not only come from the difficulties that offer multi-language development but from the combination of multi-language with machine learning frameworks. A possible explanation is that there is a lack of the tools dedicated to analyze and test the machine learning software contributions, the higher time spent to treat a machine learning pull request may be related to manual tests that a developer is doing. In future work, we plan to perform more deep study where we will try to classify the pull requests in terms of purpose, review actions, and reasons for delay. We recommend to the industry and researchers to investigate more these research directions in order to understand the exact reasons behind the more time-consuming ML pull requests.

RQ4. *Are multi-language pull requests more bug-prone than mono-language pull requests in machine learning frameworks?*

While RQ2 and RQ3 show that multi-language development in machine learning frameworks presents a challenge to developers and other roles in terms of effort required to get code contributions accepted, RQ4 do not show any difference in the bug-proneness of those contributions between multi-language machine learning frameworks and multi-language traditional systems.

Interestingly, we find that mono-PRs are more bug-prone in ML frameworks than mono-PR in traditional systems. One possible explanation is that bug-proneness is not related to multi-language development but instead is more related to the difference in domain (machine learning frameworks versus non-machine learning frameworks). Another explanation could be related to the fact that more mono-PRs are accepted in ML frameworks than multi-PRs, hence the larger volume of mono-PRs compared to multi-PRs could impact bug-proneness. More analysis is required to fully understand these observations.

V. Threats to validity

Threats to internal validity: Threats to the internal validity of our study concern the selected projects, the scripts used, the SZZ algorithm, and the pull requests analysis methodology. To mitigate these threats, first, we relied on the literature to identify projects shown to be among the largest projects in terms of lines of code and contributions. Then, we developed diverse python scripts that we ran on GitHub API. We ensured the validity of the scripts' outcome by performing a manual validation on a sample.

Last, regarding the SZZ, it is true that recent works have been improving the SZZ algorithm to increase its accuracy in specific cases [25]–[27]. However, the main challenge with those improved versions is that they had to give up the initial SZZ algorithm's language independence in favour of specialized language-specific analysis and optimizations. This makes those approaches less compatible with our paper on multi-language development, unless substantial effort is spent to adapt the tools to the studied languages. Hence, in this paper, we chose to use the basic SZZ with a number of minor improvements to support JNI systems. To mitigate this threat, we also measured the precision and the recall of the H-MLDA approach based on the SZZ implementation that we used.

Threats to external validity: Threats to external validity concern the factors that could affect the generalizability of our findings. Our findings may not be generalizable for all the existing multi-language systems (including machine learning and traditional systems) since we only studied a sample of 27 open-source projects. Software system' characteristics could vary depending on different criteria and factors. However, to mitigate this threat, we selected the largest ML frameworks [7] and Java/C systems [4] and we ensured that subjects of both categories are comparable regarding programming languages and pull requests, as our results in RQ1 showed.

Threats to conclusion validity: Threats to conclusion concern the relationship between the treatments and the findings. To mitigate this threat, we used Mann-Whitney U test *i.e.,* a non-parametric test, and Chi-Square test for Independence to compare the different analysis results across the three project categories. Regarding Mann-Whitney U test, for the control of family-wise error rate, we used the Bonferroni correction to calculate an adjusted p-value whenever the same sample is tested more than once.

Threats to reliability validity: We provided a companion package[1] with all the needed data, scripts, and results to replicate this study.

VI. Related Work

A. Multi-language development in traditional (non-ML) systems

Multi-language development has become a popular solution to many development problems faced by software developers. In a study of the state-of-the-art in on multi-language development by Mushtaq *et al.* [2], the authors highlight the many advantages of multi-language development, such as code re-usability and improved software performance. They stress the importance of addressing the complexity that arises from using more than one language in software systems, giving an overview of the existing code comprehension and maintenance tools specific for multi-language development, and elaborating on their advantages and limitations.

Several approaches have been proposed to address the challenges posed by multi-language development on code comprehension and maintenance. Kullbach *et al.* [28] presented an approach for program comprehension in multi-language systems. They showed that program comprehension plays an essential role in improving the efficiency of software development and maintenance processes in multi-language systems.

Bissyandé *et al.* [29] investigated the popularity, interoperability, and the impact of multiple programming languages in open-source projects from GitHub where they analysed this impact based on the software quality attributes. Similar to their study, we analyze the impact of multi-language programming on ML quality *e.g.,* bug-proneness.

Kochhar *et al.* [30] investigated the impact of using several programming languages on the quality of 628 GitHub

projects (traditional projects). They found that using different programming languages significantly increases bug proneness. They suggest further studies for design patterns and defects that should be used in systems that practice multi-language development. In our study, we extend this analysis to study this impact of multi-language programming on ML frameworks.

Abidi *et al.* [12] surveyed 93 developers to assess their level of knowledge about the good and bad practices of multi-language development. They proposed a set of practices initially collected from the literature. The survey was done on the proposed practices, where they found that these practice are not equally prevalent in the industry. They recommended that developers need to pay more attention to the best practices of multi-language development, such as managing exceptions between Java and C, loading libraries, etc,. Our paper complements this work by considering the application of the multi-language development in a new domain *i.e.,* ML frameworks.

B. Multi-language development in ML systems

Traditionally, AI developers had been using conventional artificial intelligence programming languages *i.e.,* Lisp or Prolog. However, with the passage of time, they started using multiple languages for their ML development. Poggi *et al.* [31] proposed HOMAGE, an environment for the development of multi-agent systems using three object-oriented programming languages *i.e.,* C++, Common Lisp, and Java. Tasharrofi *et al.* [32] developed a modular framework written in multiple languages.

The use of multi-language development is also found in AI-based games. Phelps *et al.* [9] argue that multi-language development is propagating quickly proportionally with games which lead to development challenging. The multi-language trend has appeared as well in the gaming world and has been interpreted as a development revolution.

C. Software engineering practices and Machine Learning

Several studies on the adoption of traditional SE practices in the domain of ML have been conducted over the past years. Braiek *et al.* [7] investigate the relationship of open source software and machine learning frameworks. They analyzed the influence and the negative impact of the application of SE in machine learning frameworks. They enumerated the advantage offered by the application of SE in machine learning. Our paper studies 17 of the 20 largest and well-known open-source machine learning frameworks presented in their study.

Khomh *et al.* [33] are interested in the SE challenges for machine learning frameworks, where they highlighted the importance and the need that software engineering (SE) and ML communities to cope together to deal with these challenges. They enumerated two main challenges of practicing SE in ML: software testing and software evolution. Dhasade *et al.* [34] proposed a solution (Prioritizer tool) to support developers to handle large volumes of issues in projects. They developed a machine learning based solution for prioritizing pull requests to fix issues. Prioritizer tool is based on three criteria: issue lifetime, hotness, and category. Authors evaluated their solution' accuracy by testing it on a data-set of 3000 issues. Veen *et al.* [35] proposed a tool for pull request prioritization called "PRioritizer". This tool uses machine learning to work as a priority inbox for pull requests, recommending the top pull requests the project owner should focus on. Zhao *et al.* [36] proposed a learning-to-rank (LtR) approach for recommending pull requests that can be quickly reviewed by reviewers.

VII. Conclusion And Future Work

Nowadays, there is an increasing trend in the practice of multi-language development in the software engineering domain. Despite the numerous advantages of multi-language development such as reusing legacy code and improving computational performance (*e.g.,* with C code), there are also new challenges related to the complexity of code comprehension and maintenance of such systems. In this paper, we study the prevalence of the multi-language practice in machine learning (ML) frameworks. Since the challenges of multi-language development in traditional systems were a subject of interest of several existing research studies, we perform, throughout our study, a comparison of our analysis results between ML and traditional multi-language systems.

Our major results show that (1) Multi-language PRs in ML frameworks have a lower acceptance ratio than mono-language PRs; (2) multi-language PRs in ML frameworks take longer to be accepted than mono-language PRs, and ML frameworks take longer to accept/reject a multi-PR than traditional systems; and (3) mono-language PRs of ML frameworks are more bug-prone than traditional systems. Other characteristics were found to be similar between the studied ML and traditional projects.

The paper's findings provide a correlation between the existence of multi-language development with machine learning. Multi-language influence the software contributions (Pull requests) to ML frameworks as discussed in RQ2 and RQ3. We recommend to data scientists as well as software developers to merge their effort towards understanding multi-language in ML and to deal with its complexity to benefit from the advantages that offer.

In future work, we plan (1) to extend this study to apply it on more systems for the generalization; (2) to conduct a deeper study on confounding factors *i.e.,* any contributor and other background factors that could impact the review process; (3) propose a catalogue of good and bad practices (defined through interviews and empirical studies) in using multi-language within ML frameworks and validate it through a survey; and (4) perform a qualitative study to understand the reasons behind the higher acceptance ratio in traditional systems than ML frameworks and the longer acceptance period in ML frameworks than traditional systems.

References

[1] F. Boughanmi, "Multi-language and heterogeneously-licensed software analysis," in *2010 17th Working Conference on Reverse Engineering*, Oct 2010, pp. 293–296.

[2] Z. Mushtaq and G. Rasool, "Multilingual source code analysis: State of the art and challenges," in *2015 International Conference on Open Source Systems Technologies (ICOSST)*, Dec 2015, pp. 170–175.

[3] T. Arbuckle, "Measuring multi-language software evolution: A case study," in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, ser. IWPSE-EVOL '11. New York, NY, USA: ACM, 2011, pp. 91–95.

[4] M. Grichi, M. Abidi, Y.-G. Guéhéneuc, and F. Khomh, "State of practices of java native interface," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '19. USA: IBM Corp., 2019, p. 274–283.

[5] M. Furr and J. S. Foster, "Checking type safety of foreign function calls," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. New York, NY, USA: ACM, 2005, pp. 62–72.

[6] S. Li and G. Tan, "Finding reference-counting errors in python/c programs with affine analysis," in *Proceedings of the 28th European Conference on ECOOP 2014 — Object-Oriented Programming - Volume 8586*, 2014, pp. 80–104.

[7] H. Ben Braiek, F. Khomh, and B. Adams, "The open-closed principle of modern machine learning frameworks," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, May 2018, pp. 353–363.

[8] G. Varisteas, T. Avanesov, and R. State, "Distributed c++-python embedding for fast predictions and fast prototyping," in *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, 2018, pp. 9–14.

[9] A. M. Phelps and D. M. Parks, "Fun and games: Multi-language development," *Queue*, vol. 1, no. 10, pp. 46–56, 2004.

[10] S. Buro and I. Mastroeni, "On the multi-language construction," in *European Symposium on Programming*. Springer, 2019, pp. 293–321.

[11] M. Abidi, F. Khomh, and Y. Guéhéneuc, "Anti-patterns for multi-language systems," in *Proceedings of the 24th European Conference on Pattern Languages of Programs, EuroPLoP 2019, Irsee, Germany, July 3-7, 2019*. ACM, 2019, pp. 42:1–42:14.

[12] M. Abidi, M. Grichi, and F. Khomh, "Behind the scenes: Developers' perception of multi-language practices," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '19. USA: IBM Corp., 2019, p. 72–81.

[13] M. Abidi, M. Grichi, F. Khomh, and Y.-G. Guéhéneuc, "Code smells for multi-language systems," in *Proceedings of the 24th European Conference on Pattern Languages of Programs*, ser. EuroPLop '19. New York, NY, USA: Association for Computing Machinery, 2019.

[14] K. Patel, J. Fogarty, J. A. Landay, and B. L. Harrison, "Examining difficulties software developers encounter in the adoption of statistical machine learning." in *AAAI*, 2008, pp. 1563–1566.

[15] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, and Q. Yu, "Why is developing machine learning applications challenging? a study on stack overflow posts," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, pp. 1–11.

[16] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *SIGSOFT Softw. Eng. Notes*, 2005.

[17] S. Kim, E. J. Whitehead,, and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, 2008.

[18] M. Hollander, D. A. Wolfe, and E. Chicken, *Nonparametric statistical methods*. John Wiley & Sons, 2013, vol. 751.

[19] *Analysis of Clinical Trials Using Sas®: A Practical Guide*, 1st ed. SAS Publishing, 2005.

[20] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." 1993.

[21] M. McHugh, "The chi-square test of independence," *Biochemia medica*, vol. 23, pp. 143–9, 06 2013.

[22] D. Zhang and J. J. Tsai, "Machine learning and software engineering," *Software Quality Journal*, vol. 11, no. 2, pp. 87–119, 2003.

[23] M. M. Rahman and C. K. Roy, "An insight into the pull requests of github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 364–367.

[24] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast? – case study on the linux kernel," in *Proceedings of the 10th IEEE Working Conference on Mining Software Repositories (MSR)*, San Francisco, CA, US, May 2013, pp. 101–110.

[25] E. C. Neto, D. A. da Costa, and U. Kulesza, "The impact of refactoring changes on the szz algorithm: An empirical study," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 380–390.

[26] E. C. Neto, D. A. d. Costa, and U. Kulesza, "Revisiting and improving szz implementations," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–12.

[27] M. Borg, O. Svensson, K. Berg, and D. Hansson, "SZZ unleashed: An open implementation of the SZZ algorithm - featuring example usage in a study of just-in-time bug prediction for the jenkins project," *CoRR*, vol. abs/1903.01742, 2019. [Online]. Available: http://arxiv.org/abs/1903.01742

[28] B. Kullbach, A. Winter, P. Dahm, and J. Ebert, "Program comprehension in multi-language systems," in *Proceedings Fifth Working Conference on Reverse Engineering (Cat. No.98TB100261)*, Oct 1998, pp. 135–143.

[29] T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillere, "Popularity, interoperability, and impact of programming languages in 100,000 open source projects," in *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*. IEEE, 2013, pp. 303–312.

[30] P. S. Kochhar, D. Wijedasa, and D. Lo, "A large scale study of multiple programming languages and code quality," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 1. IEEE, 2016, pp. 563–573.

[31] A. Poggi and G. Adorni, "A multi language environment to develop multi agent applications," in *International Workshop on Agent Theories, Architectures, and Languages*. Springer, 1996, pp. 325–339.

[32] S. Tasharrofi and E. Ternovska, "A semantic account for modularity in multi-language modelling of search problems," in *International Symposium on Frontiers of Combining Systems*. Springer, 2011, pp. 259–274.

[33] F. Khomh, B. Adams, J. Cheng, M. Fokaefs, and G. Antoniol, "Software engineering for machine-learning applications: The road ahead," *IEEE Software*, vol. 35, no. 5, pp. 81–84, 2018.

[34] A. B. Dhasade, A. S. M. Venigalla, and S. Chimalakonda, "Towards prioritizing github issues," in *Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference*, 2020, pp. 1–5.

[35] E. Van Der Veen, G. Gousios, and A. Zaidman, "Automatically prioritizing pull requests," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 357–361.

[36] G. Zhao, D. A. da Costa, and Y. Zou, "Improving the pull requests review process using learning-to-rank algorithms," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2140–2170, 2019.