Understanding Open-Source Contributor Profiles in Popular Machine Learning Libraries

By Jiawen Liu

A thesis submitted to the Graduate Program in Electrical And Computer Engineering in conformity with the requirements for the Degree of Master of Applied Science

Queen's University
Kingston, Ontario, Canada
January, 2025

Abstract

With the increasing popularity of machine learning (ML), many open-source software (OSS) contributors are attracted to developing and adopting ML approaches. A comprehensive understanding of ML contributors is crucial for successful ML OSS development and maintenance. Without such knowledge, there is a risk of inefficient resource allocation and hindered collaboration in ML OSS projects. Existing research focuses on understanding the difficulties and challenges perceived by ML contributors by user surveys. There is a lack of understanding of ML contributors based on their activities tracked from software repositories. In this thesis, we aim to understand ML contributors by identifying contributor profiles in ML libraries. We further study contributors' OSS engagement from four aspects: workload composition, work preferences, technical importance, and ML-specific contributions. By investigating 11,949 contributors from 8 popular ML libraries (TensorFlow, PyTorch, scikit-learn, Keras, MXNet, Theano/Aesara, ONNX, and deeplearning4j), we identify four contributor profiles: Core-Afterhour, Core-Workhour, Peripheral-Afterhour, and Peripheral-Workhour. We find that: 1) project experience, authored files, collaborations, pull requests comments received and approval ratio, and geographical location are significant features of all profiles; 2) contributors in Core profiles exhibit

significantly different OSS engagement compared to Peripheral profiles; 3) contributors' work preferences and workload compositions are significantly correlated with project popularity; 4) long-term contributors evolve towards making fewer, constant, balanced and less technical contributions.

Co-authorship

This thesis is based on a research paper co-authored with Dr. Ying Zou and Dr. Haoxiang Zhang. The research was primarily conducted by me under the supervision of Dr. Ying Zou. Dr. Haoxiang Zhang participated in regular progress meetings and provided feedback and suggestions for my work.

Acknowledgments

First, I would like to express my deepest gratitude to my academic supervisor Dr. Ying Zou, for her unwavering support, guidance, and encouragement throughout my Master's journey at Queen's University. Dr. Zou has an incredible ability to recognize the potential in each of her students, helping us discover our strengths and encouraging us to work out our own unique research paths. Her dedication to research and her professionalism have greatly influenced my growth as a graduate student. The most valuable moments during the past two years of my study have been our in-depth weekly conversations, where Dr. Zou's deep knowledge and insightful feedback have continuously guided me in refining my work. Dr. Zou's genuine care for her students is evident in her thoughtful consideration of each student's circumstances, and she always strive to provide the best resources for our research, career, and personal growth. I feel truly honored to have been her student, and I know she will be a lifelong mentor to me.

I also want to thank my collaborator, Haoxiang Zhang. He is one of the most intelligent and knowledgeable researchers I have had the chance to work with. Haoxiang always listened to my thoughts, helped me navigate challenges, and shared his brilliant ideas during our discussions. These conversations have been a great driving force in my research journey. Beyond academics, his advice on career and life has

provided valuable guidance for my future.

I feel incredibly fortunate to have worked with the amazing colleagues at Software Evolution & Analytics Lab (SEAL): Dr. Guoliang Zhao, Dr. Shayan Noei, Dr. Maram Assi, Fangjian Lei, Yiping Jia, Pouya Fathollahzadeh, Chunli Yu, Bihui Jin, Jonathan Cordeiro, Liam Johnston, and Lekhashree H J. Their kindness and expertise have made SEAL an inspiring place to work and grow, and I am grateful for the opportunity to collaborate and learn from such wonderful individuals.

A special thank goes to my friends Xinran Li, Alice Huang, Lily Wang and Yuxi Zhang. Your companionship during those long, sleepless nights of study has eased my worries and fears.

Lastly, and most importantly, I am deeply grateful to my parents for their unwavering love, patience, and belief in me. Their emotional support has been the foundation upon which I could finish this journey. Thank you for always being there to celebrate my successes, big and small, and to lift my spirits during tough times.

To all who have been part of this journey, thank you from the deep of my heart.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Open Source Software (OSS) has emerged as a dominant model in software development, gaining widespread recognition among enterprises and developers as the preferred approach for software development. The OSS community comprises globally distributed contributors and users with shared interests, who actively participate in knowledge sharing and collaborate on the development and maintenance of software projects. Anyone with the necessary knowledge and skills can be a contributor to OSS projects. They can contribute in various ways, such as writing source code, updating documentation, reporting issues, conducting code reviews, and participating in discussions. These activities are critical to the success of the OSS development process and evolution. The rising popularity of open-source machine learning (ML) libraries, such as Tensorflow[1] and PyTorch[2], has facilitated ML implementation, attracting a growing number of software developers to learn ML technologies and contribute to ML projects [1].

[1] https://www.tensorflow.org/
[2] https://pytorch.org/

1.1 Research Challenges

Developers are the most important resource in software development and maintenance. A deep understanding of developer team composition could provide valuable insights for software development management. This is particularly important in OSS projects, as contributors are distributed around the globe and have diverse backgrounds. Therefore, it is challenging for project managers and maintainers to have comprehensive knowledge to facilitate contributor productivity.

In traditional software engineering, extensive research has been conducted to portray or categorize OSS contributors from many specific aspects, such as the volume of contribution [2–4], technical expertise [5, 6], social networks [7, 8], duration [9, 10], and contribution dynamics [11]. However, few have endeavored to provide a comprehensive understanding of OSS contributors considering their behaviors, expertise, workload, work preferences, and the importance of their contributions. A lack of comprehensive insight may lead to ineffective resource allocation or collaborative challenges. For example, as reported by Balali et al. [12], contributors face challenges in their collaborations such as communication issues caused by timezone, language, and cultural differences, difficulties in managing time to collaborate, mismatch of knowledge background, and harsh project atmosphere.

To investigate the challenges faced by ML developers, existing studies primarily conduct user surveys and interviews to collect insights into their experiences, demands, and pain points [13–15]. Han et al. [16] conduct an empirical study on the onboarding process for newcomers to deep learning projects. However, there is a lack of studies that comprehensively portray ML contributors and their characteristics and support with quantitative analysis. This gap in understanding limits the ability to

promote effective collaboration, optimize task assignments, and develop strategies to ensure sustained growth and popularity of ML projects. It also limits the ability to provide suggestions for contributors with diverse characteristics.

1.2 Thesis Statement

Despite the increasing participation of OSS contributors in ML open-source projects, there is a lack of study aiming to comprehensively understand different types of ML contributors and their OSS engagements based on large-scale data of their activities in the software repositories. To enhance our understanding of ML contributors, this thesis conducts an empirical study on contributors of 8 popular ML libraries: TensorFlow, PyTorch, scikit-learn, Keras, MXNet, Theano/Aesara, ONNX, and deeplearning4j. We aim to categorize the contributors based on their activities, specifically their work time, total commits, code contribution density, and number of programming languages used. We hypothesize that these four features could capture the key behaviors of OSS contributors and identify distinct groups of contributors. Furthermore, we analyze contributors' engagement in ML OSS projects across four key dimensions: 1) workload composition, 2) work preferences, 3) technical importance, and 4) contributions to various components of ML frameworks. Moreover, we examine the evolution of individual contributors' engagement over time to understand how their behaviors change over time.

The insights gained from this analysis aim to advance the research community's understanding of the composition and characteristics of ML OSS contributors. The findings in this thesis can help various types of ML contributors to progress towards

core contributors, provide suggestions to project maintainers for optimizing task assignment strategies and sustaining project popularity, and highlight future research directions to further support the ML OSS community.

1.3 Research Questions

Particularly, we aim to address the following research questions:

RQ1: What are the characteristics of contributor profiles? We categorize ML contributors and construct contributor profiles based on four fundamental open-source contribution behaviors: working hours, number of commits, code contribution density, and programming language usage. This research question aims to provide the detailed characteristics of each contributor profile by examining a comprehensive set of open-source contribution behaviors and identifying the important features defining each profile.

RQ2: What is the OSS engagement of each contributor profile? In this research question, we compare contributor profiles from four aspects of OSS engagement: **workload composition** that describes a contributor's focus of work on the five key OSS activities within a period (i.e., reporting issues, issue discussions, commits, pull request discussions, and code reviews); **work preferences** that capture the dynamics and balance of a contributor's contributions within a period; **technical importance** that measures the importance of a contributor's contribution; and **ML-specific contributions** which categorize the content of contributions into 5 ML components (i.e., User-Level API, Graph-Level Implementation, Operation Implementation, General Utility, and Environment-Dependent Processing) and general software development. This multifaceted analysis aims to provide project managers

and maintainers insights into contributor behaviors to promote effective collaboration and expertise-aligned task assignments. The results of this question also contribute to a deeper understanding of the intricate dynamics within ML OSS ecosystems.

RQ3: What are the important factors of contributor OSS engagement for increasing the popularity of a project? This research question aims to understand the association between various contributor OSS engagements and the project's popularity. We investigate the association between the distribution of contributor workload composition and work preferences with the growth of project popularity, measured by star ratings and forks. By identifying the significant features associated with project popularity, we provide insights into task assignment strategies that can help project maintainers sustain and enhance project popularity during their daily management.

RQ4: How does contributor OSS engagement evolve? The project's demand for different types of contributors evolves as the project develops and the individual contributor's engagement also shifts as they stay longer in the project. This research question aims to explore two dimensions of evolution in ML OSS projects: (1) the changing distribution of contributors across different workload compositions in each ML project, and (2) the shifting workload composition, work preferences, and technical importance for individual contributors in each profile. Recognizing such evolutionary trends may provide project maintainers the insights into the dynamics of long-term contributors and contribute to a deeper understanding of the evolution dynamics within ML OSS projects.

1.4 Contributions

In summary, this thesis makes the following contributions to the software engineering community:

(1) Provide a comprehensive understanding of ML contributors from both static and dynamic points of view by studying their activities in a period and analyzing their engagement over time.

(2) Identify four contributor profiles in eight popular ML libraries. These profiles help establish an initial understanding of ML contributors with quantitative analysis of their OSS activities.

(3) Identify the association of contributors' work preferences and workload compositions with the growth of project popularity.

(4) Highlight differences in the distribution of ML-specific contributions between core and peripheral contributors, providing insights into how profiles vary in terms of ML-specific expertise and the code contributions they make to the project. We also provide a dataset of 384 pull requests with manually verified ML-related categories.

1.5 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 provides a literature review of related work to this thesis.

Chapter 3 describes data collection and case studies. This chapter provides detailed explanations of how contributor profiles are established and how contributor engagement is measured, including workload composition, work preferences, technical importance, and contributions to different components of ML frameworks.

Chapter 4 presents the motivations, approaches, and results of four research

questions.

Chapter 5 presents the discussion and implications of our findings. This also includes the suggestions made to the OSS contributor, project maintainers, and software engineering researchers.

Chapter 6 discusses the potential threats to the validity of our study.

Lastly, we conclude the thesis and recommend future work in **Chapter 7**. Supplementary materials and results are provided in **Appendix A**.

To facilitate the replication of this thesis, we provide our data and code publicly available at:
https://github.com/Software-Evolution-Analytics-Lab-SEAL/ML_Contributor_replication.

Chapter 2

Literature review

This chapter provides an overview of research on open-source software (OSS) contributors from aspects: contributors to traditional software and contributors to machine learning (ML) software.

2.1 OSS Contributor Characterization

Historical data in open-source software repositories can provide insights into software developers. Many existing studies in software engineering have categorized the behaviors of OSS contributors and evaluated their expertise in specific areas.

Researchers begin with classifying OSS contributors according to their amount of code contribution. Mockus et al. [3, 4] find that the top 20% active developers are usually the core developers who contribute more than 80% commits in OSS projects. Based on this finding, Robles et al. [17] further study the turnover of the core developers and propose a quantitative methodology to classify the behaviour of human resources at a project level. Nakakoji et al. [18] propose an onion model to classify OSS community members into seven categories: project leaders, core members, active developers, peripheral developers, bug fixers, bug reporters, readers, and passive

users. Milewicz et al. [19] find that open-source scientific software projects are driven by senior members of the team (e.g., professors), and they make 72% commits on average and mainly tackle architectural problems, while junior members (e.g., graduate students) mainly work on implementing new features. Costa et al. [2] categorize developers as core, active, and peripheral, and find that core developers make the majority of contributions and are associated with less buggy commits.

The duration of involvement in OSS projects is a key factor in categorizing contributors and has been extensively studied. Pinto et al. [20] find that half of the newcomers in a project only make one contribution and then depart from the project. Steinmacher et al. [21] find that more than 80% newcomers do not become long-term contributors. Zhou et al. [10], Wang et al. [22] and Bao et al. [9] develop prediction models to identify potential long-term contributors based on their initial behaviors in the project repository.

Technique expertise is an important aspect of differentiating contributors and identifying experts. Dey et al. [5] propose an approach to represent contributors' expertise with their API usage. Dakhel et al. [23] represent the domain expertise of contributors based on the projects they have contributed to, issues solved, and their API usage. Ahasanuzzaman et al. [24] propose an approach that uses language-specific functions and APIs to assess one's efficiency in a programming language. Montandon et al. [25] explore the effectiveness of clustering and machine learning classifiers in assessing contributors' expertise and identifying experts based on GitHub activities. They find that clustering methods yield better results compared to supervised classifiers. Yang et al. [26] develop a tool to visualize developers' portraits based on their Github profile pages and coding details.

There are studies exploring the impact of personal aspects on OSS contributors. Cataldo et al. [27] investigate the correlation between the quality of software projects and the geographical distributions of developers and find that projects with imbalanced distributions of the development team tend to have more defects compared to projects with balanced distributions. Claes et al. [28] conduct a large-scale study on the working hours of developers and find that approximately two-thirds of developers follow normal office hours, and no correlation between project maturation and the reduction in abnormal working hours. Yue et al. [11] find that the preference of contribution dynamics in the early career affects the contributor's technical success in the project. In contrast, we study the work preferences of the contributor's entire career in ML projects as well as the impact of contributors' work preferences on the project's popularity. Elbaum et al. [29] identify a strong association between code churn and defects. Shin et al. [30] report code churn as one of the software vulnerability indicators, indicating that the style of contributors' contributions, such as the size of their commits, can significantly impact the quality and security of the codebase.

The prior research primarily examines traditional software projects, while our study concentrates on contributors within popular machine learning projects. Unlike the existing studies that often investigate contributors from certain angles contributors (e.g., the volume of code contribution, duration, geographical distributions, or contribution dynamics), we aim to construct a comprehensive profile of open-source machine learning contributors by considering multiple perspectives and providing a holistic understanding of the characteristics and behaviors of members of the ML OSS community.

2.2 Research for Machine Learning Contributors

Research efforts to study ML software developers have been arising within the past few years. Cai et al. [13] deploy a survey to investigate the motivations, challenges, and demands of software developers when they begin learning ML, and find that the major challenge of ML beginners is the lack of understanding of ML concepts and mathematics, and they demand ML frameworks to provide such support. Hill et al. [14] interview 11 ML professionals from a company and find that their main challenge is creating a repeatable workflow for ML system development process. Ishikawa et al. [15] conduct a questionnaire survey on 278 ML practitioners and report the main difficulties perceived by ML contributors including low accuracy, absence of an oracle, and uncertainty regarding system behavior. Gangash et al. [31] conduct an empirical study analyzing ML-related posts on Stack Overflow and explore the topics frequently discussed by ML developers. Islam et al. [32] map the posts related to 10 popular ML libraries on Stack Overflow to ML system development stages. They identify model construction and data preparation as the most challenging stage of ML system development perceived by ML developers. Han et al. [16] study the onboard process for deep learning newcomers based on their activities before their first accepted commit in deep learning projects. In contrast, we study the complete duration of ML contributors in OSS ML projects and their evolution.

Previous research on ML contributors has primarily explored their motivations for learning ML and the challenges encountered in ML software development. These studies often use methods, such as surveys, interviews, and analyzing posts from Q&A systems to delve into the contributors' mentalities, focusing on their thoughts

and philosophies. In contrast, our approach is centered on understanding ML contributors and aims to establish a categorization of ML contributors based on their OSS activities. The results of our work provide a comprehensive understanding of different types of contributors in the ML OSS community and their behavioral characteristics.

Chapter 3

Case Study

In this chapter, we present data collection and our case studies. An overview of our approach is shown in Figure 3.1. We first collect the subject projects of ML OSS and their historical data from GitHub. Preprocessing is conducted on the collected project data, which involves removing non-human contributors and cleaning noises that may affect contributor feature calculation. Then, we extract ML contributors within the subject projects and extract contributor features from the processed data. Lastly, we identify contributor profiles, workload composition patterns, work preference features, and technical importance.

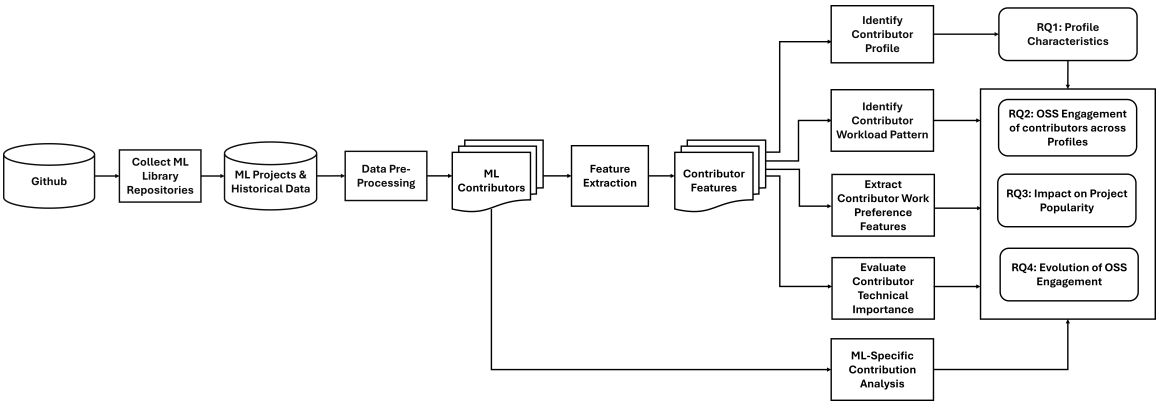


Figure 3.1: The overview of our approach.

3.1 Data Collection

We select the subject projects and collect the history data from Github using Github REST API [33] and PyDriller [34].

Project Selection. To conduct our experiments on projects that contain rich contributor information and the most up-to-date machine learning technology, we choose to study projects related to ML libraries and frameworks. These projects gather collaborations among ML experts and usually have a larger size than the projects that implement a single machine learning model to solve a problem. We review existing works that studied popular machine learning libraries and frameworks for various purposes [16, 35–73] (details can be found in our online supplementary material[1]). From this literature review, we select representative ML libraries that are widely used and commonly studied within the software engineering community. The selection of these libraries is critical to ensure our results can be comparable with the related work on a similar set of ML libraries and frameworks.

Based on their frequency in the reviewed literature, we identify the following ten popular ML libraries: TensorFlow (36 mentions), PyTorch (32), Keras (25), Caffe/-Caffe2 (18), Theano/Aesara (16), Scikit-learn (12), MXNet (11), CNTK (8), Chainer (6), and deeplearning4j (4). However, we exclude Chainer, CNTK, Caffe, and Caffe2 as their GitHub repositories have not been actively maintained or updated since 2019 or 2020. Although Theano is no longer under development after 2017, its continued project, Aesara, is active at the year of our data collection (i.e., 2023). Therefore, we consider Theano and Aesara as the same project and keep them in our subject

[1]https://github.com/seal-tosem/replication/blob/main/project_selection_literatures.pdf

Table 3.1: Descriptive statistics of the selected ML library projects.

Project Name	Contributors	Commits	Pull Requests	Issues	Creation Time
Tensorflow[2]	3,499	147,242	23,518	37,020	2015-11
PyTorch[3]	3,085	60,834	77,491	39,364	2012-01
scikit-learn[4]	2,647	27,880	16,125	9,160	2010-01
Keras[5]	1,139	9,967	6,843	11,606	2015-03
MXNet[6]	836	11,535	10,919	7,763	2015-04
Theano\Aesara[7][8]	403	24,990	4,713	2,497	2008-01
ONNX[9]	274	2,505	3,035	2,387	2012-01
Deeplearning4J[10]	66	2,681	4,306	5,115	2019-06

projects. We also include ONNX in our subject projects because, despite its appearance only once in the literature we explored, it is a newly emerged and rarely studied library crucial for ML deployments. Its significance is widely recognized, and prior studies, such as [74], have highlighted it as an area for future research. Thus, our final selection consists of eight open-source ML libraries: TensorFlow, PyTorch, Keras, Theano/Aesara, Scikit-learn, MXNet, deeplearning4j, and ONNX.

All the selected projects contain more than 2k commits and have been active for more than 5 years, ensuring they provide sufficient information for our analysis. A detailed overview of the subject projects is presented in Table 3.1.

Collecting Data. After the subject projects are selected, we retrieve the project development data from their repositories on Github. We use Github REST API to collect (*i*) commits; (*ii*) pull requests; (*iii*) issue reports of each project, and (*iv*) the account type of contributors (e.g., bot, organization, or user account) in the projects. PyDriller [34] is used to fetch the commit timestamps in the local time of the commit authors and identify their time zones.

3.2 Data Preprocessing

3.2.1 Removing Nonhuman Contributors

We observe that bots are employed in our subject projects for various project management tasks, such as synchronizing changes from the public repository on GitHub to an internal repository (e.g., tensorflower-gardener[11] and facebook-github-bot[12]), labeling issues (e.g., mxnet-label-bot[13]), merging pull requests (e.g., pytorchmergebot[14]), or testing changes (e.g., tensorflow-jenkins). As the focus of this study is to understand human contributors, we apply a rigorous method to filter out bots. However, as AI becomes a promising driving force for software development, future work can further study bots to analyze their behaviors and how they corporate with human contributors in modern OSS development.

To filter out bots, we first extract all the usernames of contributors who make at least one commit in the selected ML projects. We then use the GitHub Users API[15] to check if each username belongs to a user, bot, organization, or enterprise type account and only retain those with user accounts. Next, we sort the contributors in each project by their total number of commits and manually investigate the remaining contributors to identify any contributors with abnormal usernames (e.g., contain 'bot' or contain similar characters as the project name) or behaviors (e.g., making a large number of commits without raising any pull requests or making comments). For each

[11] https://www.oreilly.com/content/how-the-tensorflow-team-handles-open-source-support/
[12] https://github.com/pytorch/pytorch/pull/8581
[13] https://github.com/apache/mxnet-ci/blob/master/services/github-bots/LabelBotFullFunctionality/README.md
[14] https://github.com/pytorch/pytorch/wiki/Bot-commands
[15] https://docs.github.com/en/rest/users

identified contributor, we check their Github profile, check if it is mentioned in the project ReadMe or WiKi, and search for any public announcements or discussions by the development team to confirm its use as a bot. As a result, we identify and remove 14 bots (i.e., tensorflower-gardener, tensorflow-jenkins, onnxbot, facebook-github-bot, pytorchmergebot, pytorchbot, docusaurus-bot, theano-bot, deadsnakes-issues-bot, mxnet-label-bot, snyk-bot, pytorchupdatebot, scikit-learn-bot, and inclusive-coding-bot). Then, we use the GitHub bot detection tool, BotHunter, proposed by Abdellatif et al. [75] to validate whether the remaining contributors are human. We manually examine the GitHub profiles of 54 users identified as bots by BotHunter and observed that 53 identified bots are false positives (i.e., they are not bots) and BotHunter identified one bot (i.e., sklearn-ci).

3.2.2 Cleaning Noises in Pull Request Data

To extract contributor features related to pull request acceptance, such as the number of merged pull requests and the pull request approval ratio for a contributor, we need to determine whether a pull request was accepted. We use the pull request close status on Github to determine its acceptance, where accepted pull requests are set to 'Merged' status and the rejected pull requests are set to 'Closed'. However, we observe that PyTorch uses different strategies to indicate PR approval over time. Before July 2018, accepted pull requests in Pytorch are set to 'Merged' status. From May 2019 onward, most merged pull requests are set to 'closed' status but a 'Merged' label is assigned. However, during certain periods, few pull requests receive either the 'Merged' status or the label despite being accepted. To address these false negatives,

we adopt the approach proposed by Bertoncello et al. [76], which reports that some projects include a commit SHA in the close event of merged pull requests and we find it applies to Pytorch as well. Therefore, we combine three methods to identify merged pull requests in PyTorch: GitHub's 'Merged' status, the 'Merged' label assigned by maintainers, and pull requests closed with a commit SHA. A pull request identified by any of these methods is considered merged. Appendix A.1 provides more details on PyTorch's pull request approval history.

3.3 Extracting Contributor Features

We extract 11,949 human contributors from the subject projects, where the cross-project contributors are treated as separate individuals in each project to account for potential variations in their behaviors across projects. For each contributor, we extract contributor features from their traces in the project repository and their GitHub profile page to capture their OSS activities and contributions. The extracted contributor features can be grouped into four categories: *(i)* personal attributes, capturing contributors' publicly available personal information defining their identity in the OSS community, such as their timezone and number of GitHub followers; *(ii)* commit, including measures of code change and non-code change efforts (e.g., documentation) of a contributor; *(iii)* issue, which includes activities and contributions related to issue reports, such as the number of issues raised or solved by a contributor; and lastly *(iv)* pull request, which involves activities and contributions related to pull requests, such as the number of pull requests submitted or reviewed by a contributor. In total, we extract 32 contributor features. A complete list of the contributor features is presented in Table 3.2, 3.3, 3.4, and 3.5, along with their definitions and rationales.

Table 3.2: List of contributor features – Personal Attributes

Feature	Description (D) & Rationale (R)	Reference
Timezone	**D:** The primary timezone of a developer (i.e., the most frequent timezone where a contributor makes commits). In this study, we define the timezone range of -13 to -2 as Americas, -1 to 3 as Europe/Africa, and 4 to 12 as Asia. **R:** Reveal the global distribution of ML contributors and how it affects collaboration patterns across different regions.	[27, 77]
Worktime	**D:** The primary work time of a developer (i.e., the most frequent commit time of a contributor). Normal work hours are defined as 8h to 18h in this study. **R:** Reveals when contributors are most active and their working habits (i.e., whether a contributor would like to contribute to OSS in a work-like manner).	[28, 77]
Duration	**D:** Number of days a developer stays in a project (number of days between the first and the last activity of a contributor). **R:** Shows a developer's experience with a project.	[5, 77]
Number of followers	**D:** Number of GitHub followers a developer has. **R:** Reflects the contributor's social influence within the OSS community.	[9, 22, 26]
Number of collaborated developers	**D:** Number of developers in the project a developer has collaborated with. Collaboration occurs when a developer interacts with another by commenting on their pull request or issue report, reviewing their pull request, or when the other developer does the same in return. **R:** Reflects the developer's collaborative skills and social influence in the project.	
Number of authored files	**D:** The number of files in the project repository a developer has made changes to. **R:** Reflects the breadth and scope of the developer's authorship (i.e., whether focusing on specific areas or working on a wide range of areas).	[5, 20]
Number of programming languages	**D:** The number of programming languages a developer has used in their code contributions. **R:** Reflects the contributor's technical expertise across different components of ML frameworks. For example, proficiency in multiple languages, such as C++ for low-level operations and Python for high-level APIs, shows the contributor's ability to work across multiple ML components.	[26]

Table 3.3: List of contributor features – Commit

Feature	Description (D) & Rationale (R)	Reference
Number of commits	**D:** The number of commits submitted by a developer. **R:** Measures a contributor's cumulative effort towards modifying the project codebase.	[5, 9, 25], [22, 77, 78]
Commit rate	**D:** Number of commits / Duration **R:** Reflects the intensity and frequency of a contributor's commit activity, as Montandon et al. [25] highlight that both the frequency and breadth of changes are valuable in assessing a developer's expertise.	[25, 79]
Number of code change commits	**D:** The number of commits containing changes to source code files. **R:** Measures the developer's cumulative efforts to improve the project's functionalities (e.g., add features, and fix bugs).	
Code commit rate	**D:** The number of code change commits / Duration **R:** Measures the frequency of a contributor's efforts to improve the project's functionalities.	
Number of non-code change commits	**D:** The number of commits made by a developer that do not include changes to source code files (e.g., updating documentation, tutorials, and build files). **R:** Measures the developer's cumulative non-coding efforts.	[80]
Non-code change commit rate	**D:** The number of non-code commits / Duration **R:** Measures the frequency of a contributor's efforts to make non-code changes.	
Code contribution	**D:** Total number of lines of code added or deleted by a developer. **R:** Provides a finer-grained measure of a developer's coding effort compared to commits, which capture the magnitude of code changes more precisely.	[20, 26, 78]
Code contribution rate	**D:** Code contribution / Duration **R:** Measures the frequency of a developer's code changes at LOC level.	
Code contribution density	**D:** Average code churn per commit. **R:** Reflects the magnitude of changes made by a contributor. Differentiates minor updates (e.g., small fixes or tweaks) and significant development efforts (e.g., adding new features or refactoring large portions of code).	[25, 77]

Table 3.4: List of contributor features – Issue

Feature	Description (D) & Rationale (R)	Reference
Number of issues	**D:** Number of issue reports raised by a developer. **R:** Measures a contributor's contribution in identifying problems, suggesting improvements, and proposing new features for the project.	[9, 16, 22]
Number of issues participated	**D:** The number of issue reports raised by other developers that a developer has participated in. Examples of participating activities can be commenting, triggering events, triaging, or being assigned to resolve the issue. This information is obtained from the list of issue participants using the GitHub REST API. **R:** Reveals a developer's collaborative skills and community involvement through issue report engagement.	[22]
Average issue comments received	**D:** The average number of comments received on a developer's issue reports. **R:** Reflects the importance or complexity of the issue reports raised by a developer. More important or complex issues typically attract more comments.	[9]
Number of issues solved	**D:** Number of issue reports the developer has resolved. To get this value, we first parse the commit message of all commits and the title and description of pull requests made in the studied time period. If the commit message or pull request contains keywords such as "fix," "resolve," "address," "close" or "solve" together with a valid issue ID in the same line, this commit is considered solving the issue and the author of this commit is considered the one who solves the issue. Note that the issue must be raised before the time of the commit is made. Then, we use the same method to find issue IDs from pull request messages. This feature specifically captures issues resolved through commits and does not include those resolved through discussions in the issue report alone. **R:** Reveals a developer's code contribution to fixing bugs and addressing project problems through their commits.	[23, 81, 82]
Issue contribution	**D:** Total number of issues raised or issues solved by a developer. **R:** Reflects a developer's collaborative efforts in identifying and resolving issues.	[21]
Issue contribution rate	**D:** Issue Contribution / Duration **R:** Measures the frequency of a developer's contribution to identifying and resolving issues.	
Issue solving ratio	**D:** Issue solved / Number of issues participated **R:** Reflects the efficiency of a contributor in solving the issues they participated in.	
Issue solving rate	**D:** Issue solved / Duration **R:** Measures the frequency of a developer's contribution to resolving issues.	

Table 3.5: List of contributor features – Pull Request

Feature	**Description (D) & Rationale (R)**	**Reference**
Number of pull requests	**D:** The number of pull requests submitted by a developer. **R:** Pull requests capture a contributor's efforts to change the codebase at a higher abstraction level than commits, as they often contain multiple commits for the same change and involve review, discussion, and collaboration. The total number of pull requests reflects the amount of reviewed changesets the developer has proposed to the project.	[9, 16, 22], [76]
Number of PR participated	**D:** The number of pull requests submitted by other developers that a developer has participated in. Examples of participating activities can be commenting, triggering events, triaging, or reviewing. This information is obtained from the list of PR participants using the GitHub REST API. **R:** Reveals a developer's collaborative skills and their willingness to contribute to the quality of others' code contributions.	[22, 80]
Number of PR reviewed	**D:** The number of pull requests reviewed by a developer. **R:** Measures a contributor's code review contributions.	[16]
Number of PR merged	**D:** The number of pull requests submitted by a developer being merged. **R:** Measures the developer's accepted contributions to the codebase, which reflect the amount of high-quality and project-aligned changes.	[9]
PR approval ratio	**D:** Number of PR merged / Number of pull requests **R:** Assesses the quality and alignment of a contributor's work with project standards.	
Average PR comments received	**D:** The average number of comments received on a developer's pull requests. **R:** Reflects the importance or complexity of the code changes made by a developer. More important or complex code changes typically require more discussions and revisions before they can be merged.	[9]
PR contribution	**D:** Number of pull requests submitted by a developer that have been merged and the number of pull requests reviewed by the developer. **R:** PR contribution reflects a developer's direct impact on the quality of the codebase, as merged pull requests introduce changes, and reviewing them ensures the changes meet the project's quality standards.	
PR contribution rate	**D:** PR contribution / Duration **R:** Reflects how frequently developers contribute to or review code changes.	

3.4 Correlation and Redundancy Analysis

Considering that correlated features can be expressed by each other and redundant features can be expressed by other features, highly correlated and redundant features blur the importance of the features to our analysis and result in unnecessary computations. Therefore, we conduct a correlation and redundancy analysis to remove the highly-correlated and redundant features.

- **Correlation Analysis:** We find that contributor features do not follow a normal distribution, thus we use Spearman Rank Correlation to conduct the correlation analysis [83]. Two features with a correlation coefficient higher than 0.7 are considered highly correlated [84]. For each pair of highly correlated features, we keep only one in the candidate list and remove the other one. The result of correlation analysis is shown in Figure 3.2. Ten features (i.e., Code Change Commits, Code Contribution, Non-Code Change Commit Rate, Average Issue Comments Received, Issue Solving Ratio, Issue Solving Rate, Issue Contribution, PR Merged, PR Contribution, and PR Contribution Rate) are removed and 22 features remain.

- **Redundancy Analysis:** R-squared is a measure that indicates how much variance of a variable can be explained by other variables. We use a R-squared cut-off at 0.9 to identify the redundant features [85] and find that there is no redundant feature in our data. As a result, 22 uncorrelated and non-redundant features remain.

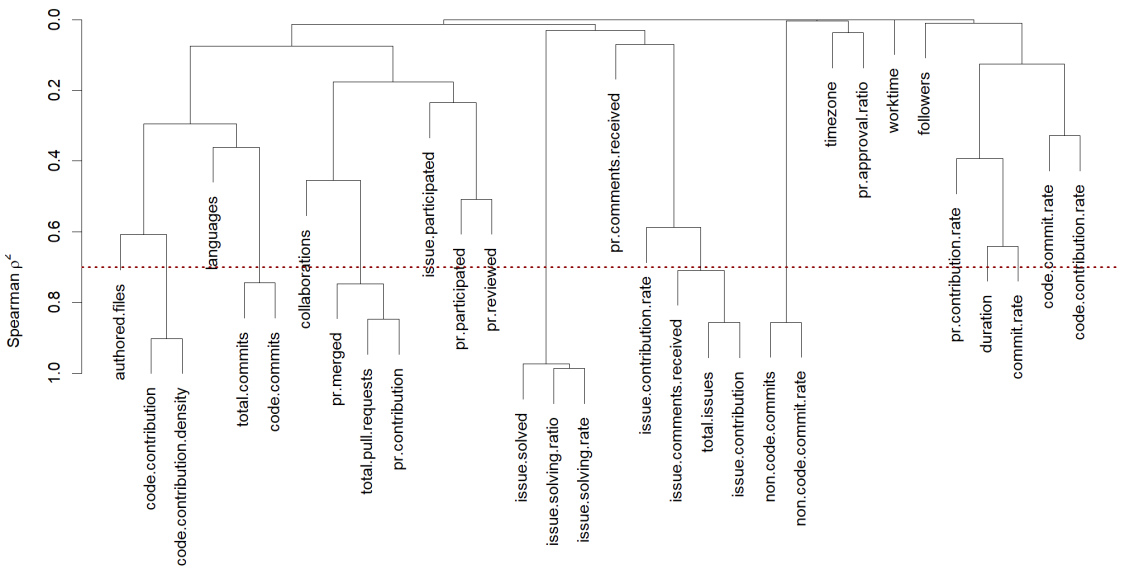


Figure 3.2: Spearman correlation analysis of contributor features, presented in a hierarchical structure. Correlated feature pairs are below the red line.

3.5 Identifying Contributor Profiles

We aim to study the characteristics of contributors' activities within open-source ML projects. To achieve this, we construct contributor profiles to categorize and capture common characteristics among groups of contributors with similar behaviors. We select the contributor features that capture the key aspects of OSS contributor behaviors and use clustering algorithms on the selected features to identify groups of contributors with similar behaviors and construct their profiles accordingly. The detailed approach involves the following four steps:

Step 1: Select contributor features for clustering. To group contributors

with similar behaviors, we select four contributor features that capture four key aspects of ML OSS contributors' behaviors to conduct the clustering. Inspired by the work of [77], we select worktime, number of commits, and code contribution density to capture contributors' fundamental behaviors. Although their study focuses on software refactoring within Apache projects, these three features are broadly applicable for capturing the key aspects of OSS contributors' behaviors regardless of the project domain. Therefore, we adapt these features to fit our objectives of understanding contributors' behavior and technical engagement across OSS ML projects. In addition, we include the number of programming languages to capture the technical expertise of ML contributors. The four selected features and the rationale behind them are as follows:

- ***Worktime*** to capture the contributor's working habit (i.e., whether one would like to contribute to OSS in a work-like manner),
- ***Number of Commits*** to capture contributor's contribution volume,
- ***Code Contribution Density*** to capture one's contributing style (i.e., whether one would like to make small changes each time or make big changes, this also reflects the complexity of one's contribution),
- ***Number of Programming Languages*** to capture the contributor's technical expertise in ML framework projects. Proficiency in multiple programming languages indicates expertise across various components of ML development. For instance, the implementation of tensors, ML operations, and computational graph execution usually utilizes C++, whereas high-level APIs are commonly written in Python.

Step 2: Normalize the selected contributor features. Clustering algorithms often use distance metrics to group similar data points. When features have different scales, those with larger scales can dominate the clustering process. Therefore, we normalize each selected contributor feature to ensure that all features contribute equally to the clustering. We apply Min-Max normalization to scale each selected feature across all contributors to a range of 0 to 1, while preserving the relative feature relationships between contributors, as shown in Equation 3.1.

$$\widehat{x_i} = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3.1}$$

where $\widehat{x_i}$ represents the normalized value of the ith contributor for a feature, x_i represents the original value of the ith contributor for the feature, $min(x)$ and $max(x)$ represent the minimum and maximum value of the feature across all contributors, $i \in \{1, 2, 3, \ldots, 7640\}$, and a feature $\in \{$*Worktime*, *Number of Commits*, *Code Contribution Density*, *Number of Programming Languages*$\}$.

Step 3: Select the clustering algorithm. We examine 8 clustering algorithms in the Python scikit-learn package [86], namely, Kmeans, Affinity Propagation, Mean Shift, Spectral Clustering, Hierarchical Clustering, DBSCAN, OPTICS, and BIRCH, to cluster ML contributors with the four normalized features. The effectiveness of these clustering algorithms is evaluated using the Silhouette score, which is a metric to assess the quality of data point groupings [86]. Silhouette scores range from -1 to 1, where a score of 1 indicates an optimal clustering and -1 indicates the worst. Scores close to 0 indicate overlapping clusters and negative values indicate that a data point has been incorrectly assigned to a cluster since it is more similar to a different cluster.

For the clustering algorithms requiring a predefined number of clusters (i.e., Kmeans,

Spectral Clustering, Hierarchical Clustering, and BIRCH), we experiment with the number of clusters range from 1 to 10 and select the optimal number of clusters with the highest Silhouette score. For the rest of the clustering algorithms, we conduct a gradient search to determine the optimal parameter set with the highest Silhouette score.

Table 3.6: Eight clustering algorithms and the associated evaluation results.

Clustering Algorithms	Kmeans	Affinity Propagation	Mean Shift	Spectral Clustering	Hierarchical Clustering	DBSCAN	OPTICS	BIRCH
Number of Clusters	3	5,542	4	4	3	12	2	4
Silhouette Score	0.559	0.053	0.564	**0.565**	0.535	0.522	0.527	0.542

Step 4: Identify contributor profiles. Based on the clustering algorithm evaluation results shown in Table 3.6, the clusters produced by Spectral Clustering have the highest Silhouette Score of 0.565, indicating the optimal clustering result among all clustering algorithms. Therefore, we identify our contributor profiles from the four clusters generated by Spectral Clustering. Figure 3.3 shows a summary of the profiles in terms of each clustering feature and we name each profile accordingly. As shown in Figure 3.3(b), 3.3(c), and 3.3(d), the two clusters represented in green and orange have higher total commits, code contribution density, and number of used programming languages compared to the other two clusters, so we name them *Core* and *Peripheral* contributors respectively. *Core* and *Peripheral* contributors can be further divided into *Workhour* and *Afterhour* subgroups based on the time they used to make OSS contributions as shown in 3.3(a). We summarize the four profiles in the following:

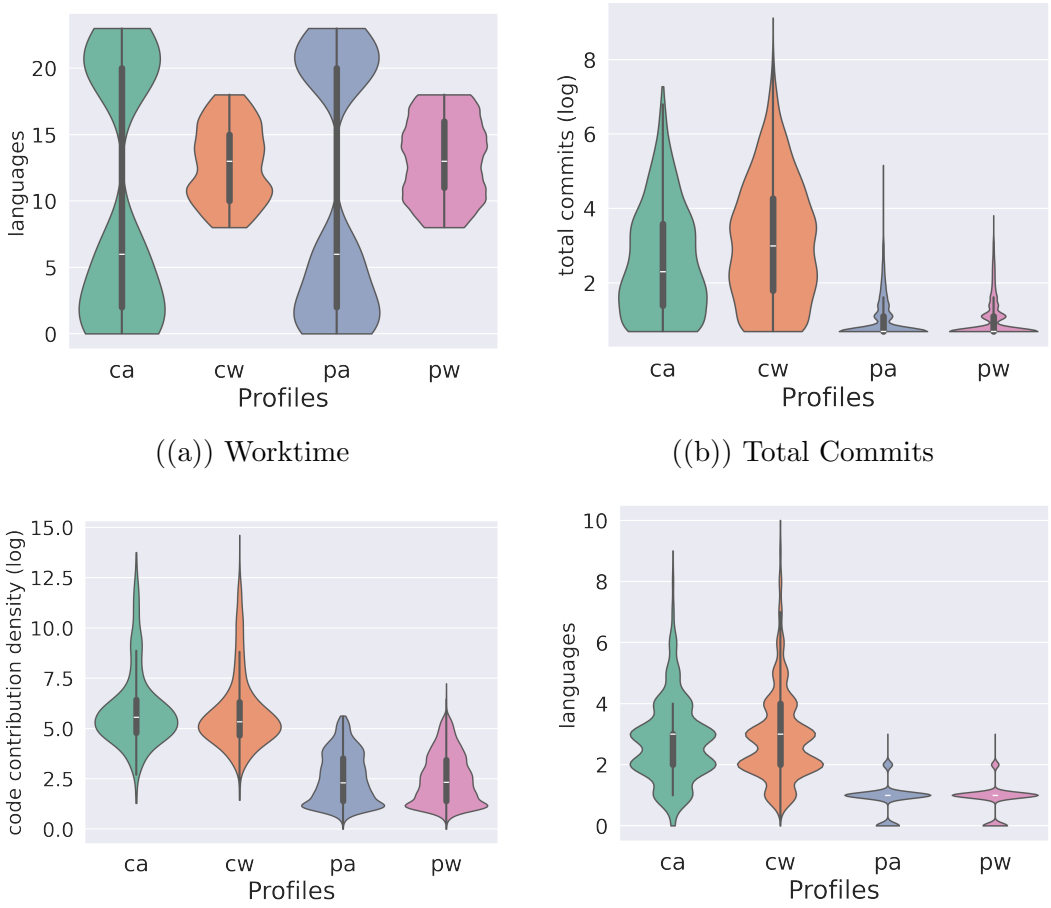
- **Core-Afterhour profile** contains 753 (6.3%) contributors. This profile has

a larger number of commits, code contribution density, and programming languages used in comparison to *peripheral* contributors, with a median of 9 commits, 260 LOC per commit, and 3 programming languages. Contributors in this profile tend to make contributions outside normal working hours (i.e., outside 8 AM to 6 PM of their local time).

- **Core-Workhour profile** contains 2,016 (16.9%) contributors. This profile is similar to *Core-Afterhour* in terms of the number of commits, code contribution density, and programming languages, with a median of 19 commits, 207 LOC, and 3 programming languages. Contributors in this profile make contributions within normal working hours.

- **Peripheral-Afterhour profile** contains 3,860 (32.3%) contributors, characterized by few commits, code contribution density, and programming languages, with a median of 1 commit, 9 LOC, and 1 programming language. Contributors within this profile make contributions outside normal working hours.

- **Peripheral-Workhour profile** contains 5,320 (44.5%) contributors. They are similar to *Peripheral-Afterhour* contributors in terms of few commits, code contribution density, and programming languages, with a median of 1 commit, 9 LOC, and 1 programming language. However, contributors in this profile tend to contribute during normal working hours.



((a)) Worktime

((b)) Total Commits

((c)) Code Contribution Density

((d)) Programming Languages

Figure 3.3: Summary of four contributor profiles ('ca' refers to Core-Afterhour, 'cw' refers to Core-Workhour, 'pw' refers to Peripheral-Workhour, and 'pa' refers to Peripheral-Afterhour).

3.6 Contributor OSS Engagement

To gain a further understanding of contributors' engagement in various OSS activities, we study their OSS engagement from four aspects: workload composition, work preferences, technical importance, and ML-specific contributions.

3.6.1 Identifying Workload Composition Patterns

Workload composition reflects how contributors allocate their effort across key OSS routines, including committing, raising issue reports, participating in issue discussions, participating in pull request discussions, and code reviews within a specific period. We identify workload composition patterns to capture the common workload compositions and study the distribution of contributors with similar workload compositions as well as their impact on the subject project. Our approach to identifying contributor workload composition patterns involves the following four steps:

Step 1: Determine the length of the period to analyze contributors' workload composition. A contributor can be assigned to different tasks at different time periods. Therefore, contributors' workload composition is often dynamic and can evolve over time. We aim to capture their temporary workload compositions that are the focus of a contributor within a time interval, assuming the workload composition remains stable over brief periods. Specifically, we divide the lifespan of a project into consecutive 90-day periods. For example, TensorFlow's first period spans from 2015/11/07 to 2016/02/05, and the second from 2016/02/05 to 2016/05/05. For each period, we count the increase in stars and forks, which serve as dependent variables for modeling project popularity (normalized per project). For each period, we also build workload composition vectors and compute work preference features for all active contributors. The independent variables are the aggregated workload composition and work preference features of the active contributors in each period. This is because the project popularity is influenced not by individual behavior alone but by the collective behavior of all contributors during a given period. This way,

we model the relationship between contributor activity and the increase of project popularity using consistent time units.

We further conduct a sensitivity analysis on 30, 60, 90, 120, and 180 days to identify the optimal period length and verify the stability of a contributor's workload composition. We observe that shorter periods, such as 30 and 60 days, lack precision in representing contributors' workload compositions, due to the potential of little activities within short periods. For example, suppose a contributor only makes one commit in 30 days but engages in pull request discussions in the following month (i.e., 30 days), we may observe an exclusive focus on committing for the first period and an exclusive focus on pull request discussions for the subsequent period. We have a total of 264 periods from all 8 subject projects with 90-day periods. Conversely, long time intervals, such as 120 and 180 days, yield too few data points for our analysis, particularly in RQ3, where we model the relationship between contributor workload compositions and project popularity (e.g., star rating and number of forks). We use the increase in project popularity within each period as the dependent variable and the aggregated workload composition-related feature of all active contributors within the period as independent variables, as project popularity is influenced by the collective behavior of all contributors, not by any single individual. Therefore, the number of periods determines the number of data points for building the model. Long period length would reduce the number of periods and result in insufficient data points for the model to converge.

Therefore, for each subject project, we segment project activities into 90-day intervals from the beginning of the project's lifetime to the date of our data collection and capture the workload composition of active contributors in each period. In the

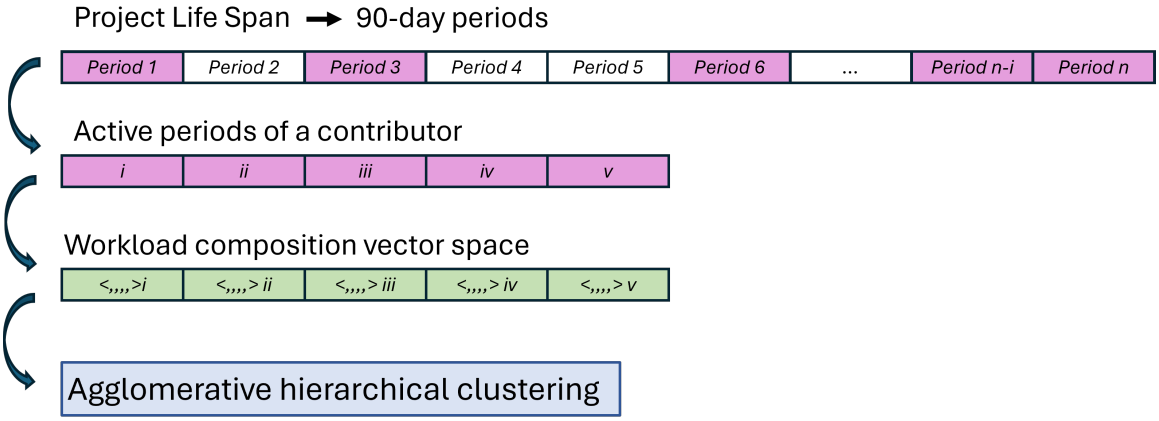


Figure 3.4: Steps for extracting periods and identifying workload composition patterns.

rest of this paper, the term '***period***' refers to these 90-day project intervals.

Step 2: Construct contributor workload composition vector space. We create a 5-dimensional vector space to represent the contributors' workload composition within a period. The five dimensions include the count of commits, raised issue reports, issue comments, pull request comments, and code reviews, which represent the key components of OSS project routines. For each period of a project, we create a workload composition vector for each active contributor. Recognizing that the absolute counts of OSS activities may not accurately reflect the actual focus of contributions (e.g., conducting 2 code reviews might be a considerable contribution while making 2 issue comments might be a small contribution), we normalize each dimension of the workload composition vector across contributors. The normalization ensures that each dimension in the vector represents the relative contribution to an OSS activity among contributors and the vector can reflect the actual workload allocations of contributors. Furthermore, the relative contribution can vary between projects and even within the same project over different periods. Hence, for each period in each

subject project, we apply Min-Max normalization to normalize each dimension of the workload composition vectors across all active contributors. The normalized vector space ensures safe comparisons between vector dimensions and vectors from different projects or periods. As a result, we build 41,628 workload composition vectors from 11,949 contributors across a total of 264 project periods from our 8 subject projects.

Step 3: Identify workload composition patterns. We employ agglomerative hierarchical clustering [86] to identify the workload composition patterns. Agglomerative hierarchical clustering begins by treating each vector as an individual cluster and successively merges smaller clusters with the highest similarity until all data points belong to a single cluster or a predefined dissimilarity threshold is reached. This iterative process naturally forms cohesive clusters and can produce a dendrogram, which is a hierarchical tree-like structure of datapoints and can be cut at different heights (i.e., dissimilarity threshold) to obtain different numbers of clusters. Compared to other clustering algorithms discussed in Section 3.5, we select hierarchical clustering to identify workload composition patterns because the dendrogram enables us to visualize and explore the similarity of workload compositions at different levels of granularity, thereby determining the the level of similarity of the resulting patterns.

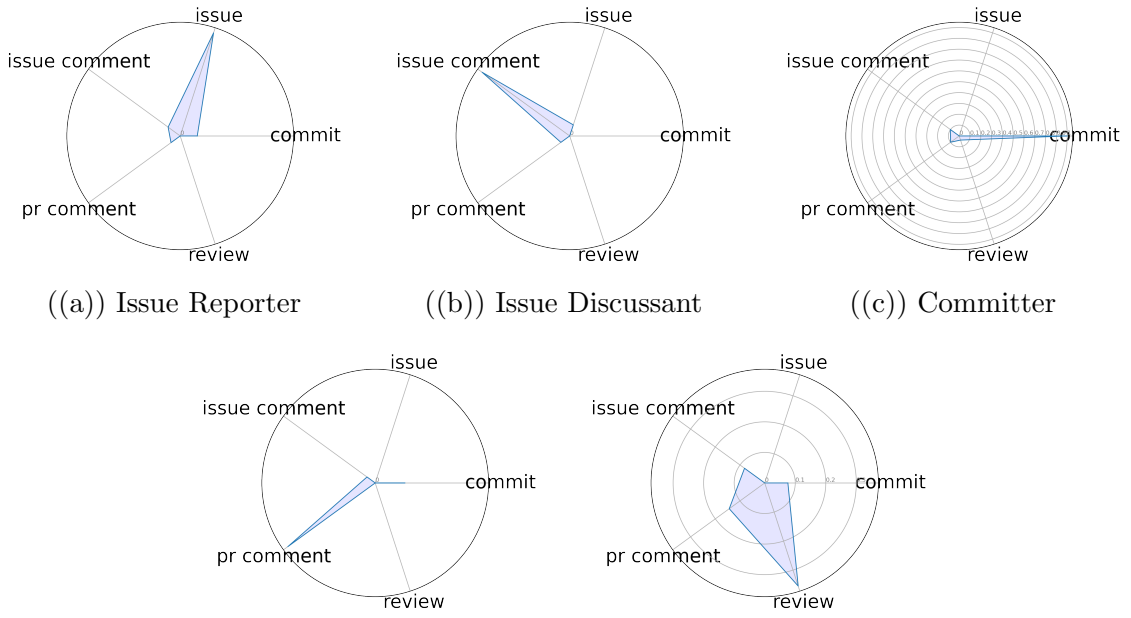
We use cosine similarity as the similarity measurement for workload composition vectors, as it gauges the similarity of vector dimensions irrespective of the vector scale. This aligns with our goal of identifying contributors with a similar focus across different types of OSS contributions, regardless of their amount of contribution.

We first compute the pairwise cosine similarity of every two workload composition vectors to build a distance matrix. Then, we apply agglomerative hierarchical clustering on the distance matrix to group similar vectors and plot the dendrogram

of the clustering. We manually inspect the clusters generated from different cut-off dissimilarity thresholds and calculate the Silhouette scores for the resulting clusters to determine the optimal number of clusters and identify workload composition patterns accordingly.

Step 4: Summarize workload composition patterns. We find five clusters as the optimal number of clusters at a cut-off threshold of 0.9, which also achieves the highest Silhouette score of 0.562 compared to other numbers of clusters. From these clusters, we identify five workload composition patterns. Figure 3.5 shows the centroid workload composition vector in each cluster, which is the point with the highest cosine similarity with other points in the cluster. We summarize the work focus of each cluster according to the centroid behavior in the following:

- **Pattern 1: Issue Reporter** accounts for 19.8% of occurrences. Contributors in this pattern focus on raising issue reports and participate minimally in other activities.
- **Pattern 2: Issue Discussant** is the second most common pattern comprising 16.1% of occurrences. They focus on discussing issue reports while infrequently engaging in other types of contributions.
- **Pattern 3: Committer** is the most common pattern and comprise 40.9% of occurrences. They only focus on making commits and solely participate in the other four activities.
- **Pattern 4: PR Discussant** comprise 15.5% of occurrences. They mainly focus on participating in pull request discussions, with occasional involvement in issue discussions and making code commits.



((a)) Issue Reporter ((b)) Issue Discussant ((c)) Committer

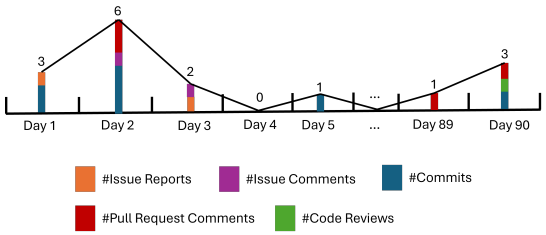
((d)) PR Discussant ((e)) Code Reviewer

Figure 3.5: Workload composition vector of the centroid of each workload composition pattern.

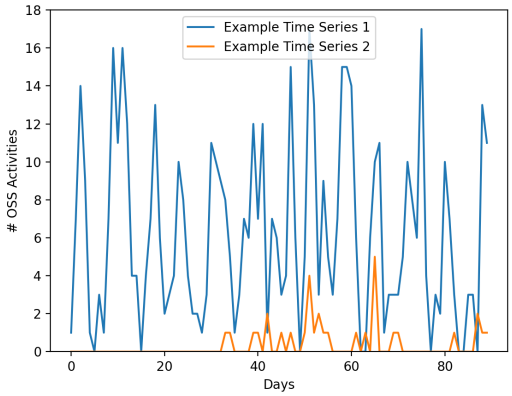
- **Pattern 5: Code Reviewer** is the least common pattern comprising 7.7% of occurrences. Contributors in this pattern primarily focus on code reviews but occasionally make commits and participate in collaborative activities (i.e., pull request and issue discussions).

3.6.2 Extracting Work Preference Features

While workload composition provides a static view of contributors' focus of OSS activities within a period of time, work preference features capture their contribution



((a)) Constructing the OSS activity time series.



((b)) Example of a complex and a less fluctuated time series.

Figure 3.6: An example of contributors' OSS activity time series within a 90-day period.

dynamics and preferences in that period, such as whether they make consistent contributions (e.g., making one commit daily) versus peak contributions (e.g., making a high number of commits on a single day and remaining inactive afterward), as well as their preference on making specific types of contributions versus an even contribution across various types.

As shown in Figure 3.6(a), we count a contributor's daily activity in terms of the five OSS activity dimensions (i.e., the number of issue reports, issue comments, commits, pull request comments, and code reviews) each day and create a time series for each contributor over a 90-day period (as discussed in 3.6.1). We use the Python *tsfresh* package [87] to extract time series features including *binned_entropy*, *c3*, *number_cwt_peaks*, *longest_strike_above_mean*, and *longest_strike_below_mean*. These time series features measure a contributor's contribution dynamics. Figure 3.6(b) presents examples of contributors' OSS activities time series with different contribution dynamics, where example time series 1 exhibits complex dynamics and example time

series 2 exhibits comparatively fewer fluctuations. We also extract two features, *diverse* and *balance*, to measure the breadth of OSS contribution types made by a contributor and the degree to which they distribute their efforts evenly across various types within each period. We consider these 9 extracted features as contributor work preference features. Detailed descriptions and implications of the work preference features are outlined in Table 3.7.

Table 3.7: Extracted features of contributor work preferences.

Contributor Work Preference Features	**Description**
binned_entropy(10)	Binned entropy measures the complexity of the sequence. A higher binned entropy indicates developers have more complex contribution dynamics and make less constant contributions.
c3(1), c3(2), c3(3)	C3 statistics measures the degree of nonlinearity or complexity in a time series by calculating the correlation between the original time series and its lag of 1, 2, 3 respectively. Higher C3 statistics indicate developers have more complex contribution dynamics and make less constant contributions.
number_cwt_peaks	Number of peaks of contributions. A higher value indicates developers make more peak contributions.
longest_strike_above_mean	The ratio of the longest subsequences above mean. A higher value indicates more consecutive days of high contribution.
longest_strike_below_mean	The ratio of the longest subsequences below mean. A higher value indicates more consecutive days of low contribution.
diverse	The number of different types of contributions a developer has made.
balance	The inverse of the variance of a developer's normalized contribution on the five OSS activities. A higher value indicates developers make a more even amount of different types of contributions.

3.6.3 Evaluating Contributor Technical Importance

The technical importance of a contributor can be measured using the importance of the commits made by the contributor. The importance of a commit can be measured with the eigenvector centrality of source code files modified by the commit. Eigenvector centrality measures the importance of a node within a network. In our case, the software repository can be viewed as a network, with the folders and source code

files as nodes, and their containment relationships as edges. Therefore, the eigenvector centrality of a source code file measures the importance of the file within the file network of a repository. Contributors who frequently make changes to important files can be considered as possessing higher technical importance to the project.

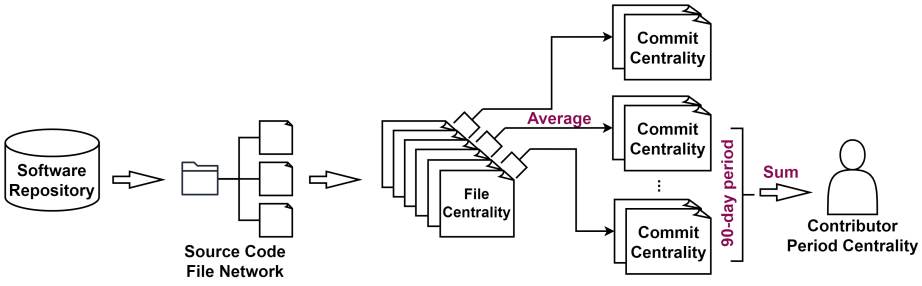


Figure 3.7: Measure contributor technical importance.

Figure 3.7 shows our approach for measuring contributor technical importance. Following a similar methodology as proposed by Yue et al. [11], we first construct a network with folders and source files as nodes and the relationship of a folder containing a file as edges, then calculate the eigenvector centrality of each source file, as *file centrality*. The *commit centrality* is calculated by averaging the eigenvector centrality of the source files that have been changed in the commit. Higher *commit centrality* means that the commit makes changes to more influential files in the repository and can have a greater impact on the codebase. We sum the *commit centrality* of all commits made by a contributor within a period to calculate the *period centrality*, which measures the technical importance of a contributor within a period, considering both the intensity and importance of their commits. We introduce four metrics to quantify the technical importance of a contributor:

- ***max_commit_centrality***: The highest commit centrality among all commits made by a contributor.

- ***max_centrality_day***: The number of days taken by a contributor to make the commit with the highest centrality.
- ***max_period_centrality***: The highest period centrality among all active periods of a contributor.
- ***max_centrality_period***: The number of periods taken by a contributor to achieve the highest period centrality.

3.6.4 Qualitative Analysis for ML-Specific Contribution

To further understand the nature of OSS ML contributors' code contributions beyond the volume of code changes, we manually analyze the content of their pull requests, focusing specifically on the ML-specific aspects they contribute to. Pull requests are the primary approach through which contributors modify the project's codebase, which provides valuable context through descriptions, comments, and code review activities. This makes pull requests more informative than commits for understanding the nature of contributions. Therefore, we focus on analyzing pull requests to determine how contributors from different profiles contribute to different ML-specific aspects of the codebase. For this analysis, we adopt the taxonomy proposed by Chen et al. [88], which categorizes the architecture of a DL framework into five levels, from the highest to the lowest:

1. **User-Level API**: is high-level APIs that are designed to simplify the use of the framework for end users, including APIs for input data processing, model building, model deployment, and other user-facing utilities that facilitate the DL workflow.

2. **Graph-Level Implementation**: includes processes and functions that manage the organization and the execution of operations within a computational graph. A computational graph is a directed graph where nodes represent operations (e.g., matrix multiplication, convolution) and edges represent the flow of data (typically in the form of tensors) between these operations. This level includes graph construction, transformation, and execution.

3. **Operation Implementation**: provides detailed implementations of operations (e.g., convolution, pooling, mathematical functions) that form the nodes in computational graphs and primarily operate at the tensor level.

4. **General Utility**: are reusable functions and data structures (e.g., type conversion, padding) that support the higher-level architecture of the DL framework.

5. **Environment-Dependent Processing**: supports DL frameworks run across different hardware (e.g., GPU) and software environments by managing environment-specific processes (e.g., memory allocation).

We select a stratified random sample of 384 pull requests from a total of 130,779 pull requests across all subject projects, with a 95% statistical confidence level and a 5% margin of error for the sample size [77, 89]. Stratification is based on the four contributor profiles, with 96 pull requests randomly sampled from the pool of pull requests raised by contributors in each profile. Two independent labelers (i.e., the first two authors) manually review the title, description, discussions, and code changes in an initial set of 40 sampled pull requests, categorizing it as a contribution to one of the five ML components or as a general software development contribution unrelated to the ML-specific architecture. After reaching an agreement of labels based on the

initial set of samples, the two labelers continue to label the rest of samples. To assess inter-rater reliability, we calculate Cohen's Kappa coefficient [90], a standard metric for evaluating agreement between raters, and achieve an overall Kappa value of 0.876. Discrepancies between labelers are resolved through discussion, where both labelers compare interpretations and reach consensus on the final label of each pull request.

Chapter 4

Experiment Results

4.1 RQ1: What are the characteristics of each contributor profile?

4.1.1 Motivation

We identify 4 contributor profiles based on multiple factors including work time, the number of commits, code contribution density, and the number of used programming languages. In this research question, we further delve into the important features of each profile and explore the differences between the profiles as well as their shared attributes. Understanding contributor profiles can equip project managers and maintainers with the knowledge to engage the contributors of different profiles in the project effectively and support tactics to accommodate the varied spectrum of contributors.

4.1.2 Approach

To gain a further understanding of contributor profiles, we build four binary logistic

regression models to identify the significant contributor features associated with each of the four contributor profiles. Our detailed approach includes the following three aspects:

Model Construction: To build the model for each contributor profile, we exclude the four features used for identifying the profiles (i.e., Worktime, Number of Commits, Code Contribution Density, and Number of Programming Languages) and use the remaining 18 non-correlated and non-redundant contributor features (as described in Section 3.4) as independent variables. We establish a binary dependent variable by labeling contributors of the target profile (e.g., *Core-Afterhour*) as '1' and contributors of the remaining three profiles (i.e., *Core-Workhour*, *Peripheral-AFterhour*, and *Core-Workhour*) as '0'. Therefore, we create four models. Binary dependent variable allows us to explore the differences between the target profile and the rest, as well as to identify the important features defining the target profile. To avoid bias caused by the imbalanced labels, we construct a balanced dataset by using an equal number of data points for both '1' and '0' labels. We include all data points labeled '1' and a corresponding number of randomly selected data points labeled '0'. When constructing the logistic regression model, we build the base model using *glm* function in R [91], and subsequently employ the *step* function to build step models, which iteratively eliminate the features that are not significant to the model.

Model Performance Evaluation: We evaluate our model's effectiveness using the Area Under the Curve (AUC) and Matthews Correlation Coefficient (MCC) metric. The AUC, representing the space beneath the Receiver Operating Characteristic (ROC) curve, quantifies the capability of a binary classifier to differentiate between

classes. AUC values range from 0 to 1, with higher values signifying better performance. Specifically, an AUC of 0.5 indicates a random classifier, while an AUC of 1 denotes a perfect classifier [92]. MCC measures the correlation between the binary predictions and actual values, considering all four categories in the confusion matrix (i.e., true positives, false negatives, true negatives, and false positives), which is a more reliable metric than the F1 score and accuracy, especially on imbalanced datasets [93]. MCC values range from -1 to 1, where 0 indicates random guessing, 1 represents a perfect classifier, and -1 represents complete misclassification.

To further evaluate the robustness of our models, we apply the bootstrapping evaluation technique, which helps mitigate the potential bias introduced by imbalanced data or by data points not included in the training set. For each model, we randomly sample 1,000 data points from both the '0' and '1' labels (500 data points for Model 1 due to the smaller size of the Core-Afterhour profile, which has 763 contributors). The model is then used to predict the labels of the combined 2,000 data points (1,000 for Model 1). This process is repeated 1000 times for each model, and we compute the mean accuracy, AUC, and MCC to evaluate their overall performance.

Model Result Analysis: To identify the significant features of each model and its corresponding profile, we employ Wald statistics ($\tilde{\chi}^2$) to estimate the feature importance and statistical significance (p-values) of each contributor feature, as Wald statistics are often used in regression analysis to assess the significance of individual coefficients or parameters in a regression model [94]. A higher $\tilde{\chi}^2$ value indicates a larger impact of the particular feature on the discriminatory capability of the model [95, 96]. We utilize the *ANOVA* function in R to compute $\tilde{\chi}^2$ values and the corresponding p-values. We further calculate the percentage of $\tilde{\chi}^2$ of each feature

relative to the sum of $\tilde{\chi}^2$ values to show the proportion to the overall importance for each feature [97]. The significant features for each profile are marked with asterisks in Table 4.2, with more asterisks indicating higher statistical significance levels.

We also analyze the coefficients of each feature in the logistic regression models. Positive coefficients imply a direct correlation with the dependent variable, suggesting that as the feature increases, the chances of a contributor being classified into the profile labeled as '1' also tend to increase. Conversely, negative coefficients imply an inverse relationship. The upward ($\nearrow$) and downward ($\searrow$) arrows in Table 4.2 denote the positive and negative correlations.

To validate the findings from the model results, we apply two-sided Mann-Whitney U test [98] to compare contributor features across contributors in *core* and *peripheral* profiles with the null hypothesis $H0_1$, as well as compare their *workhour* and *afterhour* subgroups with the null hypothesis $H0_2$ and $H0_3$ respectively. We apply Cliff's delta [99] to estimate the effect sizes.

- $H0_1$*: The distributions of the given contributor feature are the same across core contributors and peripheral contributors.*
- $H0_2$*: The distributions of the given contributor feature are the same across Core-Afterhour and Core-Workhour contributors.*
- $H0_3$*: The distributions of the given contributor feature are the same across Peripheral-Afterhour and Peripheral-Workhour contributors.*

4.1.3 Results

Table 4.1: AUC of 4 binary logistic regression models.

Evaluation Result		**Model 1 Core-Afterhour**	**Model 2 Core-Workhour**	**Model 3 Peripheral-Afterhour**	**Model 4 Peripheral-Workhour**
Model Evaluation	**Accuracy**	0.72	0.77	0.64	0.63
	AUC	0.72	0.77	0.64	0.63
	MCC	0.45	0.53	0.28	0.26
Bootstrapping Evaluation	**Accuracy**	0.71	0.75	0.63	0.62
	AUC	0.71	0.75	0.63	0.62
	MCC	0.42	0.51	0.27	0.25

Table 4.2: Result of 4 binary logistic regression models.

Contributor Features	**Core-Afterhour**			**Core-Workhour**			**Peripheral-Afterhour**			**Peripheral-Workhour**		
	Chisq%	Signf.	Rel.	Chisq%	Signf.	Rel.	Chisq%	Signf.	Rel.	Chisq%	Signf.	Rel.
Duration	2.94	**	↗	**17.93**	***	↗	2.5	***	↘	**11.14**	***	↘
Timezone	6.91	***	↗	**25.01**	***	↘	**50.05**	***	↗	**13.78**	***	↘
Authored Files	**17.01**	***	↗	**12.36**	***	↗	**9.52**	***	↘	**27.63**	***	↘
Commit Rate				1.58	***	↘				1.99	***	↗
Code Commit Rate				1.82	***	↗	1	*	↘			
Code Contribution Rate	0.94	.	↗	0.3		↗	5.38	***	↘	0.45	.	↘
Non Code Commits	0.65		↘									
Total Issues							0.61	.	↗			
Issue Solved	1.95	*	↗	1.49	***	↘				1.71	***	↗
Issue Participated	2.03	*	↘									
Issue Contribution Rate	1.32	*	↘	0.57	*	↘	1.65	**	↗	0.39		↘
Total Pull Requests				4.39	***	↗	2.76	***	↘	3.21	***	↘
PR Reviewed	8.03	***	↘	2.63	***	↘	2.71	***	↗	5.5	***	↗
PR Approval Ratio	**12.05**	***	↘	**14.33**	***	↘	**14.89**	***	↗	1.94	***	↗
Avg PR Comments Received	**18.87**	***	↗	**10.61**	***	↗	0.95	*	↘	**14.36**	***	↘
PR Participated				0.48	*	↘				0.73	*	↗
PR Contribution Rate	7.04	***	↘	2.05	***	↘	0.54	.	↗			
Followers				0.65	*	↘						
Collaborations	**15.37**	***	↗	3.79	***	↗	3.69	***	↘	8.97	***	↘
Project Encoded	4.88	***	↘				3.76	***	↘	8.2	***	↗

Signif. codes: **0 '***' 0.001 '**' 0.01 '*' 0.05** '.' 0.1 ' ' 1

Chisq%: The importance of a feature in modeling a profile in percentage.

Rel.: ↗ stands for a positive correlation between a feature and the likelihood of being a profile, and ↘ stands for a negative correlation.

The constructed logistic regression models have the highest discrimination performance for classifying Core-Afterhour contributors. Table 4.1 presents the AUC values of the models. The models for *Core-Afterhour* and *Core-Workhour* contributors have the top AUC values of 0.72 and 0.76 respectively. Both *Peripheral-Afterhour* and *Peripheral-Workhour* profile models achieve AUC values over 0.6, indicating the effectiveness of the models in classifying the respective profiles beyond random classification. Additionally, all models yield MCC scores greater than 0, which stands for a positive correlation between the predicted and actual labels. Bootstrapping evaluations show a minimal decrease in performance (i.e., 0.01 to 0.02 for AUC and 0.01 to 0.05 for MCC), which shows the robustness of the models across our entire dataset. Additionally, statistical test results show that the null hypothesis $H0_1$ is rejected for 31 (out of 32) contributor features, which shows significant differences between *core* and *peripheral* contributors regarding the majority of contributor features. Similarly, the null hypothesis $H0_2$ is rejected for 21 contributor features and $H0_3$ is rejected for 18 contributor features, highlighting the significant differences between *Core-Afterhour* and *Core-Workhour* contributors and between *Peripheral-Afterhour* and *Peripheral-Workhour* contributors respectively. The complete list of test results is presented in Table A.1 in Appendix A.2. In the rest of the section, we describe our analysis of the logistic regression models and statistical test results, and identify the important features associated with each profile.

The common important features across all four contributor profiles include duration, authored files, average comments received on pull requests, pull requests approval ratio, and collaboration, which serve as key factors in differentiating core and peripheral contributors. As shown in Table 4.2,

both *Core* profiles are associated with longer project duration, a higher number of authored files, more comments received on their pull requests, and more collaborated contributors, while *Peripheral* profiles are associated with a higher pull request approval ratio compared to their counterparts. The Mann-Whitney U test further confirms that these five features show statistically significant differences between *Core* and *Peripheral* contributors. Referring to the rationale for these features in Table 3.2, 3.3, 3.4, and 3.5, this result shows that *Core* contributors are characterized by extensive project experience, broader scope of authorship, higher contribution complexity, and a notable social influence within the project, distinguishing them from *Peripheral* contributors. Core contributors tend to receive more comments on their pull requests but have a lower acceptance ratio, suggesting that their contributions are often more complex or influential, carrying a higher risk of broad impact. Such changes typically require more rounds of discussion, rework, or even breaking them down into smaller pieces before being ready for merging without disruptions. In contrast, contributions from *Peripheral* contributors tend to be smaller, more specific, or less controversial, which allows them to be more easily accepted. Furthermore, the difference in collaborations also indicates that *Core* contributors tend to have active involvement in mentoring activity or project management, while *Peripheral* contributors tend to focus on their specific contributions to the project without involving others.

Timezone is the main factor to differentiate workhour and afterhour contributors, while working hours have negligible common association with other contributor features. Table 4.2 shows that timezone is significant for all profiles with positive associations with two *Afterhour* profiles (i.e., *Core-Afterhour*

and *Peripheral-Afterhour*) and negative associations with two *Workhour* profiles (i.e., *Core-Workhour* and *Peripheral-Workhour*). However, the significance of time zones lies in their representation of geographical regions. We categorize contributor timezones into three regions: Americas (timezone from -12 to -2, containing 48.8% of contributors), Europe/Africa (-1 to 3, containing 32.5% contributors), and Asia (4 to 12, containing 18.7% contributors). We find that 72.3% of contributors from the Americas are *Workhour* contributors, contrasting with 45.7% from Asia and 54% from Europe/Africa. This suggests a higher inclination among Americas-located contributors towards adopting a regular job-like approach to OSS ML projects compared to other regions. *Afterhour* contributors might have flexible contribute schedules or prioritize their contributions to the project during their free time or outside their regular job commitments. It is also possible that *Afterhour* contributors from Europe/Africa or Asia attempt to align their contributions with work hours in the Americas to ensure quick responses from the majority of the development team. These findings indicate the predominant influence of contributors from the Americas in driving the selected ML projects.

Core-Workhour contributors tend to stay longer in the project and are more involved in commits and collaborative activities, while Core-Afterhour contributors tend to handle more complex tasks. Logistic regression results indicate that the two *Core* profiles exhibit similar performance across most features except for timezone. The Mann-Whitney U test reveals significant differences between two *Core* profiles on 21 features. The null hypothesis $H0_2$ is rejected for the duration, with a negative effect size, indicating that *Core-Workhour*

contributors tend to stay involved in projects longer than *Core-Afterhour* contributors. $H0_2$ is also rejected for the number of authored files and 7 commit-related features (i.e, the total number of commits, commit rate, number of code commits, code commit rate, number of non-code commits, non-code commit rate, and code contribution) with negative effect sizes. This suggests that *Core-Workhours* tend to have a boarder scope of authorship and make more frequent and extensive coding contributions compared to *Core-Afterhour* contributors. Additionally, $H0_2$ is rejected for the number of issues participated in, pull requests participated in, and pull requests reviewed with negative effect sizes. This suggests that *Core-Workhour* contributors are more actively involved in discussions in other contributors' pull requests and reviews, guiding users in resolving issues and offering feedback on pull requests. These activities indicate a strong focus on mentoring, as they involve knowledge sharing and helping other contributors align their contributions with project requirements. Conversely, $H0_2$ is rejected with positive effect sizes for code contribution density and the average number of comments received on issues and pull requests, indicating that *Core-Afterhour* contributors are more likely to report complex issues or submit pull requests with complex or influential changes, often involving larger code churns in their commits than *Core-Afterhour* contributors. Additionally, $H0_2$ is also rejected with a positive effect size for the pull request approval ratio, suggesting that despite the complexity of their contributions, *Core-Afterhour* contributors maintain a high standard of quality.

Despite sharing similar work time, commit volume, code contribution density, and proficiency in multiple programming languages, contributors within the same profile may have different preferences and areas of focus.

Logistic regression model results show that the two *core* profiles tend to be characterized by positive association with independent coding features indicating their high code contributions. Moreover, the two *peripheral* profiles tend to be characterized by positive association with issue handling (i.e., total issues, issues solved, and issue contribution rate) and collaborative coding features (i.e., the number of pull requests reviewed, pull requests participated in, and pull request approval ratio), indicating active participation in collaborative activities. However, the Mann-Whitney U test results show that *core* contributors are statistically higher in 4 of these features than *peripheral* contributors excluding the pull request approval ratio. This contradicted observation indicates that the logistic regression models can effectively leverage features related to the code contributions of *core* contributors but may overlook the characteristics of those *core* contributors who actively participate in other OSS activities.

For instance, 55.4% *Core-Afterhour* and 50.5% *Core-Workhour* contributors have raised at least one issue report. We use the Mann-Whitney U test and Cliff's delta to compare the number of pull requests and pull request approval ratio between contributors who have raised at least one issue and those who have not, for both *Core* profiles respectively. We find that contributors who have raised issue reports make significantly more pull requests (with a large effect size for both profiles) and higher pull request approval ratio (with a small effect size for *Core-Workhour* contributors and negligible for *Core-Afterhour*) compared to those who have never raised any issues. This suggests that contributors within the same profile may have different levels of engagement in specific OSS activities, potentially resulting in variations in the volume and quality of collaborative coding. Further investigation is needed to

understand the distribution of workload within each profile and how contributors engage in different OSS activities.

RQ1 Summary

Project experience, breadth of file authorship, pull requests comments received and approval ratio, and collaborations are important features common to all four contributor profiles, primarily distinguishing core from peripheral profiles. Contributors in workhour and afterhour profiles mainly differ in their geographical locations. Core-Workhour contributors tend to stay longer in the project and are more involved in commits and collaborative activities. Core-Afterhour contributors tend to handle more complex tasks. Even within the same profile, contributors exhibit varied engagements in OSS activities.

4.2 RQ2: What is the OSS engagement of each contributor profile?

4.2.1 Motivation

In RQ1, we observe that contributors within the same profile may differ in their involvement in different OSS activities. Given the broad range of open-source tasks and the limited time OSS contributors can allocate to such activities, they often prioritize certain tasks over others based on their needs and preferences. Understanding contributor workload compositions is essential to enrich our comprehension of contributor profiles, as it provides detailed insights into their individual preferences and priorities. Additionally, while workload composition provides a static view of contributors' activities in their overall engagement in the OSS activities within a period, the dynamics of their activities within that period remain unexplored. With the same

amount of workload, contributors may choose to fulfill their workload gradually over a period or complete tasks in a single burst, depending on their preferences. Exploring work preferences is important as reported by Yue et al. [11] that contribution dynamics can affect the performance of contributors and their technical importance to a project. Moreover, committing is often considered the most important OSS workload. Although we have analyzed the volume of commits made by contributors across different profiles, the technical importance of their commits to the project remains unclear. Understanding contributor workload composition and work preferences can assist project managers in fostering collaborations among contributors with shared interests and similar work patterns. Moreover, understanding the technical importance and the ML-specific contributions of contributors can help project maintainers recognize important contributions and assign tasks that align with each contributor's expertise.

4.2.2 Approach

We explore and compare the behaviors of contributors across different profiles during their engagements in OSS activities from four aspects: their workload compositions, work preferences, technical importance, and ML-specific contributions. Our detailed approach is described as follows:

4.2.2.1 Workload Composition Analysis

As described in Section 3.6.1, we identify five workload composition patterns that

reflect five types of workload compositions of contributors within a period. We investigate the frequency of each type of workload composition for each contributor during one's active periods. The most frequent workload composition pattern of a contributor is identified as one's major pattern. For each profile, we calculate the proportion of contributors with each workload composition pattern as their major pattern and compare the proportions across profiles.

Chi-square test [100] is a statistical test that can determine whether the proportions of categorical variables are the same across different groups or populations. We use the Chi-Square test to compare the distributions of major workload compositions (i.e., most frequent workload composition) of core and peripheral contributors with the null hypothesis $H0_4$ (with a significance threshold of p-value < 0.05), as well as their respective *workhour* and *afterhour* subgroups with the null hypothesis $H0_5$ and $H0_6$.

- $H0_4$*: The distributions of the major workload composition patterns are the same across core contributors and peripheral contributors.*
- $H0_5$*: The distributions of the major workload composition patterns are the same across Core-Afterhour and Core-Workhour contributors.*
- $H0_6$*: The distributions of the major workload composition patterns are the same across Peripheral-Afterhour and Peripheral-Workhour contributors.*

Cramér's V is an effect size measure derived from the chi-squared statistic and is often used to quantify the magnitude of the difference identified by the Chi-square test. We calculate Cramér's V for the groups with significant differences identified by our Chi-square tests to measure effect sizes. We employ the same interpretation of Cramér's V as [101] at degrees of freedom of 4 (i.e., negligible: $0 < .05$, small: .05

< .15, medium .15 < .25, and large: >= .25).

4.2.2.2 Work Preference Analysis

To measure the temporal contribution dynamic and preferences of contributors, we extract 9 work preference features from the OSS activity time series of each contributor within each 90-day period as described in Section 3.6.2. We apply two-sided Mann-Whitney U test [98] to compare each work preference feature between contributors in *core* and *peripheral* profiles with the null hypothesis $H0_7$. We also compare their respective *workhour* and *afterhour* subgroups with the null hypothesis $H0_8$ and $H0_9$, as specified below. We apply Cliff's delta [99] to estimate the effect sizes.

- $H0_7$*: The distributions of the given work preference feature are the same across core contributors and peripheral contributors.*
- $H0_8$*: The distributions of the given work preference feature are the same across Core-Afterhour and Core-Workhour contributors.*
- $H0_9$*: The distributions of the given work preference feature are the same across Peripheral-Afterhour and Peripheral-Workhour contributors.*

4.2.2.3 Technical Importance Assessment

Four technical importance metrics are introduced in Section 3.6.3 to evaluate the importance of a contributor's code contributions. We apply the Mann-Whitney U test to compare each technical importance metric across *core* and *peripheral* contributors

with the null hypothesis $H0_{10}$, as well as their respective *workhour* and *afterhour* subgroups with the null hypothesis $H0_{11}$ and $H0_{12}$ respectively. We also calculate Cliff's delta to estimate the effect sizes.

- $H0_{10}$*: The distributions of the given technical importance metric are the same across core contributors and peripheral contributors.*
- $H0_{11}$*: The distributions of the given technical importance metric are the same across Core-Afterhour and Core-Workhour contributors.*
- $H0_{12}$*: The distributions of the given technical importance metric are the same across Peripheral-Afterhour and Peripheral-Workhour contributors.*

4.2.2.4 ML-Specific Contribution Analysis

To categorize the content of contributors' code contributions, we label 96 sampled pull requests from each profile (in total 384) into one of six categories: User-Level API, Graph-Level Implementation, Operation Implementation, General Utility, Environment-Dependent Processing, or General Software Development (as described in Section 3.6.4). We apply the Chi-square test to compare the distribution of different types of ML-specific contributions between *core* and *peripheral* contributors with the null hypothesis $H0_{13}$, as well as their respective *workhour* and *afterhour* subgroups with the null hypothesis $H0_{14}$ and $H0_{15}$ respectively. We calculate Cramér's V to measure effect sizes. We employ the same interpretation of Cramér's V as [101] at degrees of freedom of 5 (i.e., negligible: $0 < .04$, small: $.04 < .13$, medium $.13 < .22$, and large: $>= .22$).

- $H0_{13}$*: The distributions of the given category of contribution are the same across core contributors and peripheral contributors.*
- $H0_{14}$*: The distributions of the given category of contribution are the same across Core-Afterhour and Core-Workhour contributors.*
- $H0_{15}$*: The distributions of the given category of contribution are the same across Peripheral-Afterhour and Peripheral-Workhour contributors.*

4.2.3 Results

4.2.3.1 Workload Composition

Table 4.3: Distribution of major workload composition patterns across contributor profiles.

Major Workload Composition Pattern	**Core-Afterhour**	**Core-Workhour**	**Peripheral-Afterhour**	**Peripheral-Workhour**
Issue Reporter	14.6%	10.4%	**17.6%**	**14.8%**
Issue Discussant	8.2%	7.5%	9.1%	9.3%
Committer	**45.9%**	**54.4%**	**57.6%**	**58.5%**
PR Discussant	**23.9%**	**17.8%**	14.2%	**15.5%**
Core Reviewer	7.3%	9.9%	1.6%	1.8%

Based on our Chi-Squared test and Cramér's V results, we reject the null hypothesis $H0_4$, $H0_5$, and $H0_6$. The test results indicate significant differences in the distribution of major workload composition patterns between contributor profiles, with a medium effect size for the difference between *core* and *peripheral* contributors and a small effect size for both the difference between their *Workhour* and *Afterhour* subgroups.

Committer is the most common workload composition pattern for all contributor profiles. PR Discussant is the second most frequent for core contributors and Peripheral-workhour contributors, whereas Issue Reporter is the second most for peripheral-afterhour contributors. As shown in Table 4.3, contributors with Committer as their major pattern constitute the highest proportion in all profiles ranging from 45.9% to 58.5%. This finding challenges the intuitive assumption that peripheral contributors would primarily focus on issue reporting or discussion and less focus on code contributions. Instead, the majority of peripheral contributors focus on code contribution through committing. *Core-Afterhour* contributors have the highest percentage of PR Discussants (23.9%) and the lowest percentage of Committers (45.9%). This result aligns with the findings in RQ1 (i.e., *Core-Afterhour* contributors tend to make more complex contributions that require substantial discussions in pull requests), often with fewer but larger commits compared to *Core-Workhour* contributors. The high prevalence of PR Discussant in both *core* profiles, combined with the result from RQ1 (i.e., *core* contributors are significantly more involved in discussions on others' pull requests), indicates that a considerable proportion of *core* contributors are multitaskers who often engage in a mix of coding, mentoring, and project management activities. Although Code Reviewers make up the smallest proportion of core contributors, the majority of Code Reviewers are *core* contributors, indicating their focus on maintaining and enhancing the quality of the codebase.

For *peripheral* contributors, both Issue Reporter and PR Discussant are secondary workload composition patterns with similar prevalence (ranging from 14.2% to 17.6%). Given that less than 24% of *peripheral* contributors have commented on

others' pull requests, this finding indicates that *peripheral* contributors largely consist of project users who identify issues or request specific features based on their use of the ML library. Their involvement in pull request discussions often revolves around collaborating with core members to implement desired changes, reflecting a user-driven engagement with the project.

4.2.3.2 Work Preferences

Table 4.4: Result of Mann-Whitney U test and effect size results for work preference features.

Work Preference Features	**Group1: Core** **Group2: Peripheral**	**Group1: Core-Afterhour** **Group2: Core-Workhour**	**Group1: Peripheral-Afterhour** **Group2: Peripheral-Workhour**
binned_entropy	large (0.49)	negligible (-0.056)	negligible (0.051)
c3(1)	medium (0.331)	negligible (-0.035)	negligible (0.026)
c3(2)	small (0.279)	negligible (-0.037)	negligible (0.02)
c3(3)	small (0.292)	negligible (-0.042)	negligible (0.016)
number_cwt_peaks	large (0.495)	negligible (-0.079)	negligible (0.048)
longest_strike_above_mean	medium (0.372)	negligible (-0.026)	negligible (0.044)
longest_strike_below_mean	medium (-0.463)	negligible (0.088)	not significant
diverse	small (0.271)	not significant	negligible (0.067)
balance	small (-0.327)	negligible (0.077)	negligible (-0.018)

Groups with no significant difference (i.e., null hypothesis is accepted) are indicated with 'not significant'. Groups that are significantly different examined by Mann-Whitney U test are indicated with Cliff's delta value and its interpretation (i.e., negligible, small, medium, or large). A positive value means group1 is greater than group2, and vice versa. Three columns from left to right correspond to the null hypothesis $H0_7$, $H0_8$, and $H0_9$ respectively.

As shown in Table 4.4, the null hypothesis $H0_7$ is rejected for all work preference features. This indicates that *core* and *peripheral* contributors have significantly different work preferences, in terms of their contribution dynamics and the level of balance of their contributions across various types of OSS activities.

Core contributors exhibit more complex contribution dynamics compared to peripheral contributors. Contribution dynamics of *core* contributors, measured by *binned_entropy*, *c3(1)*, *c3(2)*, and *c3(3)*, exhibit significantly higher complexity than *peripheral* contributors, indicating greater fluctuations in their period OSS activity time series. Moreover, *core* contributors have significantly more peaks (i.e., *number_cwt_peaks*), a higher ratio of continuous time above mean (i.e., *longest_strike_above_mean*), and a lower ratio of the continuous time below mean (i.e., *longest_strike_below_mean*) in their OSS activities time series, compared to *peripheral* contributors. This indicates that ***core* contributors tend to make more peak contributions within a period and engage in longer times of continuous intensive contributions, while *peripheral* contributors tend to maintain constant contribution levels and have longer continuous inactive time.**

Core contributors make less balanced contributions than peripheral contributors. As shown in Table 4.4, *Core* contributors are significantly higher than *Peripheral* contributors on *diverse* with a small effect size and significantly lower on *balance* with a medium effect size. **This indicates that *Core* contributors tend to make more diverse contributions across various types of OSS activities with a stronger focus on one or few types of OSS activities, while peripheral contributors make more even amount of contributions across fewer types of OSS activities.**

The work preference differences between Workhour and Afterhour subgroups are negligible. The null hypothesis $H0_8$ and is rejected for 8 (out of 9) work preference features, indicating that compared to *Core-Afterhour* contributors, *Core-Workhour* contributors tend to have more complex contribution dynamics, and make

Table 4.5: Mann-Whitney U test and effect size results for contributor technical importance.

Technical Importance	Group1: Core Group2: Peripheral	Group1: Core-Afterhour Group2: Core-Workhour	Group1: Peripheral-Afterhour Group2: Peripheral-Workhour
max_period_centrality	medium (0.473)	negligible (-0.094)	not significant
max_centrality_period	medium (0.467)	negligible (-0.089)	not significant
max_commit_centrality	medium (0.452)	negligible (-0.078)	not significant
max_centrality_day	medium (0.46)	not significant	negligible (0.045)

Three columns from left to right correspond to the null hypothesis $H0_{10}$, $H0_{11}$, and $H0_{12}$ respectively.

more diverse and less balanced contributions across different types of OSS activities. However, the differences are negligible in practice. Similarly, the null hypothesis $H0_9$ is also rejected for 8 work preference features, with negligible effect sizes. This shows that *Peripheral-Afterhour* tend to have more complex contribution dynamics than *Peripheral-Workhour* contributors, but the differences are as well negligible in practice.

4.2.3.3 Technical Importance

Core contributors tend to achieve a higher maximum technical importance within a project than peripheral contributors. As shown in Table 4.5, the null hypothesis $H0_{10}$ is rejected for all technical importance metrics with medium effect sizes. *Core* contributors have significantly higher maximum technical importance than *peripheral* contributors, both at the commit-level (i.e., *max_commit_centrality*) and period-level (i.e., *max_period_centrality*). This means that when comparing the importance of individual commits, which considers the average influence of files changed within the commit, the most important commit made by *core* contributors tends to have higher importance (i.e., *commit_centrality*) than the most important commit made by *peripheral* contributors. Similarly, the period

of the highest technical importance for *core* contributors, which considers both the importance and intensity of commits made within a period, tends to have higher importance (i.e., *period_centrality*) than those for *peripheral* contributors. Therefore, *core* contributors tend to reach higher technical importance within a project by making intensive changes to central files, which have a greater impact on the codebase and core functionalities. *Peripheral* contributors tend to stay with modifying peripheral files that have less impact on the system and are often confined to very specific functions, such as updating documentation or fixing a parameter in an API. An example illustrating commit contributions of different levels of technical importance can be found in Figure A.5 in Appendix A.3, where commits with high and low centrality are made to core kernel functions and TensorFlow Lite demo code, respectively.

Core contributors tend to require a longer time to progress to their maximum technical importance than peripheral contributors. As shown in Table 4.5, *core* contributors are significantly higher than *peripheral* contributors in terms of both the number of days before making their most important commit (i.e., *max_centrality_day*) and the number of periods to reach the period of their highest technical importance (i.e., *max_centrality_period*), indicating they require a longer time to reach their maximum technical importance within a project. *Core-Afterhour* and *Core-Workhour* contributors have a median of 100.5 and 126 days, respectively, to make their most important commit. For both *core* profiles, contributors take a median of 2 periods to reach the period of the highest technical importance. **This suggests a progression in technical importance for *core* contributors within the first six months of activity, followed by a shift away from highly technical**

and intensive contributions. In contrast, *Peripheral-Afterhour* and *Peripheral-Workhour* contributors have a median of reaching the period of the highest technical importance in 1 period and making the most important commit within 6 and 4 days respectively, indicating that **the technical importance of *peripheral* contributors tends to remain the same as they join the project. The *peripheral* contributors do not make significant progression over time.**

The differences in technical importance between the Workhour and Afterhour subgroups are negligible. The null hypothesis $H0_{11}$ is rejected for the technical importance metrics *max_period_centrality*, *max_centrality_period*, and *max_commit_centrality*, all with negligible effect sizes. This means that *Core-Workhour* contributors tend to achieve a higher maximum level of technical importance and take longer to reach this maximum compared to *Core-Afterhour* contributors, although the difference is negligible in practice. The null hypothesis $H0_{12}$ is only rejected for *max_centrality_day* with negligible effect sizes. This indicates that while both *Peripheral-Workhour* and *Peripheral-Afterhour* contributors reach similar levels of maximum technical importance, *Peripheral-Afterhour* contributors tend to take longer to make their most important commit compared to their *workhour* counterparts, though this difference is also practically negligible.

4.2.3.4 ML-Specific Contributions

Table 4.6: Distribution of ML-specific contributions across contributor profiles.

Category of Contribution	Core-Afterhour	Core-Workhour	Peripheral-Afterhour	Peripheral-workhour
User-Level API	8.3%	16.3%	**27.1%**	**37.5%**
Graph-Level Implementation	11.5%	17.4%	14.6%	9.4%
Operation Implementation	**29.2%**	**18.5%**	17.7%	16.7%
General Utility	11.5%	5.4%	4.2%	6.3%
Environmental-Dependent Processing	19.8%	13%	7.3%	10.4%
General Software Development	**19.8%**	**29.3%**	**29.2%**	**19.8%**

Based on our Chi-Squared test and Cramér's V results, we reject the null hypothesis $H0_{13}$ with a large effect size, while accepting $H0_{14}$ and $H0_{15}$. The test results indicate significant differences in the distribution of ML-specific contributions between *core* and *peripheral* contributors, with a large effect size. The differences between their *Workhour* and *Afterhour* subgroups are insignificant.

Operation implementation is the most frequent ML-specific contribution for Core contributors. *Core-Afterhour* contributors have the highest percentage of contributions to operation implementations (30.2%), which are core functionalities of the deep learning framework and often involve complex, DL-specific algorithms. This highlights their expertise and focus on the more complex technical aspects of the project. This finding aligns with previous observations that *Core-Afterhour* contributors tend to make more complex code contributions, which require extensive discussion and result in larger commits. *Core-Workhour* contributors demonstrate a more balanced distribution across different ML-specific contributions, indicating that they often take on diverse responsibilities, ranging from implementing core ML operations to graph-level optimizations and supporting general project

development. This broad involvement indicates that these contributors are likely to be dedicated project maintainers who possess the knowledge and contribute across multiple layers of the ML framework. As stated in the PyTorch contributor guide [1], once a pull request is merged, maintainers must be capable of maintaining the corresponding code. Notably, 65% Pytorch maintainers (as listed in the persons of interests [2]) identified in our dataset are categorized to the *Core-Workhour* profile.

User-level API is the most frequent ML-specific contribution for peripheral contributors. Peripheral contributors have the highest involvement in the User-Level API category, focusing on making changes to APIs related to data processing, model building, model deployment, and utility functions. This indicates their role as users of the ML libraries who observe issues or have demands when using the ML library through the APIs and therefore make code contributions to solve their needs. Therefore, their contributions mainly improve the usability and accessibility of these projects.

Contributions to general utility and environmental-dependent processing are relatively low across all profiles. This can be attributed to the fact that these components take up only a small portion of the project's source code and require less maintenance effort. Additionally, general utilities involve common functions and data structures that are highly reusable across different components of the project, thereby reducing the need for frequent updates. Environmental-dependent processing contributions often require specialized expertise related to specific hardware devices or software environments. Therefore, only a limited number of contributors possess

[1] https://pytorch.org/docs/stable/community/contribution_guide.html
[2] https://pytorch.org/docs/main/community/persons_of_interest.html

the necessary knowledge to make these changes. For example, third-party contributors from companies like Intel or AMD might contribute to ensure compatibility with their new products or introduce new CUDA optimization techniques to improve performance.

General software development contributions are prevalent among all four contributor profiles. This indicates that, similar to traditional OSS projects, contributors in ML projects invest considerable effort in maintaining general software infrastructure and supporting non-ML-specific tasks, regardless of their profiles. These tasks include updating CI/CD scripts, refactoring code, testing, enhancing graphical user interfaces, updating documentation, and other general development activities. Such efforts are important for streamlining the software development workflow and also ensuring the quality and effective integration of ML-specific contributions into the project.

RQ2 Summary

The most common workload composition pattern among all contributor profiles is Committer. Core contributors demonstrate more complex contribution dynamics and make less balanced contributions compared to peripheral contributors. Core contributors also tend to achieve higher maximum technical importance and exhibit a progression towards increasing technical importance within the first 2 active periods, whereas peripheral contributors typically have the highest technical importance when initially joining the project. The most frequent ML-specific contribution is operation implementation for core contributors, while contributions to user-level API are the most frequent for peripheral contributors.

4.3 RQ3: What are the important factors of contributor OSS engagement for increasing the popularity of a project?

4.3.1 Motivation

The diverse distribution of contributors with varying OSS engagements results in varied project dynamics. Analogous to how demographic diversity influences a nation's growth and stability, the distribution of contributors with different workload compositions and work preferences can significantly impact the success and vitality of an OSS project. This becomes particularly valuable during periods of significant turnover, enabling project maintainers to quickly recognize the types of contributions (e.g., issue reporting, issue discussion, committing, pull request discussion, and code review) from departing contributors and assess the potential impact on various project activities in their absence. In this research question, we investigate the association of the diversity of contributors with different workload compositions and work preferences with the project popularity, in terms of the increase in star ratings and the number of forks.

4.3.2 Approach

We build four mixed-effect models to explore how contributor workload composition patterns and work preferences are associated with the increase of project stars and forks. Our detailed approach is outlined as follows:

Independent variable Preparation: We prepare two sets of independent variables: workload composition pattern-related variables and work preference-related

variables as listed in Table 4.7.

- **Workload composition pattern related-variables:** For each period (as defined in Section 3.6.1) within a project, we calculate the ratio of active contributors belonging to each of the five contributor workload composition patterns, and it denotes as *<Pattern_name>_ratio*. For example, *Issue_Reporter_ratio* circulates the proportion of contributors belonging to the Issue Reporter pattern among all active contributors in that period. This yields five variables describing the distribution of contributors of different workload composition patterns during that specific period.

  We also compute the ratio of the five types of OSS contributions (i.e., issue reports, issue comments, commits, pull request comments, and code reviews) made by contributors belonging to each workload composition pattern. We denote it as *<Pattern_name>_<contribution_type>_ratio*. For example, *Issue_Reporter_issue_ratio* denotes the proportion of issue reports raised by contributors belonging to the Issue Reporter pattern among all issue reports raised in that period. Therefore, 25 variables are extracted to capture the diversity of contributions from contributors of five workload composition patterns at that period.

  We add a variable *period* to account for the influence of different developmental stages of software projects on their popularity. In total, 31 workload composition pattern related-variables are extracted. We apply the Spearman Rank Correlation test to measure the variable correlations and remove one from each pair of highly correlated features with a coefficient greater than 0.7 [84]. 13 non-correlated variables remain, as shown in Table 4.7.

- **Workload preference related-variables:** For each period in a project, we calculate the median value of each work preference feature (identified in Section 3.6.2) among the active contributors within that period. A median value represents the collective contributor work preference during that period. As a result, we obtain 9 variables corresponding to the respective median value of each work preference feature. We compute the average contribution made by active contributors to each of the five types of OSS activities within the period, to represent the project-level preference for different types of OSS activities. As previously mentioned, we include a *period* variable to account for the impact of project development stages on the project popularity. In total, we obtain 15 work preference-related variables. We apply the Spearman Rank Correlation test to identify and remove 5 correlated variables. 10 remaining variables are shown in Table 4.7.

Dependent Variable Preparation: We use the number of stars and forks as project popularity metrics. More specifically, the number of forks reflects the interest in a project from potential contributors, while the number of stars indicates interest from both potential users and contributors. We use Github GraphQL API[3] to collect the timestamp of each star received by a project and use Github Forks API[4] to collect the timestamp when each fork occurs. For each project, we count the number of stars and forks gained within each period (i.e., 90 days period as described in Section 3.6.1) and use the increases of stars and forks in consecutive periods as our dependent variables. Due to variations in popularity levels among the subject projects, the counts of stars and forks may be at different scales for different projects.

[3]https://docs.github.com/en/graphql
[4]https://docs.github.com/en/rest/repos/forks

Using the counts of stars and forks directly as dependent variables can skew the model results. Therefore, we apply Min-Max Normalization to scale the star and fork counts for different periods within each project to a range of 0 to 1. The formula of Min-Max Normalization is presented in Equation 3.1, where features are the subject projects and i indexes the periods of the projects. The normalized stars and forks serve as our dependent variables.

Model Construction: We construct four mixed-effect models to identify the associations between *(i)* contributor workload composition pattern related-variables and the increase in star ratings, *(ii)* workload composition pattern related-variables and the increase in forks, *(iii)* work preference related-variables and the increase in star ratings, and *(iv)* work preference related-variables and the increase in forks respectively.

Mixed-effects models are statistical regression models that integrate both fixed and random effects [102]. Fixed effects are variables with consistent slopes and intercepts across all data points, while random effects account for variations between experimental groups, such as projects in our case. Mixed-effect models can estimate different slopes or intercepts for different experimental groups. Our mixed-effect models use projects as a grouping factor and assume different intercepts for different projects. We construct the mixed-effect models using Python package *statsmodels* [103]. Note that we exclude the initial periods for each project, where no forks are recorded (i.e., first 19, 13, 2, and 1 periods for Pytorch, Theano, scikit-learn, and MXNet respectively), due to a concern of these repositories being private during those times. Similarly, the first period for every project is excluded from the model training, because we find that for all projects, the first period has significantly more stars and forks than other

periods, which could skew the model results.

Model Explanatory Power Evaluation: We use conditional R^2 and marginal R^2 metrics to evaluate the fit of our mixed-effect models. Marginal R^2 describes the proportion of variance explained by fixed effects. Conditional R^2 describes the proportion of variance explained by both fixed effects and random effects [104].

Model Result Analysis: To understand the important features in the mixed-effect models, we examine the z-values and their corresponding p-values for each independent variable in the model result. z-value is associated with the t-statistic which tests the significance of each fixed effect coefficient, and it is more reliable than Wald Statistics with limited data points [94]. A fixed effect is significant if it has $Pr(>|z|) < 0.05$, and such variables are marked with asterisks in Table 4.7. We also analyze the coefficient sign of each independent variable. Positive coefficients, denoted by upward arrows (↗) in Table 4.7, suggest a positive correlation with the dependent variables (i.e., stars or forks), whereas negative coefficients, indicated by downward arrows (↘), imply a negative correlation.

4.3.3 Results

The mixed-effect models of modeling star increase with work preference features have the highest explanatory power. Our mixed-effect model for analyzing work preferences and star increase has both conditional R^2 and marginal R^2 of 0.64, and the model for work preferences and fork increase has a conditional R^2 and marginal R^2 of both 0.54. This result shows that both models have good explanatory power and most of the variances are explained by the fixed effects. The mixed-effect

Table 4.7: Mixed-effects models results.

Work Preference	**Star**		**Fork**	
	Signif.	**Rel.**	**Signif.**	**Rel.**
period	***	↘	***	↘
binned_entropy	**	↘	**	↘
c3_1		↗	.	↗
diverse		↗		↗
balance	***	↗	***	↗
commit	***	↘	***	↘
issue	**	↗		↗
issue_comment		↘		↗
pr_comment		↘		↗
review	***	↗	***	↗
Workload Composition Pattern				
Issue_reporter_ratio	*	↗	*	↗
Issue_discussant_ratio	***	↗	***	↗
Committer_ratio		↗	.	↗
Issue_reporter_issue_ratio		↗		↗
Issue_discussant_issue_comment_ratio	**	↘	**	↘
Issue_discussant_review_ratio	*	↗	.	↗
Committer_issue_ratio	*	↘		↘
Committer_review_ratio	.	↗		↗
PR_Discussant_commit_ratio	*	↘	.	↘
PR_Discussant_issue_ratio		↗		↗
PR_Discussant_review_ratio	***	↗	***	↗
Code_reviewer_issue_ratio		↗	.	↗
Code_Reviewer_review_ratio	.	↗	*	↗

Signif. codes: Pr(>|z|) = **0 '***' 0.001 '**' 0.01 '*' 0.05** '.' 0.1 ' ' 1
Rel.: ↗ stands for a positive association between the independent and dependent variables, and ↘ stands for a negative association.

model of workload composition patterns with star increase has both conditional R^2 and marginal R^2 of 0.31, and the model of fork increase has both conditional R^2 and marginal R^2 of 0.26, indicating that both models have all the variances explained by the fixed effects.

4.3.3.1 Important Work Preference Features to Project Popularity

Work preference features such as the balance of contribution and the average contributor code reviews have significant positive associations with project popularity in terms of both star rating and the number of forks. Table 4.7 presents the variable importance and coefficients of our mixed effect models. We observe that more balanced contributions across various types of OSS activities (i.e., *balance*) and a higher number of average code reviews conducted by each contributor (i.e., *review*) have a significant positive association with increases in both stars and forks. This association indicates that when contributors distribute their efforts evenly across various OSS activities or engage in more code reviews, the likelihood of attracting new users or contributors tends to rise.

The developmental stages of a project (i.e., *period*) and the average number of commits per contributor (i.e., *commit*) are negatively associated with star and fork increases. This indicates a reduced likelihood of attracting more new users and contributors as project development progresses to later stages and when contributors are heavily taken on committing responsibilities. The complexity of contributors' contribution dynamics (i.e., *binned_entropy*) is also negatively correlated with star and fork increases, indicating an increased likelihood of attracting potential contributors when existing contributors have constant and less random contribution frequencies. The average number of issue reports raised per contributor (i.e., *issue*) is positively associated with star increases. As issue reporting is often an accessible entry point for newcomers[5], the active participation of current contributors in the issue discussions (i.e.,*Issue_Discussants_ratio*) could help newcomers efficiently resolve their problems or convert their ideas into appropriate code contributions. Potential users and contributors may see the project as responsive and supportive, then star the project.

[5]https://github.com/pytorch/ignite/blob/master/CONTRIBUTING.md

4.3.3.2 Important Workload Composition Features to Project Popularity

Workload composition pattern features, such as the ratio of Issue Reporters and Issue Discussants among all contributors, and the ratio of code reviews made by PR Discussants, have significant positive associations with project popularity in terms of both star rating and the number of forks. As shown in Table 4.7, the ratio of Issue Reporters (i.e., *Issue_Reporter_ratio*) and Issue Discussants (i.e., *Issue_Discussants_ratio*) among all contributors in the project has a significant positive association with the increase in both stars and forks. This suggests that projects are more likely to gain popularity when a higher proportion of contributors focus on reporting issues or requesting new features, and when more contributors are actively involved in discussing these issues and actively responding to questions or requests from users or potential contributors. Additionally, PR Discussants focus on participating in pull request discussions, often guiding other contributors to revise and align their changes with project requirements. Their familiarity with proposed changes makes their subsequent code reviews more reliable, which potentially improves the quality of the codebase. As a result, projects with more code reviews conducted by PR Discussants are more likely to gain community trust and ultimately increase their popularity. Furthermore, when newcomers submit pull requests, receive guidance from experienced contributors, and ultimately have their contributions approved by them, they are likely to feel welcomed, supported, and acknowledged for their efforts. This continuity of support fosters a sense of belonging and value, helping to create an inclusive environment that attracts more users and contributors.

A higher ratio of code reviews conducted by Issue Discussants (i.e.,

***Issue_Discussant_review_ratio*) has a significant positive association with the increase in star ratings.** This association indicates that projects tend to gain more star ratings when a larger portion of code reviews are made by Issue Discussants. A possible explanation is that Issue Discussants are familiar with ongoing project issues and feature requests through their active participation in the issue discussions. Therefore, their code reviews are particularly effective in assisting contributors to resolve their issues or implement features in a way that aligns with the immediate needs of the project. This interaction could be especially beneficial for retaining newcomers, as Issue Discussants may already engage with newcomers during issue discussions before helping them with code reviews. When newcomers begin their journey in the project by discussing their issues with existing contributors and eventually making successful pull requests with their guidance, they are likely to feel welcomed and accepted within the project community. A real-world example is shown in Figure A.4 in Appendix A.3.

A higher proportion of code reviews conducted by Code Reviewers is significantly positively associated with the increase in forks. Table 4.7 shows that the proportion of code reviews made by Code Reviewers (i.e., *Code_Reviewer_review_ratio*) have a significant positive correlation with fork increases but not with stars. This observation can be explained by the fact that, while both users and potential contributors star projects to express their interest, individuals who fork projects often have the intention of starting to make code contributions. Therefore, the code reviews provided by Code Reviewers hold greater significance for potential contributors than users.

RQ3 Summary

Contributors making balanced contributions across various OSS activities and a higher average number of code reviews are significantly associated with the increase in project popularity in terms of star ratings and forks. Moreover, a higher proportion of Issue reporters and Issue Discussants among existing contributors and a larger portion of code reviews made by PR Discussants are significantly associated with greater increases in star ratings and forks. The proportion of code reviews conducted by Issue Discussants is significantly associated with star ratings. The proportion of code reviews made by Code Reviewers is significantly associated with the increase in forks.

4.4 RQ4: How does contributor OSS engagement evolve?

4.4.1 Motivation

From our observation of contributor engagement within a project in RQ2, we note that their OSS engagements are not static; they shift over time. This motivates us to explore how the workload composition of contributors across different profiles changes as they spend more time on a project, as well as how the demand for contributors with diverse workload compositions changes as project development progresses. Moreover, we are also interested in understanding how contributors' work preferences and technical importance shift as they stay longer in the project and make more contributions. To provide insights into the behavior and trajectory of contributors, we aim to identify if there are significant trends in the evolution of contributors' workload composition, work preferences, and technical importance. Recognizing such trends

may assist project maintainers in better comprehending the dynamics of potential long-term contributors, offer a roadmap for motivated new contributors, and suggest potential paths for their progression and advancement within the project.

4.4.2 Approach

We explore the trends in the evolution of workload compositions, work preferences, and technical importance. Our detailed approach is described as follows:

4.4.2.1 Evolution of Workload Composition

To identify the evolution of a contributor's workload composition, we first extract the contributor's active *periods* (as described in Section 3.6.1), during which the contributor has made OSS contributions to the project. We then extract the workload composition pattern associated with the contributor in each period, thus obtaining a time series of workload composition patterns that reflects the contributor's workload compositions over time. We encode the workload composition patterns into numerical values, with Issue Reporter represented as 1, Issue Discussant as 2, Committer as 3, Collaborative Committer as 4, and Code Reviewer as 5. This numerical representation aligns intuitively with the level of expertise needed to handle the respective workload.

Cox-Stuart trend test [105] is a statistical method to detect the presence of a monotonic trend in time series. We employ the Cox-Stuart trend test from the R package *randtests* to identify if a significant increasing or decreasing trend exists in our numerical workload composition time series. For each time series, we use the

right-sided Cox-Stuart trend test to detect the increasing trend with null hypothesis $H0_{16}$ and the left-sided Cox-Stuart trend test to detect the decreasing trend with $H0_{17}$, with a 95% confidence level. We test against these two null hypotheses for all the Cox-Stuart trend tests in the rest of the paper.

- $H0_{16}$*: There is no significant increasing trend in the given time series/sequence.*
- $H0_{17}$*: There is no significant decreasing trend in the given time series/sequence.*

When applying the Cox-Stuart trend test on workload composition time series, we exclude the contributors who have less than 2 active periods, as the test requires at least two datapoints in the time series.

To pinpoint the changes in contributors' workload composition at different stages of their project involvement, we divide their workload composition time series into three equal-length subsequences, representing their workload compositions in the early, middle, and late stages of the project involvement. We exclude the contributors with fewer than 3 active periods (at least one datapoint in each stage is required) and apply the Cox-Stuart trend test on the jointed sequence of the early and middle stages to examine the changes in workload composition from the early to middle stage, and the same for the middle to late stage.

Additionally, we examine the evolution of both the prevalence and distribution of each workload composition pattern within the subject projects, to understand the sustainability and the demand for contributors with particular workload compositions across different stages of project development. For each project, we count the number of contributors belonging to a workload composition pattern in each period into a time series to analyze the prevalence of the pattern as project development progresses. Similarly, we build another time series for the ratio of contributors belonging to a

workload composition pattern among all contributors in each period to analyze how the distribution of the pattern evolves within a project. We apply the Cox-Stuart trend test to examine the trends in the time series of counts and ratios of contributors belonging to each workload composition pattern. To further pinpoint the major changes in contributor workload composition at different project development stages, we divide each time series into three equal-length subsequences, representing the early, middle, and late stages of project development. We apply the Cox-Stuart trend test on the jointed sequence of the early and middle stages to identify the change of contributor workload compositions in a project, from its early to middle stages, and the same for the middle to late stage.

4.4.2.2 Evolution of Work Preferences

To identify trends in the evolution of contributors' work preferences, our approach is similar to identifying the evolution of workload composition in Section 4.4.2.1. Figure 4.1 illustrates our detailed approach. For each contributor, we first extract the periods where the contributor is active in the project. We extract nine work preference features (as described in Section 3.6.2) from the contributor's OSS activity time series within each active period. We build a time series for each work preference feature, representing its evolution across the contributor's active periods. Therefore, we obtain nine work preference feature time series for each contributor to analyze the evolution of their work preferences. We apply the same procedure as in Section 4.4.2.1 to select contributors and examine the trends in their work preference feature time series, as well as the trends in the early-to-middle-stage and middle-to-late-stage subsequences.

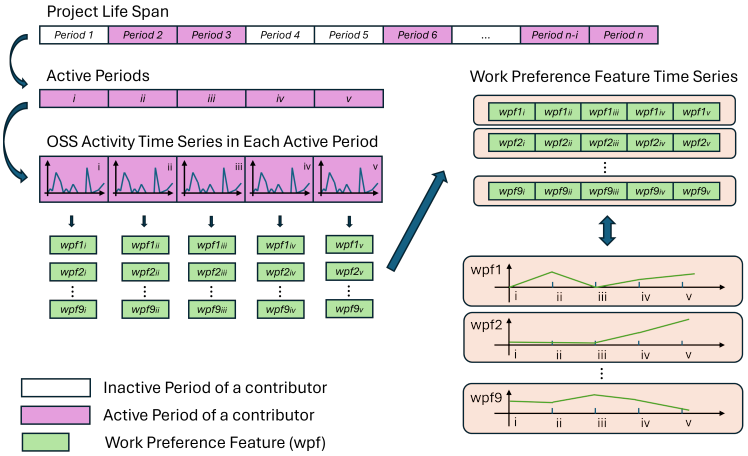


Figure 4.1: Example of building a contributor's work preference feature time series.

4.4.2.3 Evolution of Technical Importance

We study the evolution of contributors' technical importance at two levels of granularity: per commit and per period. Analyzing contributors' technical importance at the commit level allows us to understand the shift in the importance of their code changes more precisely, such as from making changes to the very specific functions in the peripheral files to modifying the influential files that have a greater impact on the system, or the opposite. Moreover, period-level evolution reflects contributors' changes in technical importance at a higher level, considering both the importance and the intensity of their code contribution over time.

To identify the trend of contributors' commit importance, we extract all commits made by a contributor and sort them into chronological order. We calculate the *commit centrality* (as described in Section 3.6.3) of each commit, thereby obtaining a sequence of commit centrality that reflects the importance of each commit made by a contributor over time. We apply the same procedure as in Section 4.4.2.1 to select

contributors and examine the trends in their commit centrality sequence, as well as the trends in the early-to-middle-stage and middle-to-late-stage subsequences.

Similarly, to identify the trend of contributors' period technical importance, we extract the contributors' active periods, in which a contributor has made OSS contributions to the project. We calculate the *period centrality* (as described in Section 3.6.3) of each active period and construct a time series of period centrality for each contributor. We examine the trends in their period centrality time series and the trends in the early-to-middle-stage and middle-to-late-stage subsequences.

4.4.3 Results

Table A.2 shows the overall evolution trends of workload composition, work preference, and technical importance for contributors in each profile. Table A.3 presents the early-to-middle-stage and middle-to-late-stage trends. In these tables, the upward arrows denote rejection of the null hypothesis $H0_{16}$, indicating increasing trends in the given time series, with the percentages indicating the proportion of contributors within the profile exhibiting increasing trends. Likewise, downward arrows represent the rejection of the null hypothesis $H0_{17}$ and decreasing trends.

4.4.3.1 Evolution of Workload Composition

A small portion of long-term contributors exhibit significant trends in their workload composition towards handling tasks requiring higher expertise, while the majority of contributors do not exhibit a significant trend in

shifting workload composition. As shown in Table A.2, approximately 3% *core* contributors and 0.3% *peripheral* contributors demonstrate significant trends in shifting workload composition. We manually investigate these contributors and observe that they are all long-term contributors with at least 10 active periods (i.e., being active in the subject project for more than 2.5 years). Among these contributors, 78% have an increasing trend in their workload composition, which means that their workload composition evolves in the direction of Issue Reporters → Issue Discussants → Committer → PR Discussant → Code Reviewer. This indicates that these long-term contributors tend to shift their focus toward the OSS tasks that are typically considered to require higher expertise. Table A.3 shows that the increasing trends of workload composition for long-term contributors primarily occur during their early to middle stage of project involvement. Approximately 1% *core* contributors exhibit an increasing trend in workload composition during this stage, while only 0.2% experience a decreasing trend. In contrast, an equal proportion of long-term contributors show increasing and decreasing trends in workload composition during their middle to late stages of involvement.

The Committer's and Code Reviewer's contributions are more sustainable compared with other roles as ML project development progresses. Table 4.8 shows the trends in the prevalence (i.e. count) and the proportion (i.e., ratio) of contributors exhibiting each workload composition pattern within the subject projects. In all projects, both in their early stages and overall trends, the number of Committers and their ratio within the project either remains constant or demonstrates increasing trends. This shows the enduring popularity and community engagement of the selected ML projects. However, in the late stages, MXNet and Theano/Aesara

exhibit a decreasing trend in the number of Committers, possibly due to the delaying popularity of these projects. Seven (out of eight) projects show either an increasing trend or a stable ratio of Code Reviewers, indicating the sustained demand for code reviews throughout the ML project development stages. Additionally, five projects exhibit an increasing trend in the count and ratio of Code Reviewers in their early to middle stages, indicating significant progression of contributors to Code Reviewers during this period, with only a few contributors eligible for code review in the early stage.

The Issue Reporter's and Issue Discussant's contributions are less sustainable compared with other workload composition patterns as ML project development progresses. As shown in Table 4.8, 4 (out of 8) projects exhibit a decreasing trend in the prevalence of Issue Reporters, and 5 (out of 8) projects exhibit decreasing prevalence of Issue Discussants. This trend likely stems from the emerging phase of the ML library. When a new ML library emerges, there is often a surge of interest due to the growing popularity of ML in recent years. For instance, we observe that TensorFlow, Keras, MXNet, and ONNX reach the peak number of Issue Reporters within 1 to 2 years after they open-sourced. During this time, OSS community members are eager to explore the new ML library and actively provide feedback, report bugs, and request new features. After this initial interest, enthusiastic contributors may shift their focus toward code contributions possibly transitioning to roles like Committers, while others may depart from the project.

4.4.3.2 Evolution of Work Preferences

Table 4.8: Project-level trend of workload composition.

Project Stages	Project	Issue Reporter		Issue Discussant		Committer		PR Discussant		Code Reviewer	
		Count	Ratio	Count	Ratio	Count	Ratio	Count	Ratio	Count	Ratio
Early-Middle Trend	tensorflow	-	(***)↘	-	(***)↘	(***)↗	(***)↗	-	-	-	-
	pytorch	-	-	(*)↗	-	(*)↗	-	-	-	(**)↗	(**)↗
	scikit-learn	(***)↗	-	(***)↗	-	(***)↗	-	(***)↗	(*)↘	(***)↗	(***)↗
	keras	(**)↘	(***)↘	(*)↘	-	-	-	-	-	-	-
	mxnet	(***)↘	(***)↘	-	-	-	-	(*)↗	(**)↗	(**)↗	(***)↗
	theano_aesara	-	-	-	-	-	-	-	-	(***)↗	(***)↗
	onnx	-	-	-	-	-	-	-	-	(*)↘	-
	deeplearning4j	(**)↗	(**)↘	(**)↗	-	(*)↗	(*)↗	(***)↗	-	(***)↗	(*)↗
Middle-Late Trend	tensorflow	(***)↘	(***)↘	(***)↘	(***)↘	-	(***)↗	(**)↘	-	-	-
	pytorch	(*)↗	-	-	-	-	(*)↘	-	-	(**)↗	-
	scikit-learn	-	-	-	-	(*)↗	(*)↗	-	(*)↘	-	-
	keras	-	-	-	(*)↘	(*)↗	(***)↗	-	(*)↘	-	-
	mxnet	(***)↘	(**)↘	(***)↘	-	(***)↘	(*)↘	(***)↘	-	(***)↘	-
	theano_aesara	-	-	-	(*)↗	(**)↘	(***)↘	-	-	-	-
	onnx	-	-	-	-	-	-	-	-	-	-
	deeplearning4j	(*)↘	-	(***)↘	(***)↘	-	-	-	-	(***)↘	-
Overall Trend	tensorflow	(*)↘	(***)↘	(*)↘	(***)↘	(***)↗	(***)↗	-	-	-	(*)↘
	pytorch	(*)↗	-	(*)↗	-	(***)↗	(**)↘	(*)↗	(*)↗	(***)↗	(***)↗
	scikit-learn	(***)↗	-	(***)↗	-	(***)↗	-	(*)↗	(***)↘	(***)↗	(**)↗
	keras	(***)↘	(**)↘	(***)↘	-	-	(**)↗	-	(*)↗	-	-
	mxnet	(***)↘	(***)↘	(***)↘	-	(**)↘	(*)↘	-	(*)↗	-	(***)↗
	theano_aesara	(***)↘	-	(**)↘	-	(***)↘	(**)↘	(***)↘	-	(**)↗	(***)↗
	onnx	-	-	-	-	-	-	-	-	(**)↘	-
	deeplearning4j	-	-	-	(*)↘	(***)↗	(***)↗	-	-	-	(**)↗

↗ : The null hypothesis $H0_{16}$ is rejected and the given time series has an increasing trend.
↘ : The null hypothesis $H0_{17}$ is rejected and the given time series has a decreasing trend.
- : Both $H0_{16}$ and $H0_{17}$ are accepted. There is no significant trend in the given time series.
Confidence Level: (***) 99.9%, (**) 99%, (*) 95%.

A portion of long-term contributors show a tendency towards fewer, constant, and balanced contributions, while the majority of contributors do not demonstrate a clear trend in shifting work preferences. Similar to our finding on workload compositions, significant trends in work preferences are observed only among long-term contributors (primarily core contributors). For the contributors with significant changes in each work preference feature, we observe that 69% to 81% of them have a decreasing trend for the complexity of contribution dynamics (i.e., *binned_entropy* and *C3 statistics*), the number of peak contributions (i.e., *number_cwt_peaks*), and the time with continuous intensive contribution (i.e., *longest_strike_above_mean*). This suggests that these long-term contributors tend to reduce intensive contributions and progress towards making constant contributions.

Moreover, among contributors who have significant changes in their contribution balance, we observe 79% of them have increasing trends. This indicates that long-term contributors tend to progress towards allocating their contributions more evenly across various OSS activities. We also observe a greater proportion of contributors have decreasing trends for the number of commits, issues, issue comments, and pull request comments, while having increasing trends for code reviews. This suggests a shift towards undertaking more review duties and reducing contributions to other OSS tasks. Table A.3 shows that significant shifts of work preferences primarily occur in the middle to later stages for the long-term contributors, with few significant trends of work preferences in their early to middle stages.

4.4.3.3 Evolution of Technical Importance

A portion of long-term contributors progress towards reduced technical importance, while the majority of contributors do not demonstrate a clear trend in shifting technical importance. We observe that approximately 8% *core* contributors exhibit significant increasing trends in the evolution of their commit importance (i.e., commit_centrality), and 12% show significant decreasing trends. This indicates two different evolution patterns in the commit importance (i.e., commit_centrality): a gradual increase in the importance of their code contributions or a shift away from highly technical code contributions that modify the influential files in the repository. Period technical importance (i.e., period_centrality) measures both the volume and the importance of the commits made within a period. Similar to

our observations in the evolution of workload compositions, significant trends in period technical importance are observed only among long-term contributors who have been active in the projects for over 2.5 years. Among these contributors with significant changes in their technical importance, we observe a predominant trend toward decreasing technical importance. This finding implies a tendency of long-term contributors to shift away from intensively making highly technical code contributions. Table A.3 shows that a greater proportion of long-term contributors have decreasing period technical importance in their middle to later stages, compared to their early to middle stages.

RQ4 Summary

The Committer's and Code Reviewer's contributions are more sustainable as project development progresses, while the Issue Reporter's contributions are less sustainable. A portion of long-term contributors show a tendency towards fewer, constant, and balanced contributions, and shift away from highly technical and intensive contributions. Nevertheless, the majority of contributors do not exhibit a significant trend in their evolving OSS engagement.

Chapter 5

Discussion

In this chapter, we compare our findings with prior studies and derive implications for OSS contributors, ML project maintainers, and software engineering researchers.

5.1 Company participation in ML OSS projects

The open-source ML ecosystem on GitHub is now largely supported by technology companies through financial backing or providing professionals [72]. This trend is also evident in our subject projects: TensorFlow and Keras are driven by Google developers, PyTorch by Meta, MXNet by AWS, and ONNX by AWS, Microsoft, Meta, and other partners. This OSS collaboration dynamic introduces differences in the behavior and performance of internal contributors (i.e., paid developers from these companies) and external contributors (i.e., volunteers from the open-source community). Our contributor profiles categorize ML contributors into *workhour* and *afterhour* groups based on their work hours, differentiating between a work-like contribution manner and an after-hours contribution manner.

Our findings reveal that *Core-Workhour* contributors tend to stay longer in projects, commit more frequently, contribute more evenly across all levels of ML framework,

and engage in collaborative activities, such as reviewing pull requests and discussing other contributor's issues or pull requests. This indicates that core internal contributors may focus more on quality assurance, project management, and mentorship within ML projects, compared to core external contributors. These results align with previous studies [79] on non-ML OSS projects, which find that internal contributors are more likely to become long-term, and that core internal contributors typically contribute more frequently than core external ones. Previous research also shows that internal contributors in company-supported OSS projects often take on significant management responsibilities [106], indicating the similar role internal contributors play in both traditional OSS and ML projects.

As reported in RQ1 and RQ2, while *Core-Workhour* contributors reach slightly higher technical importance, their contributions are generally smaller and less complex than *Core-afterhour* contributors, showing that their contributions are small but influential, which is again likely oriented toward management or maintenance tasks. In contrast, *Core-afterhour* contributors tend to make larger and more complex contributions, such as changes to ML operators, while maintaining a high acceptance ratio. This finding contrasts with the results of Pinto et al. [107] that volunteers face more rejects than employees. This indicates that, within the open-source ML community, core external contributors bring strong technical expertise and benefit from feedback and guidance provided by maintainers (who are likely to be internal developers). This also points to a welcoming environment in which external contributors are actively encouraged and supported in making complex contributions.

Understanding the roles of internal and external contributors can help companies

and the open-source ML community build effective collaboration platforms to support the sustainability of their projects. For instance, ML projects could establish mentorship programs where internal contributors help newcomers understand project requirements and the best practices to contribute. Given that *core-afterhour* contributors demonstrate high-quality and highly specialized contributions, the project could introduce recognition systems, such as additional badges, to highlight impactful contributions from externals, increasing their visibility and promoting further engagement. To address retention challenges and attract skilled externals, companies could develop sponsorship platforms for contributors seeking financial support or employment, especially if they demonstrate valuable expertise. Hiring core externals who have already worked on the project could streamline recruitment and onboarding, as these contributors have demonstrated relevant skills and alignment with the company's needs. At the same time, companies should be mindful of their internal contributors' influence on the open-source ML community. The interactions and conduct of internal contributors shape perceptions of the company and the projects they lead. If internal developers appear unwelcoming or overly critical, external contributors may feel alienated and choose to redirect their efforts to other ML projects.

5.2 Core vs Peripheral Contributors In OSS Projects

Our work focuses on understanding the contributor profiles in ML OSS projects by clustering the behaviors of contributors, and as a result of clustering, we find four distinct groups that we name by core vs peripheral based on observed characteristics. In comparison, prior studies employed various approaches to specifically distinguish between core and peripheral developers, such as utilizing social-technical

networks [78].

In our results, core and peripheral contributors share some similarities with prior research on traditional OSS projects. Similar to prior research [2, 19, 108], we find that core contributors make more independent coding contributions (e.g., code change commits and add new lines of code) with large-scope file authorships, and their changes tend to be more technical and influential. Peripheral contributors tend to have a specialized and narrow focus in their coding efforts. Our findings agree with prior research on the characteristics of core and peripheral contributors in collaborative activities. For example, core contributors are more involved in project management, mentoring, and code review, while peripheral contributors participate less in collaborations [19, 78, 108].

Different from prior research, we further identify different behaviors within the *core* contributor profile and *peripheral* contributor profile in terms of work hour and after hours. For example, as reported in RQ1 and RQ2, in the core profiles, the *Core-Workhour* contributors focus more on collaborative activities, including project management, mentoring, and reviewing, whereas *Core-Afterhour* contributors focus more on making complex code contributions. This difference may reflect the different behaviors of internal and external contributors in company-supported projects, as discussed in Section 4.1. Further research could validate the behavioral differences and study the contributor role from the dual perspectives of technical and support roles, as reported by prior work [109].

Wang et al. [108] report that core contributors, in the later stages of their involvement, often shift their focus to non-technical tasks, such as participating in issue discussions and reducing their effort on technical contributions, resulting in losses in

project productivity and quality. Similar to Wang et al.'s work, we observe that core contributors move away from intensive technical contributions over time from our dataset (i.e., ML OSS projects). Further the prior research, we find that the increase in issue discussants is positively associated with project popularity.

In terms of the content of contributions, Milewicz et al. [19] find that senior contributors usually work on the implementation of core features and infrastructure, while junior and third-party contributors focus on peripheral features and bug fixes. In our study, we find a similar pattern in ML OSS projects, where core contributors frequently contribute to operation implementations, while peripheral contributors frequently contribute to user-level APIs. Future research could further integrate the taxonomy of traditional software engineering tasks (e.g., feature, corrective, perfective, and non-functional commits as reported by [110]) with ML components (e.g., user-level APIs, operation implementations, and graph-level implementation) when studying ML contributors. For example, core contributors could frequently work on fixing bugs in ML operators, while peripheral contributors could implement new user-level APIs or update the documentation for ML operators.

Finally, Yue et al. [11] report that OSS newcomers who demonstrate consistent and persistent contributions are more likely to achieve higher technical importance, while those with complex contribution patterns or long periods of inactivity are less likely to achieve so. Our study shows that core contributors in ML projects exhibit more complex contribution dynamics but also avoid long periods of inactivity, compared to peripheral contributors. Both studies emphasize the importance of persistent participation during the early stages of involvement for OSS contributors to progress within the project. This is especially critical for peripheral contributors, as their

sporadic contributions could be one of the key factors limiting their progression and potential for long-term involvement.

5.3 Implications for OSS contributors

Prior studies report that raising issue reports and participating in issue discussions are effective ways for newcomers to initiate contributions to non-ML OSS projects [111–113]. Our study on the ML OSS projects agrees with the prior studies on the non-ML OSS projects, indicating that these behaviors are generally reflected among successfully onboarded peripheral contributors in OSS projects without specific to a domain. Therefore, we encourage the newcomers to actively report any issues they encounter while using the ML project, request features they find valuable, and openly share their thoughts and discuss with core project members. In addition, our findings in RQ1 show that peripheral contributors tend to submit smaller, less complex contributions, and have a high pull request approval ratio. A prior study by Han et al. [16] finds that DL newcomers typically contribute sparse code and focus heavily on documentation, such as updating documentation or enhancing examples and tutorials with minimal functional changes. Together, these behaviors highlight practices that help newcomers onboard successfully and build confidence before tackling more core, influential code. Moreover, our findings in RQ2 and prior study by Yue et al. [11] show that maintaining persistent and continuous contributions, rather than long periods of inactivity or sudden decreases in activities, could support technical success for OSS newcomers.

Based on our findings in Ml-specific contributions in RQ2, we recommend that ML newcomers start by contributing to User-Level APIs (e.g., data processing, model

building, and other user-facing APIs). We also suggest contributors begin with straightforward software development tasks (e.g., small-scale refactoring, updating tests, and fixing documentation). These types of contributions are effective starting points, as demonstrated by the successful onboarding of *peripheral* contributors.

We suggest newcomers avoid jumping directly into complex changes, such as adding new ML operators, as these contributions often face a higher rejection ratio even for *core* contributors. As stated in Pytorch contribution guide [1], adding new ML operators proposed in recent research is not accepted unless they represent significant breakthroughs. In such cases, contributors should open an issue and discuss the proposal with project maintainers before proceeding. Overall, our study shows that the open-source ML community is a welcoming environment. Motivated newcomers should not hesitate to engage in discussions with core members and begin contributing.

In RQ1, our findings reveal that *core* contributors, regardless of their working time, often make large and complex code changes (i.e., have a higher code contribution density). However, they demonstrate a lower pull request approval ratio and pull request approval density than *peripheral* contributors. Prior research by Yu et al. [114] reports a similar finding that the addition churn of pull requests (i.e., the number of lines of code added) exhibits a significant negative correlation with pull request acceptance and a significant positive correlation with pull request latency. They find that small pull requests tend to be accepted quickly. Our study further indicates that especially for *core* contributors, even though they are working on one pull request, they are recommended to break down their task list and split a large pull request into smaller and more manageable segments, thus avoiding unintended bugs and extensive

[1] https://pytorch.org/docs/stable/community/contribution_guide.html#contribution-process

review efforts.

Furthermore, in RQ1, we identify duration, the number of authored files, and collaborations as the most important factors distinguishing core and peripheral contributors. Hence, for peripheral contributors who wish to progress to core contributors within an ML project, we suggest that they can combine various efforts, including extending their involvement in the project, taking more file authorship, and diversifying their contributions by collaborating with others. Moreover, our findings in RQ2 reveal that core contributors tend to exhibit a progression toward higher technical importance within the first 6 months of activities in a project, whereas peripheral contributors typically maintain the same level of technical importance upon joining the project. Therefore, we encourage contributors to persist in making contributions and gradually tackle more challenging tasks in order to progress toward core contributors. We conduct a trial experiment to analyze the trend of how core contributors gradually work on various tasks. We randomly sample 20 Core-Afterhour contributors and analyze their commits before they make their most important commit (i.e., highest commit centrality) by dividing these commits into halves ordered by time. We observe that the commits of adding new features increase from 12.5% to 22.5%, indicating that these contributors gradually became more competent in implementing new features (e.g., adding an isnan operator[2]). We also observe that they increasingly make commits related to environment-dependent processing (e.g., optimizing GPU kernel implementations[3]), from 12.5% to 17.5%, indicating that these contributors gradually started to handle either low-level hardware related code change or fix more complex software compatibility issues.

[2]https://github.com/apache/mxnet/commit/c92f95490e288e0658b4a349d48f95f1cf2d389d
[3]https://github.com/tensorflow/tensorflow/commit/a4bbf33e76e3de721a74230f7ea1f83e75f7c6ad

5.4 Implications for project maintainers and managers

Managing open-source ML projects could require a higher level of technical expertise, coordination, and resource management than traditional OSS projects. In ML projects, operators and algorithms may become obsolete quickly as new research emerges, which requires constant updates to stay relevant. Maintainers could establish processes for continuous monitoring of advancements, ensuring that core contributors stay informed about the latest developments. Since ML projects contain multiple ML components (e.g., the five-level ML architecture studied in Section 3.6.4) which could be frequently updated, maintainers may benefit from adopting a modular design approach. Prioritizing modularity, or assigning contributors to refactor the codebase for greater modularity, allows maintainers to facilitate convenient updates for individual components. Additionally, because ML projects run across diverse hardware and software environments, maintainers could designate core members to coordinate with upstream contributors and better manage contributions for environment-specific processing.

ML projects attract a diverse range of contributors, including a mix of researchers or data scientists with limited OSS experience and OSS community members interested in learning ML. To support effective onboarding, maintainers could implement tailored mentoring strategies. For researchers, maintainers could establish specific contribution guides and contribution templates to help them familiar with OSS workflows and community practices. Assigning mentors who are active in collaborative activities (e.g., issue or PR discussants) can further navigate researchers in community-driven development. For OSS developers, maintainers could offer low-barrier entry points, such as detailed tutorials or example notebooks, to help familiarize them with

ML concepts. Using GitHub issue labels (e.g., "good first issue") can also direct these contributors to beginner-friendly tasks and reduce their onboard effort.

As shown in our RQ3 findings, issue discussants significantly impact project popularity, as users and contributors rely on them for problem-solving and design feedback. To meet these demands, project managers should allocate sufficient resources for issue handling and train discussants to ensure timely, respectful, and constructive communication. This potentially fosters contributor engagement and retention, and enhances the project's reputation within the OSS community. Additionally, with the exploding interest in ML projects, managers should establish clear guidelines for raising issues and consider hosting regular live Q&A sessions to address questions promptly. This could reduce the burden of asynchronous communication and help prevent teams from becoming overwhelmed.

Despite peripheral contributors making fewer code contributions, they can play a vital role in other OSS activities, such as raising and resolving issues. Project maintainers should acknowledge the significance of these contributors, as highlighted in RQ3, where projects with a higher ratio of issue-reporting-focused and issue-discussing-focused contributors attract greater popularity. *Peripheral* profiles include motivated contributors who begin their contributions by addressing issues they have identified or implementing features they wish to have. However, being part of a *Peripheral* profile, their involvement in the project is typically brief, which may lead to their contributions becoming difficult to maintain. To mitigate this, project maintainers can improve the stickiness of these contributors for better document/feature maintenance. Additionally, as indicated in RQ3, projects gain more popularity with increased code review from Issue Discussants and PR Discussants. We suggest project

maintainers actively assign them to review code changes from newcomers, especially the newcomers who have previously interacted with in issue discussions. This strategy could foster a welcoming environment for new contributors. Although some of these findings may already be recognized by project maintainers and documented in contribution guides (e.g., newcomers could start by raising issue reports[4]), our insights extend beyond what is typically covered. We reveal the factors correlated with project popularity in RQ3 and the evolution of individual contributors in RQ4, which require quantitative analysis to uncover and are generally outside the scope of daily project management of maintainers.

5.5 Implications for software engineering researchers

Our research can be leveraged by future research in software engineering that aims to deeply understand ML OSS contributors and delve into their detailed characteristics. We identify four contributor profiles in popular ML libraries and describe their respective contributor characteristics. This categorization approach provides a framework for researchers to extend upon based on their dimensions of interest. For example, they might analyze the quality of code commits from various contributors using code analysis tools, and provide insights into best practices that enhance the code quality from open-source contributors. In RQ3, we find that a higher ratio of contributors focusing on raising issues and discussing issues is associated with the growth of project popularity. This finding can be extended to further investigations, such as the impact of various issue topics on a project's popularity. Our finding that increasing code reviews from Issue Discussants and PR Discussants are associated

[4]https://github.com/pytorch/ignite/blob/master/CONTRIBUTING.md

with the growth of a project's popularity can be leveraged to effectively recommend code reviewers not only to examine the quality of code based on code reviewer's expertise and workload, but also to ensure a project's long-term sustainability (e.g., welcoming and retaining newcomers). Moreover, our contributor profiles categorize ML contributors into workhour and afterhour groups, offering initial insights into the internal (i.e., tech company employees) and external (i.e., volunteers from OSS community) contributors in the open-source ML community. Future research could develop precise methods to identify internal contributors, such as checking their Github site_admin flag, checking their Github profile descriptions, or linking to LinkedIn profiles where applicable, to better understand the different roles and impacts of internal and external contributors on the ML OSS community. Internal contributors can also be further differentiated into direct employees of the company driving the project and those from partner organizations, such as upstream partners (e.g., AMD, Intel) or downstream companies using the ML library. Exploring how these groups collaboratively contribute to the ML library's growth would yield valuable insights into community dynamics. For the externals contributing to ML OSS with career development in mind, researchers could survey them to understand their motivations and needs. Meanwhile, researchers could survey project managers on the qualities they seek in potential hires. Such insight could help external contributors better align their OSS experience with career pathways in ML.

Chapter 6

Threats To Validity

In this chapter, we discuss the possible threats to the validity of our study.

Threats to Internal Validity concern the effectiveness of our analysis approaches. One internal threat to our results could be the effectiveness of the clustering techniques on grouping contributors. In this study, we employ various clustering techniques. We select the clustering result from eight clustering algorithms to identify the contributor profiles in Section 3.5. We also use hierarchical clustering to identify workload composition patterns in Section 3.6.1. To select the optimal number of clusters, we experiment with a range of number of clusters, and use the clustering evaluation metric (i.e., Silhouette score) to find the optimal one. For algorithms not requiring a predetermined cluster number, we conduct a gradient search to find the optimal parameter set that yields the highest Silhouette score. Altering the number of clusters may impact our results. To address this issue, for all the clustering results, beyond the clustering evaluation metric, we also manually investigate the datapoints in the resulting clusters to ensure the clustering yields meaningful grouping and insightful outcomes based on our designated features.

Another potential internal threat is the effectiveness of the logistic regression

model for modeling contributor profiles. We use binary logistic regression in RQ1 to identify important features for each profile, because it is a simple model that outputs the importance and direction of the associations between contributor features and profiles, which aligns with the objectives of RQ1. However, other ML algorithms could be applied in the future to further explore contributor characteristics.

Threats to External Validity concern the generalizability of our results. Our study has been conducted over 8 ML projects on Github. We also limit our study to analyzing ML library and framework projects, instead of studying all the projects on Github that are implementing ML technology or developing ML libraries. To augment the generalizability of our result, we select the ML projects created for different purposes by different organizations with different sizes to conduct our experiment. Additionally, our selected projects are mainly written in Python and C++. Including more ML frameworks that are developed in other programming languages, such as Java or Matlab, might reveal more meaningful findings regards ML contributors. However, considering that Python is the most preferable programming language for scientific computing and machine learning [115], the generalizability of our results to other ML projects is still adequate.

Threats to Construct Validity concern whether our analyses measure what we claim to analyze. In our work, we aim to identify the profiles and understand the characteristics of ML OSS contributors. Although our dataset contains only ML projects, there might still be traditional software developers with little ML knowledge. Unfortunately, we are not able to consult all contributors and ask for their expertise in developing ML software. To minimize this threat, we select the ML projects that are developing ML frameworks and libraries instead of adopting ML techniques

for different domains. We believe that there is a greater portion of ML experts in projects developing ML tools than in projects adopting ML tools. Additionally, during the calculation of the feature *number of issues solved*, we employ a keyword-based approach similar to prior studies [81, 82] to identify the issue-fixing commits or pull requests, and count the number of issues being solved accordingly. However, there might be cases that which an issue is solved without making commits or pull requests, or without mentioning the keywords. Future research is encouraged to consider such cases to further enhance the issue data quality.

Moreover, we categorize contributors into *workhour* and *afterhour* groups by clustering their most frequent contribution times. However, this classification is not exclusive—*workhour* contributors may still contribute outside normal working hours and vice versa. Additionally, it does not distinguish between company employees and volunteers, as employees may also contribute after hours and vice versa. Future research aiming specifically to study contribution times and the differences between company employees and volunteers in OSS projects is encouraged to consider more factors to enhance soundness.

Chapter 7

Conclusions and Future Work

A comprehensive understanding of contributors lays the foundation for successful open-source software (OSS) development and maintenance by promoting efficient resource allocation and collaboration. In this thesis, we conduct an empirical study aimed at advancing the understanding of contributors within the machine learning open-source software (ML OSS) community.

In the following sections, we summarize the contributions of this thesis and propose future research directions to advance our work.

7.1 Contributions

By studying 11,949 contributors from 8 popular ML libraries, we are able to identify four distinct contributor profiles with significant differences based on their work time, total commits, code contribution density, and the number of programming languages used, and reveal the most influential features associated with the profiles, such as project experience, authorship diversity, collaborations, and geographical location. We study the OSS engagement of contributors from four aspects:

workload composition, work preferences, technical importance, and ML-specific contributions, and reveal the variations in the OSS engagement across contributors in different profiles. Moreover, we explore the impact of contributor OSS engagement on project popularity and identify the important factors of workload composition and work preference associated with the increase in project popularity. The study of the evolution of contributor OSS engagement highlights that long-term contributors evolve towards fewer, constant, and balanced contribution dynamics, moving away from highly technical and intensive contributions. These findings contribute to a deeper understanding of the ML OSS ecosystem and offer practical implications for project maintainers and software engineering researchers aiming to foster a welcoming, efficient, and collaborative OSS environment. Our findings also offer suggestions for OSS contributors, including newcomers (i.e., potential contributors who have not yet onboard), peripheral contributors, and core contributors to progress within ML projects.

7.2 Future Work

This thesis provides a comprehensive understanding of contributors in ML OSS projects by analyzing their contributor profiles, OSS engagement, impact on project growth, and evolution trends. Future research can build on our findings and explore diverse perspectives to support the ML OSS community.

Understanding employees and volunteers. The ML OSS ecosystem on GitHub is increasingly driven by contributions from technology companies. Our result shows notable differences in the behaviors and contributions between company employees and volunteers. Future research could use our findings as a baseline and

develop reliable approaches to distinguish between employees and volunteers, further deepening the understanding of their roles and impact within the ML OSS community.

Enhancing human resource management system. Our results identify the important contributor engagement factors that can impact a project's popularity growth. Future research could leverage these findings to advance resource allocation management systems and code reviewer recommendation systems, aiming to optimize collaboration and promote the sustained growth of project popularity.

Understanding the role of bots in ML OSS development. In ML OSS projects, we observe that both human contributors and bots play active roles. While this thesis focuses on understanding human contributors, bots are essential for automating repetitive tasks and assisting project management. Future research could explore the behaviors of bots in greater depth and examine strategies to enhance their cooperation with human contributors.

Understanding the collaborations in the ML OSS ecosystem. This thesis examines collaborations within individual ML frameworks. Modern ML functions as a multi-layered technology stack, where interactions occur between foundational frameworks like TensorFlow or PyTorch, upstream hardware partners such as AMD and Intel, and downstream applications across multiple domains. Future research could explore collaborations across the entire ecosystem, identifying strategies to optimize the technology chain and promote effective partnerships.

Bibliography

[1] P. Guo, "How did people write machine learning code in the past?" 2018. [Online]. Available: https://cacm.acm.org/blogs/blog-cacm/230805-how-did-people-write-machine-learning-code-in-the-past/fulltext

[2] D. A. da Costa, U. Kulesza, E. Aranha, and R. Coelho, "Unveiling developers contributions behind code commits: an exploratory study," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 1152–1157.

[3] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: the apache server," in *Proceedings of the 22nd international conference on Software engineering*, 2000, pp. 263–272.

[4] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.

[5] T. Dey, A. Karnauch, and A. Mockus, "Representation of developer expertise in open source software," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 995–1007.

[6] J. E. Montandon, L. L. Silva, and M. T. Valente, "Identifying experts in software libraries and frameworks among github users," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 276–287.

[7] E. Cohen and M. P. Consens, "Large-scale analysis of the co-commit patterns of the active developers in github's top repositories," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 2018, pp. 426–436.

[8] M. El Mezouar, F. Zhang, and Y. Zou, "An empirical study on the teams structures in social coding using github projects," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3790–3823, 2019.

[9] L. Bao, X. Xia, D. Lo, and G. C. Murphy, "A large scale study of long-time contributor prediction for github projects," *IEEE Transactions on Software Engineering*, vol. 47, no. 6, pp. 1277–1298, Jun. 2021.

[10] M. Zhou and A. Mockus, "Who will stay in the floss community? modeling participant's initial behavior," *IEEE Transactions on Software Engineering*, vol. 41, no. 1, pp. 82–99, 2015.

[11] Y. Yue, Y. Wang, and D. Redmiles, "Off to a good start: Dynamic contribution patterns and technical success in an oss newcomer's early career," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 529–548, 2022.

[12] S. Balali, I. Steinmacher, U. Annamalai, A. Sarma, and M. A. Gerosa, "Newcomers' barriers... is that all? an analysis of mentors' and newcomers' barriers

in oss projects," *Computer Supported Cooperative Work (CSCW)*, vol. 27, pp. 679–714, 2018.

[13] C. J. Cai and P. J. Guo, "Software developers learning machine learning: Motivations, hurdles, and desires," in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2019, pp. 25–34.

[14] C. Hill, R. Bellamy, T. Erickson, and M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2016, pp. 162–170.

[15] F. Ishikawa and N. Yoshioka, "How do engineers perceive difficulties in engineering of machine-learning systems? - questionnaire survey," in *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, 2019, pp. 2–9.

[16] J. Han, J. Zhang, D. Lo, X. Xia, S. Deng, and M. Wu, "Understanding newcomers' onboarding process in deep learning projects," *IEEE Transactions on Software Engineering*, pp. 1–18, 2024.

[17] G. Robles and J. M. Gonzalez-Barahona, "Contributor turnover in libre software projects," in *Open Source Systems*, E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, and G. Succi, Eds. Boston, MA: Springer US, 2006, pp. 273–286.

[18] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *Proceedings of the international workshop on Principles of software evolution*, 2002, pp. 76–85.

[19] R. Milewicz, G. Pinto, and P. Rodeghero, "Characterizing the roles of contributors in open-source scientific software projects," 03 2019.

[20] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 112–123.

[21] I. Steinmacher, I. Wiese, A. P. Chaves, and M. A. Gerosa, "Why do newcomers abandon open source software projects?" in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2013, pp. 25–32.

[22] T. Wang, Y. Zhang, G. Yin, Y. Yu, and H. Wang, "Who will become a long-term contributor? a prediction model based on the early phase behaviors," in *Proceedings of the 10th Asia-Pacific Symposium on Internetware*, 2018, pp. 1–10.

[23] A. M. Dakhel, M. C. Desmarais, and F. Khomh, "Dev2vec: Representing domain expertise of developers in an embedding space," *Information and Software Technology*, vol. 159, p. 107218, Jul. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2023.107218

[24] M. Ahasanuzzaman, G. A. Oliva, and A. E. Hassan, "Using knowledge units of programming languages to recommend reviewers for pull requests: an empirical study," *Empirical Software Engineering*, vol. 29, no. 1, p. 33, 2024.

[25] J. E. Montandon, L. L. Silva, and M. T. Valente, "Identifying experts in software libraries and frameworks among github users," *CoRR*, vol. abs/1903.08113, 2019. [Online]. Available: http://arxiv.org/abs/1903.08113

[26] W. Yang, M. Pan, Y. Zhou, and Z. Huang, "Developer portraying: A quick approach to understanding developers on oss platforms," *Information and Software Technology*, vol. 125, p. 106336, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920301038

[27] M. Cataldo and S. Nambiar, "The impact of geographic distribution and the nature of technical coupling on the quality of global software development projects," *Journal of Software: Evolution and Process*, vol. 24, pp. 153–168, 03 2012.

[28] M. Claes, M. V. Mäntylä, M. Kuutila, and B. Adams, "Do programmers work at night or during the weekend?" in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 705–715.

[29] J. C. Munson and S. G. Elbaum, "Code churn: A measure for estimating the impact of code change," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE, 1998, pp. 24–31.

[30] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE transactions on software engineering*, vol. 37, no. 6, pp. 772–787, 2010.

[31] A. A. Bangash, H. Sahar, S. Chowdhury, A. W. Wong, A. Hindle, and K. Ali, "What do developers know about machine learning: A study of ml discussions on stackoverflow," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 260–264.

[32] M. J. Islam, H. A. Nguyen, R. Pan, and H. Rajan, "What do developers ask about ml libraries? a large-scale study using stack overflow," *arXiv preprint arXiv:1906.11940*, 2019.

[33] "Github rest api," https://docs.github.com/en/rest, accessed: 2024-04-09.

[34] 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*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 908–911. [Online]. Available: https://doi.org/10.1145/3236024.3264598

[35] X. Sun, T. Zhou, G. Li, J. Hu, H. Yang, and B. Li, "An empirical study on real bugs for machine learning programs," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 348–357.

[36] Z. Wang, K. Liu, J. Li, Y. Zhu, and Y. Zhang, "Various frameworks and libraries of machine learning and deep learning: a survey," *Archives of computational methods in engineering*, pp. 1–24, 2019.

[37] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 510–520.

[38] T. Zhang, C. Gao, L. Ma, M. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 104–115.

[39] H. Jebnoun, H. Ben Braiek, M. M. Rahman, and F. Khomh, "The scent of deep learning code: An empirical study," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 420–430.

[40] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 1110–1121.

[41] J. Han, S. Deng, D. Lo, C. Zhi, J. Yin, and X. Xia, "An empirical study of the dependency networks of deep learning libraries," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 868–878.

[42] J. Han, E. Shihab, Z. Wan, S. Deng, and X. Xia, "What do programmers discuss about deep learning frameworks," *Empirical Software Engineering*, vol. 25, pp. 2694–2747, 2020.

[43] M. J. Islam, R. Pan, G. Nguyen, and H. Rajan, "Repairing deep neural networks: Fix patterns and challenges," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 1135–1146.

[44] S. Wang, N. Shrestha, A. K. Subburaman, J. Wang, M. Wei, and N. Nagappan, "Automatic unit test generation for machine learning libraries: How far are we?" in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1548–1560.

[45] G. Long and T. Chen, "On reporting performance and accuracy bugs for deep learning frameworks: An exploratory study from github," in *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, 2022, pp. 90–99.

[46] M. Dilhara, A. Ketkar, and D. Dig, "Understanding software-2.0: A study of machine learning library usage and evolution," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–42, 2021.

[47] X. Jiang, X. Zhang, and X. Peng, "Flet-edge: A full life-cycle evaluation tool for deep learning framework on the edge," in *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2023, pp. 680–687.

[48] J. Gesi, S. Liu, J. Li, I. Ahmed, N. Nagappan, D. Lo, E. S. de Almeida, P. S. Kochhar, and L. Bao, "Code smells in machine learning systems," *arXiv preprint arXiv:2203.00803*, 2022.

[49] X. Tan, K. Gao, M. Zhou, and L. Zhang, "An exploratory study of deep learning supply chain," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 86–98.

[50] L. N. Tidjon, B. Rombaut, F. Khomh, A. E. Hassan *et al.*, "An empirical study of library usage and dependency in deep learning frameworks," *arXiv preprint arXiv:2211.15733*, 2022.

[51] N. S. Harzevili, J. Shin, J. Wang, S. Wang, and N. Nagappan, "Characterizing and understanding software security vulnerabilities in machine learning libraries," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 2023, pp. 27–38.

[52] T. Kavarakuntla, L. Han, H. Lloyd, A. Latham, A. Kleerekoper, and S. B. Akintoye, "A generic performance model for deep learning in a distributed environment," *IEEE Access*, 2024.

[53] N. S. Harzevili, J. Shin, J. Wang, S. Wang, and N. Nagappan, "Automatic static vulnerability detection for machine learning libraries: Are we there yet?" in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 795–806.

[54] R. Mo, Y. Zhang, Y. Wang, S. Zhang, P. Xiong, Z. Li, and Y. Zhao, "Exploring the impact of code clones on deep learning software," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 1–34, 2023.

[55] S. S. Khairunnesa, S. Ahmed, S. M. Imtiaz, H. Rajan, and G. T. Leavens, "What kinds of contracts do ml apis need?" *Empirical Software Engineering*, vol. 28, no. 6, p. 142, 2023.

[56] J. Shin, M. Wei, J. Wang, L. Shi, and S. Wang, "The good, the bad, and the missing: Neural code generation for machine learning tasks," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 2, pp. 1–24, 2023.

[57] A. Narayanan, J. Wang, L. Shi, M. Wei, S. Wang *et al.*, "Automatic unit test generation for deep learning frameworks based on api knowledge," *arXiv preprint arXiv:2307.00404*, 2023.

[58] P. L. Foalem, F. Khomh, and H. Li, "Studying logging practice in machine learning-based applications," *Information and Software Technology*, vol. 170, p. 107450, 2024.

[59] N. Louloudakis, P. Gibson, J. Cano, and A. Rajan, "Fault localization for buggy deep learning framework conversions in image recognition," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 1795–1799.

[60] A. Ghadesi, M. Lamothe, and H. Li, "What causes exceptions in machine learning applications? mining machine learning-related stack traces on stack overflow," *arXiv preprint arXiv:2304.12857*, 2023.

[61] Z. Zhao, B. Kou, M. Y. Ibrahim, M. Chen, and T. Zhang, "Knowledge-based version incompatibility detection for deep learning," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 708–719.

[62] Y. Zhou, Y. Qiao, and T. Xu, "Exploring the characteristics of popular deep learning github repositories," in *2023 International Conference on Intelligent Computing and Next Generation Networks (ICNGN)*. IEEE, 2023, pp. 1–6.

[63] H. Lei, S. Zhang, J. Wang, G. Xiao, Y. Liu, and Y. Sui, "Why do deep learning projects differ in compatible framework versionsf an exploratory study," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 509–520.

[64] J. Han, J. Liu, D. Lo, C. Zhi, Y. Chen, and S. Deng, "On the sustainability of deep learning projects: Maintainers' perspective," *Journal of Software: Evolution and Process*, p. e2645, 2023.

[65] J. Wang, G. Xiao, S. Zhang, H. Lei, Y. Liu, and Y. Sui, "Compatibility issues in deep learning systems: Problems and opportunities," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 476–488.

[66] M. M. Morovati, A. Nikanjam, F. Tambon, F. Khomh, and Z. M. Jiang, "Bug characterization in machine learning-based systems," *Empirical Software Engineering*, vol. 29, no. 1, p. 14, 2024.

[67] C. S. Xia, S. Dutta, S. Misailovic, D. Marinov, and L. Zhang, "Balancing effectiveness and flakiness of non-deterministic machine learning tests," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1801–1813.

[68] E. Panourgia, T. Plessas, I. Balampanis, and D. Spinellis, "Good tools are half the work: Tool usage in deep learning projects," 2024. [Online]. Available: https://arxiv.org/abs/2310.19124

[69] C. Liu, R. Cai, Y. Zhou, X. Chen, H. Hu, and M. Yan, "Understanding the implementation issues when using deep learning frameworks," *Information and Software Technology*, vol. 166, p. 107367, 2024.

[70] Z. Lai, H. Chen, R. Sun, Y. Zhang, M. Xue, and D. Yuan, "On security weaknesses and vulnerabilities in deep learning systems," *arXiv preprint arXiv:2406.08688*, 2024.

[71] H. Guan, Y. Xiao, J. Li, Y. Liu, and G. Bai, "A comprehensive study of real-world bugs in machine learning model optimization," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 147–158.

[72] H. B. Braiek, F. Khomh, and B. Adams, "The open-closed principle of modern machine learning frameworks," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 353–363.

[73] 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*, 2020, pp. 431–442.

[74] T. Zhang, C. Gao, L. Ma, M. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, 2019, pp. 104–115.

[75] A. Abdellatif, M. Wessel, I. Steinmacher, M. A. Gerosa, and E. Shihab, "Bothunter: an approach to detect software bots in github," ser. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 6–17. [Online]. Available: https://doi.org/10.1145/3524842.3527959

[76] M. V. Bertoncello, G. Pinto, I. S. Wiese, and I. Steinmacher, "Pull requests or commits? which method should we use to study contributors' behavior?" in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 592–601.

[77] S. Noei, H. Li, S. Georgiou, and Y. Zou, "An empirical study of refactoring rhythms and tactics in the software development process," *IEEE Transactions on Software Engineering*, vol. 49, no. 12, pp. 5103–5119, dec 2023.

[78] M. Joblin, S. Apel, C. Hunsen, and W. Mauerer, "Classifying developers into core and peripheral: An empirical study on count and network metrics," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 164–174.

[79] Y. Zhang, M. Qin, K.-J. Stol, M. Zhou, and H. Liu, "How are paid and volunteer open source developers different? a study of the rust project," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[80] J. T. Liang, T. Zimmermann, and D. Ford, "Towards mining oss skills from github activity," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, 2022, pp. 106–110.

[81] B. A. Muse, M. M. Rahman, C. Nagy, A. Cleve, F. Khomh, and G. Antoniol, "On the prevalence, impact, and evolution of sql code smells in data-intensive systems," in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp. 327–338.

[82] F. Wen, C. Nagy, M. Lanza, and G. Bavota, "Quick remedy commits and their impact on mining software repositories," *Empirical Software Engineering*, vol. 27, pp. 1–31, 2022.

[83] J. H. Zar, "Spearman rank correlation," *Encyclopedia of biostatistics*, vol. 7, 2005.

[84] E. Noei, F. Zhang, and Y. Zou, "Too many user-reviews! what should app developers look at first?" *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 367–378, 2021.

[85] J. Miles, "R-squared, adjusted r-squared," *Encyclopedia of statistics in behavioral science*, 2005.

[86] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[87] M. Christ, A. W. Kempa-Liehr, and M. Feindt, "Distributed and parallel time series feature extraction for industrial big data applications," *arXiv preprint arXiv:1610.07717*, 2016.

[88] J. Chen, Y. Liang, Q. Shen, J. Jiang, and S. Li, "Toward understanding deep learning framework bugs," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 1–31, 2023.

[89] S. Noei, H. Li, and Y. Zou, "Detecting refactoring commits in machine learning python projects: A machine learning-based approach," *ACM Transactions on Software Engineering and Methodology*, 2024.

[90] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[91] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," pp. 1–22, 2010. [Online]. Available: https://www.jstatsoft.org/v33/i01/

[92] J. Huang and C. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[93] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, pp. 1–13, 2020.

[94] V. Bewick, L. Cheek, and J. Ball, "Statistics review 14: Logistic regression," *Critical care*, vol. 9, no. 1, pp. 1–7, 2005.

[95] D. S. Moore, "Generalized inverses, wald's method, and the construction of chi-squared tests of fit," *Journal of the American Statistical Association*, vol. 72, no. 357, pp. 131–137, 1977.

[96] H. Y. Toda and T. Yamamoto, "Statistical inference in vector autoregressions with possibly integrated processes," *Journal of econometrics*, vol. 66, no. 1-2, pp. 225–250, 1995.

[97] S. Hassan, C. Tantithamthavorn, C.-P. Bezemer, and A. E. Hassan, "Studying the dialogue between users and developers of free apps in the google play store," *Empirical Software Engineering*, vol. 23, pp. 1275–1312, 2018.

[98] N. Nachar *et al.*, "The mann-whitney u: A test for assessing whether two independent samples come from the same distribution," *Tutorials in quantitative Methods for Psychology*, vol. 4, no. 1, pp. 13–20, 2008.

[99] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[100] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling,"

The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, vol. 50, no. 302, pp. 157–175, Jul. 1900. [Online]. Available: https://doi.org/10.1080/14786440009463897

[101] J. Cohen, *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[102] J. Pinheiro and D. Bates, *Mixed-effects models in S and S-PLUS*. Springer science & business media, 2006.

[103] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[104] S. Nakagawa, H. Schielzeth, and R. B. O'Hara, "A general and simple method for obtaining r2 from generalized linear mixed-effects models. methods ecol evol 4: 133–142," 2013.

[105] D. R. Cox and A. Stuart, "Some quick sign tests for trend in location and dispersion," *Biometrika*, vol. 42, no. 1/2, pp. 80–95, 1955.

[106] L. F. Dias, C. Barbosa, G. Pinto, I. Steinmacher, B. Fonseca, M. Ribeiro, C. Treude, and D. A. Da Costa, "Refactoring from 9 to 5? what and when employees and volunteers contribute to oss," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2020, pp. 1–5.

[107] G. Pinto, L. F. Dias, and I. Steinmacher, "Who gets a patch accepted first? comparing the contributions of employees and volunteers," in *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2018, pp. 110–113.

[108] Z. Wang, Y. Feng, Y. Wang, J. A. Jones, and D. Redmiles, "Unveiling elite developers' activities in open source projects," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 3, pp. 1–35, 2020.

[109] J. Cheng and J. L. Guo, "Activity-based analysis of open source software contributors: Roles and dynamics," in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2019, pp. 11–18.

[110] G. E. dos Santos and E. Figueiredo, "Commit classification using natural language processing: Experiments over labeled datasets." in *CIbSE*, 2020, pp. 110–123.

[111] X. Tan, M. Zhou, and Z. Sun, "A first look at good first issues on github," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 398–409.

[112] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in oss community," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 518–528.

[113] W. Xiao, H. He, W. Xu, Y. Zhang, and M. Zhou, "How early participation determines long-term sustained activity in github projects?" in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 29–41.

[114] Y. Yu, G. Yin, T. Wang, C. Yang, and H. Wang, "Determinants of pull-based development in the context of continuous integration," *Science China Information Sciences*, vol. 59, pp. 1–14, 2016.

[115] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *Information*, vol. 11, no. 4, p. 193, 2020.

[116] J. Geldenhuys, "Finding the core developers," in *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2010, pp. 447–450.

[117] H. A. Çetin and E. Tüzün, "Analyzing developer contributions using artifact traceability graphs," *Empirical Software Engineering*, vol. 27, no. 3, pp. 1–49, 2022.

[118] S. S. SHAPIRO and M. B. WILK, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3-4, pp. 591–611, dec 1965. [Online]. Available: https://doi.org/10.1093/biomet/52.3-4.591

[119] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," *Data mining and knowledge discovery*, vol. 2, no. 3, pp. 283–304, 1998.

[120] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[121] M. Syakur, B. Khotimah, E. Rochman, and B. D. Satoto, "Integration k-means clustering method and elbow method for identification of the best customer

profile cluster," in *IOP conference series: materials science and engineering*, vol. 336, no. 1. IOP Publishing, 2018, p. 012017.

[122] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[123] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, "Machine learning applications in cancer prognosis and prediction," *Computational and Structural Biotechnology Journal*, vol. 13, pp. 8–17, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2001037014000464

[124] S. Shafiq, A. Mashkoor, C. Mayr-Dorn, and A. Egyed, "Machine learning for software engineering: A systematic mapping," *arXiv preprint arXiv:2005.13299*, 2020.

[125] B. Trinkenreich, M. Guizani, I. Wiese, A. Sarma, and I. Steinmacher, "Hidden figures: Roles and pathways of successful oss contributors," *Proc. ACM Hum.-Comput. Interact.*, vol. 4, no. CSCW2, oct 2020. [Online]. Available: https://doi.org/10.1145/3415251

[126] G. Robles and J. M. Gonzalez-Barahona, "Contributor turnover in libre software projects," in *IFIP International Conference on Open Source Systems*. Springer, 2006, pp. 273–286.

[127] C. Cheng, B. Li, Z.-Y. Li, Y.-Q. Zhao, and F.-L. Liao, "Developer role evolution in open source software ecosystem: An explanatory study on gnome," *Journal of computer science and technology*, vol. 32, no. 2, pp. 396–414, 2017.

[128] I. El Asri, N. Kerzazi, L. Benhiba, and M. Janati, "From periphery to core: a temporal analysis of github contributors' collaboration network," in *Working Conference on Virtual Enterprises*. Springer, 2017, pp. 217–229.

[129] J. Geldenhuys, "Finding the core developers," in *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2010, pp. 447–450.

[130] C. Binnewies, S. Sonnentag, and E. J. Mojza, "Recovery during the weekend and fluctuations in weekly job performance: A week-level study examining intra-individual relationships," *Journal of Occupational and Organizational Psychology*, vol. 83, no. 2, pp. 419–441, 2010.

[131] J. Zhang, Y. Chen, Q. Gong, X. Wang, A. Y. Ding, Y. Xiao, and P. Hui, "Understanding the working time of developers in it companies in china and the united states," *IEEE Software*, vol. 38, no. 2, pp. 96–106, 2020.

[132] A. J. Scott and M. Knott, "A cluster analysis method for grouping means in the analysis of variance," *Biometrics*, pp. 507–512, 1974.

[133] E. G. Jelihovschi, J. C. Faria, and I. B. Allaman, "Scottknott: a package for performing the scott-knott clustering algorithm in r," *TEMA (São Carlos)*, vol. 15, pp. 3–17, 2014.

[134] M. Wiesche and H. Krcmar, "The relationship of personality models and development tasks in software engineering," in *Proceedings of the 52nd ACM conference on Computers and people research*, 2014, pp. 149–161.

[135] A. Jain, A. A. Awan, Q. Anthony, H. Subramoni, and D. K. D. Panda, "Performance characterization of dnn training using tensorflow and pytorch on modern clusters," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 1–11.

[136] O.-C. Novac, M. C. Chirodea, C. M. Novac, N. Bizon, M. Oproescu, O. P. Stan, and C. E. Gordan, "Analysis of the application efficiency of tensorflow and pytorch in convolutional neural network," *Sensors*, vol. 22, no. 22, p. 8872, 2022.

[137] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*, 4th ed. New York: Springer, 2002, iSBN 0-387-95457-0. [Online]. Available: https://www.stats.ox.ac.uk/pub/MASS4/

[138] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[139] P. Bruce, A. Bruce, and P. Gedeck, *Practical statistics for data scientists: 50+ essential concepts using R and Python*. O'Reilly Media, 2020.

[140] T. Warren Liao, "Clustering of time series data—a survey," *Pattern Recognition*, vol. 38, no. 11, pp. 1857–1874, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320305001305

[141] X. Zhang, Y. Yu, G. Gousios, and A. Rastogi, "Pull request decisions explained: An empirical overview," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 849–871, 2023.

[142] M. C. O. Silva, M. T. Valente, and R. Terra, "Does technical debt lead to the rejection of pull requests?" *arXiv preprint arXiv:1604.01450*, 2016.

[143] J. Fox and S. Weisberg, *An R Companion to Applied Regression*, 3rd ed. Thousand Oaks CA: Sage, 2019. [Online]. Available: https://socialsciences.mcmaster.ca/jfox/Books/Companion/

[144] A. Vogelsang and M. Borg, "Requirements engineering for machine learning: Perspectives from data scientists," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2019, pp. 245–251.

[145] A. Serban and J. Visser, "An empirical study of software architecture for machine learning," *arXiv preprint arXiv:2105.12422*, vol. 39, 2021.

[146] E. Breck, N. Polyzotis, S. Roy, S. Whang, and M. Zinkevich, "Data validation for machine learning." in *MLSys*, 2019.

[147] D. E. Rzig, F. Hassan, C. Bansal, and N. Nagappan, "Characterizing the usage of ci tools in ml projects," in *Proceedings of the 16th ACM/IEEE International*

Symposium on Empirical Software Engineering and Measurement, 2022, pp. 69–79.

[148] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," *Advances in neural information processing systems*, vol. 28, 2015.

[149] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches and perceived productivity," *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1178–1193, 2017.

[150] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," no. 1, 2017.

[151] ——, "The impact of automated parameter optimization for defect prediction models," 2018.

[152] A. Sardá-Espinosa, "Time-Series Clustering in R Using the dtwclust Package," *The R Journal*, vol. 11, no. 1, pp. 22–43, 2019. [Online]. Available: https://doi.org/10.32614/RJ-2019-023

[153] L. Ghadhab, I. Jenhani, M. W. Mkaouer, and M. B. Messaoud, "Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model," *Information and Software Technology*, vol. 135, p. 106566, 2021.

Appendix A

Supplementary Materials and Results

A.1 Pytorch PR Merge Status

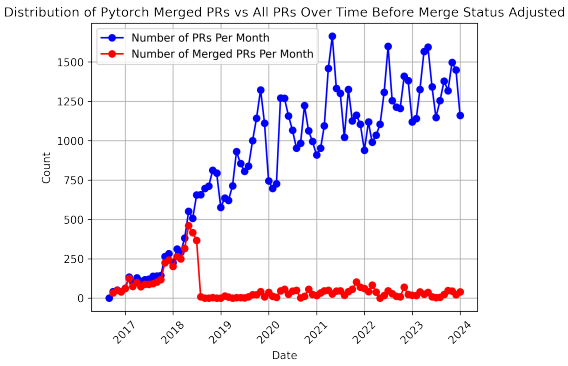


Figure A.1: The number of pull requests raised in PyTorch per month and the number of pull requests merged, only based on GitHub's pull request merged status.

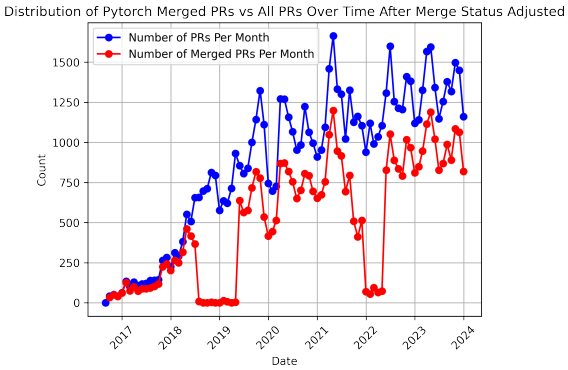


Figure A.2: The number of pull requests raised in PyTorch per month and the number of pull requests merged, based on both GitHub's pull request merged status and the 'Merged' label.

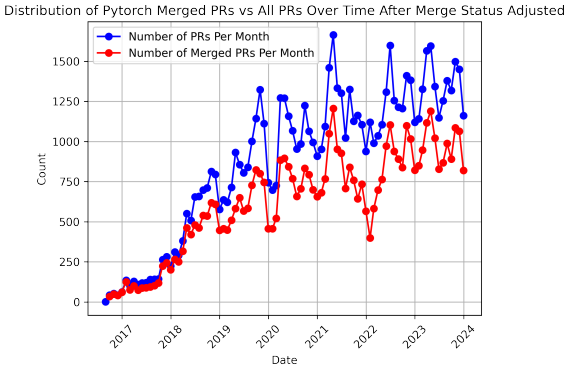


Figure A.3: The number of pull requests raised in PyTorch per month and the number of pull requests merged, based on the combination of three methods: GitHub's pull request merged status, the 'Merged' label, and the presence of commit SHA in PR closing event.

As shown in Figure A.1, prior to July 2018, Pytorch set accepted pull requests to 'Merged' status, but after July 2018, very few pull requests (i.e., less than 100 each month) were set to 'Merged' status.A comment in pull request[1] mentions Pytorch uses facebook-github-bot to copy the approved commits into an internal repository, and

[1]https://github.com/pytorch/pytorch/pull/8581

another comment in pull request[2] mentions that facebook-github-bot automatically closes the PR on the Github repository after merging the changes into the internal repository. We find that most pull requests are closed by the bot, with the status always set to 'Closed,' regardless of whether the pull request was merged. However, contributors with merge permissions (e.g., pull request[3]) could bypass the bot and set the pull request close status to 'Merged' instead of 'Closed'.

As shown in Figure A.2, PyTorch assigns a 'Merged' label to the approved and merged pull requests starting in May 2019. This begins with pull request[4] on October 6, 2020, the facebook-github-bot starts assigning a 'Merged' label when closing pull requests that are merged. Additionally, the 'Merged' label is retroactively applied to pull requests closed before this date, up until July 2019 (note: some older pull requests are labeled 'merged' with a lowercase, which no longer exists in the Pytorch label system and cannot be collected into our dataset). Therefore, we consider both the 'Merged' status and the 'Merged' label as the indication of acceptance for PyTorch pull requests before July 2018 and after May 2019.

However, between July 2018 and May 2019, and between December 2021 and May 2022, few pull requests were set to 'Merged' status or assigned a 'Merged' label, even though the pull request submission rate was similar to other periods (as shown by the two intervals with inconsistent PR acceptance in Figure A.2). We manually investigate the comments under the pull requests submitted during this period, and find that many without a 'Merged' label or 'Merged' status are actually accepted. We could not find further documentation or official announcements from the PyTorch team about these two periods. To address these false negatives, we follow the approach

[2]https://github.com/pytorch/pytorch/pull/8591
[3]https://github.com/pytorch/pytorch/pull/135068
[4]https://github.com/pytorch/pytorch/pull/45773

proposed by Bertoncello et al. [76], which reports that some projects include the commit SHA in the merged PRs' close event. We find that PyTorch also applies this practice. Therefore, we combine three methods to identify merged PRs in PyTorch: GitHub's 'Merged' status, the 'Merged' label assigned by project maintainers, and PRs closed with a commit SHA. As shown in Figure A.3, this approach results in a relatively consistent PR approval rate aligned with PR submission rates across all time periods.

A.2 RQ1 Supplementary Result

Table A.1: Result of Mann-Whitney U tests and effect sizes for contributor features.

Contributor Features	Group1: Core Group2: Peripheral	Group1: Core-Afterhour Group2: Core-Workhour	Group1: Peripheral-Afterhour Group2: Peripheral-Workhour
Timezone	small (-0.238)	small (0.319)	small (0.203)
Worktime	not significant	small (-0.177)	small (-0.148)
Duration	large (0.555)	negligible (-0.143)	not significant
Followers	negligible (0.081)	not significant	not significant
Total Commits	large (0.769)	small (-0.188)	not significant
Collaborations	medium (0.466)	not significant	negligible (0.062)
Authored Files	large (0.924)	negligible (-0.109)	not significant
Languages	large (0.856)	not significant	negligible (0.023)
Commit Rate	negligible (-0.03)	negligible (-0.077)	not significant
Code Commits	large (0.787)	small (-0.182)	not significant
Code Commit Rate	small (0.195)	negligible (-0.078)	not significant
Non Code Commits	medium (0.453)	small (-0.164)	not significant
Non Code Commit Rate	small (0.262)	negligible (-0.126)	not significant
Code Contribution	large (0.963)	negligible (-0.111)	not significant
Code Contribution Rate	large (0.657)	not significant	not significant
Code Contribution Density	large (0.906)	negligible (0.069)	not significant
Total Issues	small (0.231)	negligible (0.061)	negligible (0.071)
Issue Participated	medium (0.444)	negligible (-0.056)	negligible (0.038)
Issue Comments Received	small (0.172)	negligible (0.067)	negligible (0.072)
Issue Solved	small (0.224)	not significant	negligible (0.025)
Issue Solving Ratio	small (0.178)	not significant	negligible (0.024)
Issue Solving Rate	small (0.189)	not significant	negligible (0.024)
Issue Contribution	small (0.298)	not significant	negligible (0.073)
Issue Contribution Rate	small (0.151)	negligible (0.08)	negligible (0.077)
Total Pull Requests	medium (0.405)	not significant	negligible (0.076)
PR Participated	large (0.531)	negligible (-0.106)	not significant
PR Reviewed	medium (0.413)	negligible (-0.105)	not significant
PR Merged	medium (0.385)	not significant	negligible (0.095)
PR Approval Ratio	small (-0.333)	negligible (0.072)	negligible (0.071)
PR Comments Received	small (0.189)	small (0.174)	negligible (0.085)
PR Contribution	large (0.492)	not significant	negligible (0.077)
PR Contribution Rate	small (-0.156)	negligible (0.074)	negligible (0.069)

Groups with no significant difference (i.e., null hypothesis is accepted) are indicated with 'not significant'. Groups that are significantly different examined by Mann-Whitney U test are indicated with Cliff's delta value and its interpretation (i.e., negligible, small, medium, or large). A positive value means group1 is greater than group2, and vice versa. Three columns from left to right correspond to Hypothesis 1, 2, and 3, respectively.

A.3 RQ2 and RQ3 Supplementary Result

Figure A.4 shows an example of a newcomer being encouraged by Issue Discussants to become a contributor in TensorFlow. Issue Discussant A and Issue Discussant B are two existing contributors in TensorFlow with the workload composition pattern of Issue Discussant during the period from Oct 27th, 2017, to January 25th, 2018. In this period, a newcomer raises one's first two issue reports (i.e., Issue 1 and Issue 2) on TensorFlow. Issue Discussant A actively engages in discussions for Issue 1 and Issue Discussant B participates in the discussions for both issues. In both issue reports, Issue Discussant B encourages the newcomer to make contributions. The newcomer submits one's first pull request to address the observed issue (i.e., Pull Request 1), which is then reviewed and approved by Issue Discussant B. This newcomer eventually becomes a long-contributor to Tensorflow and remains active in the project for more than 4 years.

Figure A.5(a) shows a commit with high commit importance (i.e., *commit centrality* = 0.018). This commit fixes the memory leak in the tensors, which is the fundamental functionality of the deep learning frameworks. The two files being modified in this commit contain the core and kernel functions, and changing these files can affect all computations happening within the framework. Figure A.5(b) shows a commit with relatively low commit importance (i.e., *commit centrality* = 1.22e-24). This commit makes changes to the demo code for a specific implementation of TensorFlow Lite on an Android device's camera. The impact of such modifications is often limited to the specific functionalities and unlikely to cause large-scale disruptions.

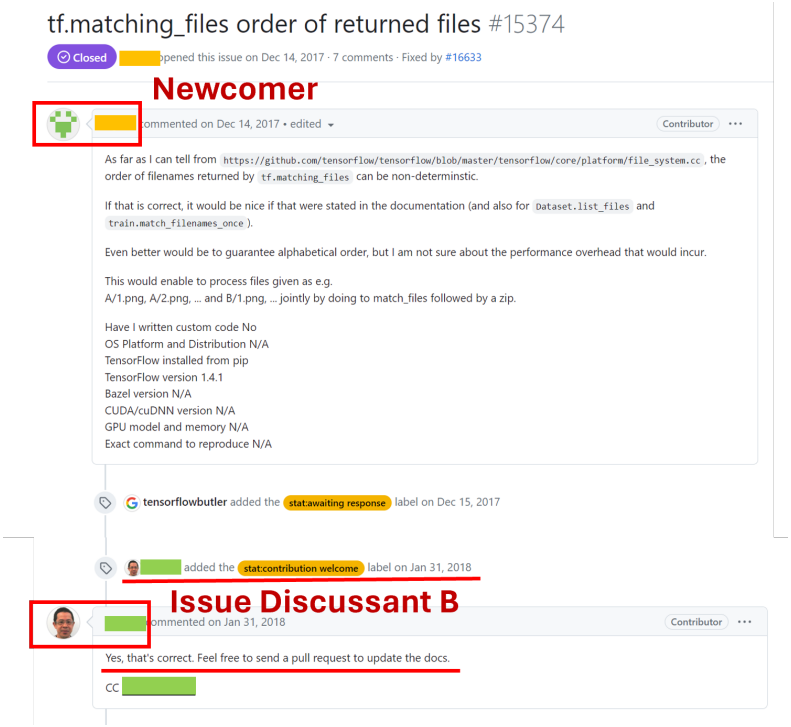
Note that the figures in this appendix have been cropped to omit unimportant

details. The original content can be found on TensorFlow GitHub repository [5][6][7][8][9].

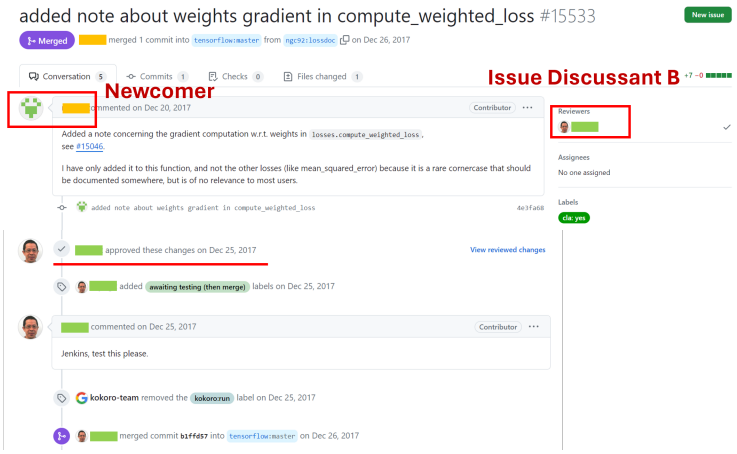
[5]https://github.com/tensorflow/tensorflow/issues/15046
[6]https://github.com/tensorflow/tensorflow/issues/15374
[7]https://github.com/tensorflow/tensorflow/pull/15533
[8]https://github.com/tensorflow/tensorflow/commit/2e8726c84a70ff962e11fac26472603aa92cc94e
[9]https://github.com/tensorflow/tensorflow/commit/4b7511f4ecd6d0bd491ec557fe05fdfe731ecdae


stop gradients for weights in tf.losses #15046
Closed
Newcomer
Contributor
In the case that the weights given to tf.losses.* depend in some way on the model parameters,
the derivative of that loss also calculated with respect to the weights.
(Stupid) minimal example:
import tensorflow as tf
x = tf.constant(0)
w = tf.get_variable(name="W", shape=(), initializer=tf.zeros_initializer())
L = tf.losses.mean_squared_error(x, x, weights=w)
tf.train.AdamOptimizer().compute_gradients(L)
results in
[(<tf.Tensor 'gradients/mean_squared_error/Mul_grad/tuple/control_dependency_1:0' shape=() dtype=float32>,
<tf.Variable 'W:0' shape=() dtype=float32_ref>)]
I would expect the weights to be considered constant for the calculation of a loss. In case you agree with me, I can make a PR that adds stop_gradient around the weights parameter.
System information
Have I written custom code (as opposed to using a stock example script provided in TensorFlow): No
OS Platform and Distribution (e.g., Linux Ubuntu 16.04): N/A
TensorFlow installed from (source or binary): N/A
TensorFlow version (use command below): N/A
Python version: N/A
Bazel version (if compiling from source): N/A
GCC/Compiler version (if compiling from source): N/A
CUDA/cuDNN version: N/A
GPU model and memory: N/A
Exact command to reproduce: N/A
Issue Discussant A
Member
Perhaps you'd better to use tf.constant for weight... instead of a variable.
Contributor
Author
In my case the weights are not constant, but calculated in the model. The above example is only supposed to be a minimal example to demonstrate the problem.
Member
You can convert a variable to tensor when used, for example, weights_ = weights + 0 or weights_ = weights * 1, and then pass weights_ to mean_squared_error.
Issue Discussant B
Contributor
Right, I also disagree that we should stop the gradients. I understand where you're coming from, but I think this would be confusing non-transparent behavior for people. Also, once you stop the gradient within the loss it's not possible to go back to previous behavior.
completed
on Dec 6, 2017
Contributor
Author
Then it would make sense to include this in the documentation, making this behaviour deliberate instead of a design accident.
Contributor
Sounds good. Feel free to send a PR
added note about weights gradient in compute_weighted_loss #15533
Merged

((a)) Issue 1

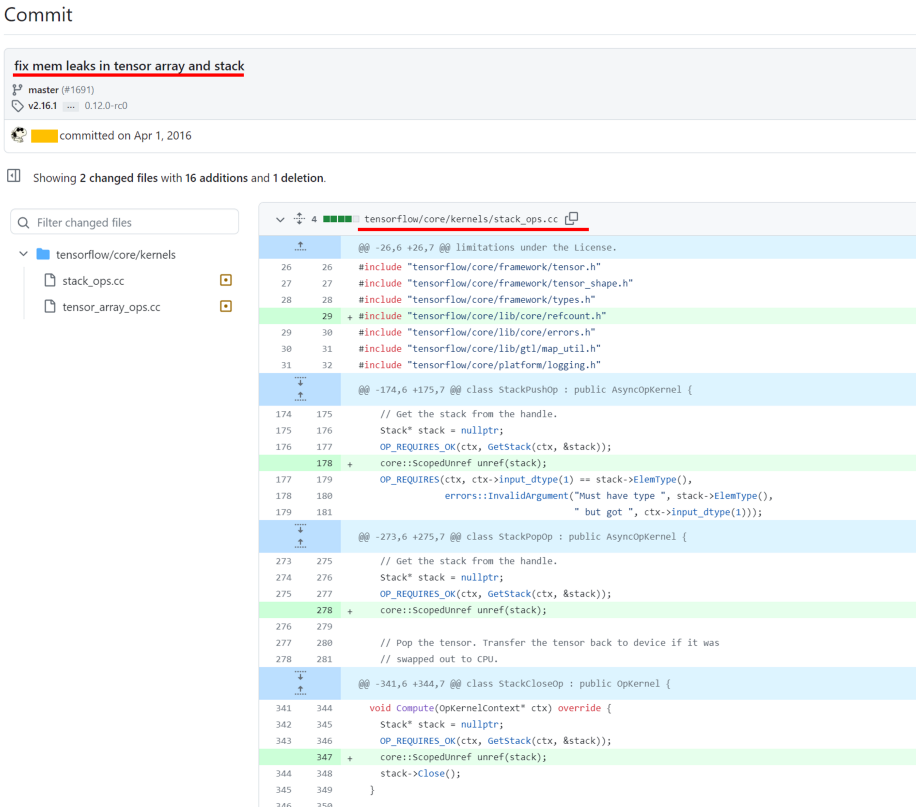


((b)) Issue 2

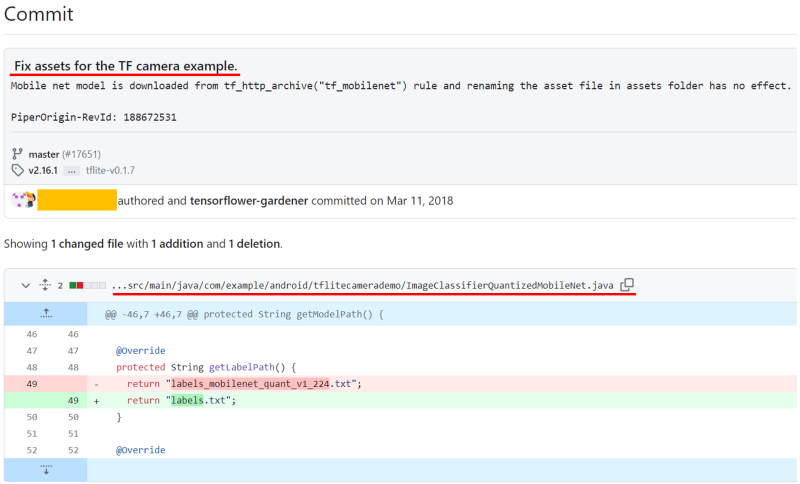


((c)) Pull Request 1

Figure A.4: A real-world example of newcomers encouraged by Issue Discussants to make contributions and become long-term contributors.



((a)) High commit centrality.



((b)) Low commit centrality.

Figure A.5: An example of commits with different levels of commit importance (commit centrality).

A.4 RQ4 Supplementary Result

Table A.2: Overall trend for Workload Composition, Work preference, and technical importance.

Profile	Core-Afterhour		Core-Workhour		Peripheral-Afterhour		Peripheral-Workhour	
workload_sequence	↗**(2.5%)**	↘(0.6%)	↗**(2.5%)**	↘(0.6%)	↗(0.4%)	↘(0.1%)	↗(0.1%)	↘(0.2%)
binned_entropy	↗(1.8%)	↘**(4.0%)**	↗(2.2%)	↘**(5.9%)**	↗(0.0%)	↘(0.3%)	↗(0.2%)	↘(0.6%)
c3(1)	↗(2.0%)	↘(2.6%)	↗(1.7%)	↘**(4.6%)**	↗(0.0%)	↘(0.1%)	↗(0.0%)	↘(0.2%)
c3(2)	↗(1.5%)	↘(2.1%)	↗(1.3%)	↘**(4.2%)**	↗(0.1%)	↘(0.0%)	↗(0.0%)	↘(0.1%)
c3(3)	↗(2.0%)	↘(2.3%)	↗(1.7%)	↘**(4.3%)**	↗(0.0%)	↘(0.1%)	↗(0.0%)	↘(0.1%)
number_cwt_peaks	↗(1.5%)	↘(2.8%)	↗(1.7%)	↘**(5.4%)**	↗(0.0%)	↘(0.3%)	↗(0.0%)	↘(0.4%)
longest_strike_above_mean	↗(1.1%)	↘(3.4%)	↗(1.1%)	↘**(4.7%)**	↗(0.0%)	↘(0.2%)	↗(0.0%)	↘(0.2%)
longest_strike_below_mean	↗(2.5%)	↘(1.7%)	↗**(4.2%)**	↘(1.5%)	↗(0.2%)	↘(0.2%)	↗(0.2%)	↘(0.1%)
diverse	↗(2.5%)	↘(2.3%)	↗(2.6%)	↘**(4.0%)**	↗(0.1%)	↘(0.3%)	↗(0.1%)	↘(0.2%)
balance	↗**(4.1%)**	↘(1.4%)	↗**(6.8%)**	↘(1.6%)	↗(0.3%)	↘(0.2%)	↗(0.3%)	↘(0.1%)
commit	↗(1.5%)	↘**(4.9%)**	↗(2.3%)	↘**(7.2%)**	↗(0.1%)	↘(0.1%)	↗(0.0%)	↘(0.1%)
issue	↗(1.1%)	↘(2.6%)	↗(1.3%)	↘(2.2%)	↗(0.0%)	↘(0.3%)	↗(0.0%)	↘(0.2%)
issue comment	↗(1.8%)	↘(3.2%)	↗(2.0%)	↘**(5.2%)**	↗(0.1%)	↘(0.2%)	↗(0.0%)	↘(0.5%)
pr comment	↗(2.6%)	↘(2.5%)	↗(2.6%)	↘**(4.6%)**	↗(0.1%)	↘(0.1%)	↗(0.0%)	↘(0.3%)
review	↗**(2.8%)**	↘(0.8%)	↗**(3.7%)**	↘(2.3%)	↗(0.1%)	↘(0.0%)	↗(0.1%)	↘(0.1%)
per commit centrality	↗**(7.1%)**	↘**(9.9%)**	↗**(7.9%)**	↘**(12.4%)**	↗**(0.5%)**	↘**(0.9%)**	↗**(0.7%)**	↘**(0.7%)**
period commit centrality	↗(1.4%)	↘**(4.9%)**	↗(1.6%)	↘**(7.3%)**	↗(0.0%)	↘(0.1%)	↗(0.0%)	↘(0.1%)

(%)↗ : The percentage of contributors reject the null hypothesis $H0_{13}$ and have an increasing trend in the given feature.

(%)↘ : The percentage of contributors reject the null hypothesis $H0_{14}$ and have a decreasing trend in the given feature.

Table A.3: Early-middle and middle-late stage trend for workload composition, work preference, and technical importance.

Profile	Core-Afterhour		Core-Workhour		Peripheral-Afterhour		Peripheral-Workhour	
	Early-Middle	Middle-Late	Early-Middle	Middle-Late	Early-Middle	Middle-Late	Early-Middle	Middle-Late
workload_sequence	↗(0.9%)	↘(0.0%) ↗(1.1%) ↘(0.6%)	↗(1.0%)	↘(0.3%) ↗(0.5%) ↘(0.8%)	↗(0.0%)	↘(0.0%) ↗(0.2%) ↘(0.2%)	↗(0.0%)	↘(0.0%) ↗(0.1%) ↘(0.0%)
binned_entropy	↗(1.7%)	↘(1.1%) ↗(0.9%) ↘**(4.0%)**	↗**(2.2%)**	↘(1.3%) ↗(0.8%) ↘**(7.3%)**	↗(0.1%)	↘(0.1%) ↗(0.1%) ↘(0.7%)	↗(0.1%)	↘(0.0%) ↗(0.0%) ↘(0.1%)
c3(1)	↗(1.1%)	↘(1.1%) ↗(1.1%) ↘**(2.1%)**	↗**(2.0%)**	↘(0.8%) ↗(0.8%) ↘**(5.3%)**	↗(0.1%)	↘(0.0%) ↗(0.0%) ↘(0.1%)	↗(0.0%)	↘(0.1%) ↗(0.0%) ↘(0.1%)
c3(2)	↗(1.1%)	↘(0.8%) ↗(0.9%) ↘(1.5%)	↗(1.7%)	↘(1.0%) ↗(0.8%) ↘**(4.6%)**	↗(0.0%)	↘(0.0%) ↗(0.0%) ↘(0.1%)	↗(0.0%)	↘(0.1%) ↗(0.0%) ↘(0.1%)
c3(3)	↗(1.5%)	↘(0.6%) ↗(1.3%) ↘**(2.3%)**	↗(1.9%)	↘(1.0%) ↗(0.9%) ↘**(4.7%)**	↗(0.0%)	↘(0.0%) ↗(0.0%) ↘(0.0%)	↗(0.0%)	↘(0.1%) ↗(0.0%) ↘(0.1%)
number_cwt_peaks	↗(0.8%)	↘(0.4%) ↗(0.9%) ↘**(3.2%)**	↗(1.9%)	↘(0.9%) ↗(0.3%) ↘**(6.5%)**	↗(0.2%)	↘(0.2%) ↗(0.1%) ↘(0.4%)	↗(0.0%)	↘(0.1%) ↗(0.0%) ↘(0.1%)
longest_strike_above_mean	↗(1.1%)	↘(0.9%) ↗(0.6%) ↘(1.7%)	↗(1.6%)	↘(0.7%) ↗(0.3%) ↘**(4.4%)**	↗(0.0%)	↘(0.0%) ↗(0.0%) ↘(0.1%)	↗(0.0%)	↘(0.0%) ↗(0.0%) ↘(0.1%)
longest_strike_below_mean	↗(0.6%)	↘(1.1%) ↗(2.6%) ↘(0.9%)	↗(0.6%)	↘**(2.6%)** ↗**(6.3%)** ↘(0.3%)	↗(0.0%)	↘(0.4%) ↗(0.3%) ↘(0.0%)	↗(0.1%)	↘(0.0%) ↗(0.5%) ↘(0.0%)
diverse	↗(0.8%)	↘(0.8%) ↗(0.6%) ↘(1.7%)	↗**(2.2%)**	↘(0.6%) ↗(0.6%) ↘**(4.2%)**	↗(0.1%)	↘(0.0%) ↗(0.2%) ↘(0.3%)	↗(0.1%)	↘(0.1%) ↗(0.0%) ↘(0.1%)
balance	↗(0.8%)	↘(1.3%) ↗**(3.2%)** ↘(0.4%)	↗(1.4%)	↘**(2.2%)** ↗**(6.8%)** ↘(0.6%)	↗(0.2%)	↘(0.1%) ↗(0.7%) ↘(0.1%)	↗(0.2%)	↘(0.2%) ↗(0.4%) ↘(0.1%)
commit	↗(0.6%)	↘(1.5%) ↗(0.6%) ↘**(2.5%)**	↗(1.7%)	↘**(2.4%)** ↗(1.0%) ↘(6.1%)	↗(0.0%)	↘(0.0%) ↗(0.0%) ↘(0.1%)	↗(0.0%)	↘(0.0%) ↗(0.0%) ↘(0.0%)
issue	↗(0.6%)	↘(0.2%) ↗(0.8%) ↘(1.1%)	↗(1.0%)	↘(0.6%) ↗(0.4%) ↘(1.4%)	↗(0.0%)	↘(0.1%) ↗(0.0%) ↘(0.3%)	↗(0.0%)	↘(0.1%) ↗(0.0%) ↘(0.0%)
issue comment	↗(1.5%)	↘(1.7%) ↗(1.1%) ↘**(3.4%)**	↗**(2.3%)**	↘(1.4%) ↗(0.8%) ↘**(5.9%)**	↗(0.2%)	↘(0.0%) ↗(0.1%) ↘(0.5%)	↗(0.1%)	↘(0.1%) ↗(0.0%) ↘(0.1%)
pr comment	↗(1.7%)	↘(0.6%) ↗(1.1%) ↘**(2.1%)**	↗**(2.4%)**	↘(1.4%) ↗(0.6%) ↘**(5.8%)**	↗(0.3%)	↘(0.1%) ↗(0.1%) ↘(0.3%)	↗(0.0%)	↘(0.1%) ↗(0.0%) ↘(0.1%)
review	↗**(2.8%)**	↘(0.2%) ↗(1.3%) ↘**(2.3%)**	↗**(3.8%)**	↘(0.4%) ↗(1.3%) ↘**(4.2%)**	↗(0.2%)	↘(0.0%) ↗(0.0%) ↘(0.1%)	↗(0.1%)	↘(0.1%) ↗(0.0%) ↘(0.1%)
commit centrality	↗**(4.9%)**	↘**(9.0%)** ↗**(5.8%)** ↘**(6.8%)**	↗**(6.5%)**	↘**(11.2%)** ↗**(7.4%)** ↘**(9.6%)**	↗(0.6%)	↘(0.7%) ↗(0.7%) ↘(0.6%)	↗(0.5%)	↘(0.1%) ↗(0.4%) ↘(0.2%)
period centrality	↗(0.4%)	↘(0.9%) ↗(0.8%) ↘**(2.1%)**	↗(1.3%)	↘**(2.5%)** ↗(0.6%) ↘**(4.4%)**	↗(0.0%)	↘(0.0%) ↗(0.0%) ↘(0.1%)	↗(0.0%)	↘(0.0%) ↗(0.0%) ↘(0.0%)

(%)↗: The percentage of contributors reject the null hypothesis $H0_{13}$ and have an increasing trend in the given feature.
(%)↘: The percentage of contributors reject the null hypothesis $H0_{14}$ and have a decreasing trend in the given feature.