Understanding the OSS Communities of Deep Learning Frameworks: A Comparative Case Study of PyTorch and TensorFlow

YUNQI CHEN, ZHIYUAN WAN*†, and YIFEI ZHUANG, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China
NING LIU, City University of Hong Kong, Hong Kong
DAVID LO, Singapore Management University, Singapore
XIAOHU YANG, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China

Over the past two decades, deep learning has received tremendous success in developing software systems across various domains. Deep learning frameworks have been proposed to facilitate the development of such software systems, among which, PyTorch and TensorFlow stand out as notable examples. Considerable attention focuses on exploring software engineering practices and addressing diverse technical aspects in developing and deploying deep learning frameworks and software systems. Despite these efforts, little is known about the open-source software communities involved in the development of deep learning frameworks.

In this paper, we perform a comparative investigation into the open-source software communities of the two representative deep learning frameworks, PyTorch and TensorFlow. To facilitate the investigation, we compile a dataset of 2,792 and 3,288 code commit authors, along with 9,826 and 19,750 participants engaged in issue events on GitHub, from the two communities, respectively. With the dataset, we first characterize the structures of the two communities by employing four operationalizations to classify contributors into various roles and inspect the contributions made by common contributors across the two communities. We then conduct a longitudinal analysis to characterize the evolution of the two communities across various releases, in terms of the numbers of contributors with various roles and role transitions among contributors. Finally, we explore the causal effects between community characteristics and the popularity of the two frameworks.

We find that the TensorFlow community harbors a larger base of contributors, encompassing a higher proportion of core developers and a more extensive cohort of active users compared to the PyTorch community. In terms of the technical background of the developers, 64.4% and 56.1% developers in the PyTorch and TensorFlow communities are employed by the leading companies of the corresponding open-source software projects, Meta and Google, respectively. 25.9% and 21.9% core developers in the PyTorch and TensorFlow communities possess Ph.D. degrees, while 77.2% and 77.7% contribute to other machine learning or deep learning open-source projects, respectively. Developers contributing to both communities demonstrate spatial and temporal similarities to some extent in their pull requests across the respective projects. The evolution of contributors with various roles exhibits a consistent upward trend over time in the PyTorch community. Conversely, a noticeable turning point in the growth of contributors characterizes the evolution of the TensorFlow community. Both communities show a statistically significant decreasing trend in the inflow rates of core developers. Furthermore, we observe statistically significant causal effects between the expansion of communities and retention of core developers and the popularity of deep learning frameworks. Based on our findings, we discuss implications, provide recommendations for sustaining open-source software communities of deep learning frameworks, and outline directions for future research.

CCS Concepts: • **Software and its engineering → Open source model**; • **Human-centered computing → Open source software**; **Empirical studies in collaborative and social computing**.

*Zhiyuan Wan is the corresponding author.
†Also with Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security.

Authors' addresses: Yunqi Chen, yunqichen@zju.edu.cn; Zhiyuan Wan, wanzhiyuan@zju.edu.cn; Yifei Zhuang, zhuangyf@zju.edu.cn, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China; Ning Liu, City University of Hong Kong, Hong Kong, ninliu@cityu.edu.hk; David Lo, Singapore Management University, Singapore, davidlo@smu.edu.sg; Xiaohu Yang, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China, yangxh@zju.edu.cn.

Additional Key Words and Phrases: Deep learning, community evolution, GitHub, developer classification

ACM Reference Format:
Yunqi Chen, Zhiyuan Wan, Yifei Zhuang, Ning Liu, David Lo, and Xiaohu Yang. 2024. Understanding the OSS Communities of Deep Learning Frameworks: A Comparative Case Study of PyTorch and TensorFlow. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (November 2024), 31 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep learning (DL), as a branch of machine learning (ML), has progressed dramatically over the past two decades, from a laboratory curiosity to a practical technology in widespread commercial use [55]. DL is receiving massive attention in developing software systems in many applications, including autonomous vehicles [20], image recognition [54], natural language processing [28], speech recognition [61] and disease diagnosis [35]. Various DL frameworks have been proposed to facilitate the development of such software systems, including PyTorch [72], TensorFlow [3], Keras [23], Google JAX [16], and Deeplearning4j [38]. Among the frameworks, PyTorch and TensorFlow are the two representative ones that are developed and maintained by open-source software (OSS) communities [37].

Considerable attention in practice focuses on the construction of robust pipelines for training and deploying DL models in a scalable fashion [29, 58, 60]. In the meantime, intensive investigation has been conducted into software engineering practices involved in the development of DL/ML systems [5, 17, 30, 67, 73, 86, 94, 96], as well as diverse technical aspects in DL/ML frameworks and software systems, such as bugs [89, 101], faults [46], and program failures [99]. Nonetheless, the OSS communities engaged in the development of DL frameworks have received less attention. Meanwhile, considerable prior research has explored various aspects of OSS communities, including proposing diverse operationalizations for classifying contributors [11, 26, 51, 52, 62, 66, 98], as well as investigating various community factors that influence sustainability [34, 76, 104, 106], and popularity of OSS projects [4, 9, 13, 31, 32]. However, little is known about the OSS communities of DL frameworks, and how these communities have evolved during the development of DL frameworks. With a better understanding of OSS communities behind DL frameworks, we could gain insights into sustaining the governance of OSS communities and the popularity of DL frameworks.

To address this gap, we followed a mixed-methods approach to perform a comparative study on the OSS communities of PyTorch and TensorFlow. Specifically, we investigated the structures of the two communities with respect to various contributor roles, traced the evolution of community structures and role transitions across project releases, and explored how community characteristics influence project popularity. We compiled a dataset of 2,792 and 3,288 code commit authors, as well as 9,826 and 19,750 participants involved in issue events on GitHub, from the two communities, respectively, spanning over 5 years in history. With our dataset, we investigated the following research questions:

RQ1. What are the structures of the two communities with respect to various contributor roles?

Hierarchical structures manifest among the contributors in OSS communities, usually referred to as the onion model, in which newcomers start as observers and move up the hierarchy to become active users, peripheral developers, and ultimately core developers [26, 84, 106]. Understanding community structures of OSS communities provides insights into coordination mechanisms employed by successful OSS projects [52], and functions of core development teams in OSS communities [106].

In RQ1, we investigated the structures of the two communities by classifying their contributors into core, peripheral developers, and active users with multiple operationalization, analyzed the technical backgrounds of contributors by examining their affiliations, education background and

work experience, and inspected the contributions of common contributors across the two communities. In comparison to PyTorch, the TensorFlow community demonstrates a larger contingent of contributors, comprising higher ratios of core developers and active users. Core developers exhibit consistent involvement in both projects, unlike peripheral developers. Additionally, 64.4% of developers in PyTorch and 56.1% in TensorFlow are employed by the leading companies behind these projects, Meta and Google. Among core developers, 25.9% and 21.9% hold Ph.D. degrees, while 77.2% and 77.7% contribute to other ML/DL open-source projects, respectively. Furthermore, developers engaged in both communities demonstrate spatial and temporal similarities to some extent in their pull requests across the respective projects.

RQ2. How do the two communities evolve across releases?

The evolution of OSS communities usually occurs spontaneously, characterized by the transition of contributors across various roles over time [68]. Understanding how OSS communities evolve is crucial in sustaining the continuous involvement of developers [107] and mitigating developer turnover [33]. Previous studies reveal that the retention, dropout, and inflow of core developers significantly influence the productivity and code quality of OSS projects [33, 48, 65].

In RQ2, we conducted a longitudinal analysis to investigate the evolution of the two communities across releases, in terms of numbers of contributors with different roles and role transitions among contributors. The evolution of the PyTorch community exhibits an upward trend in its contributors across various roles, accompanied by fluctuations in the growth of active users and peripheral developers. In contrast, the evolution of the TensorFlow community experiences an initial surge followed by a decline in its growth of contributors. Major releases of the two projects coincide with increased peripheral developers and active users in both communities. The PyTorch community tends to attract more peripheral developers relative to core developers, while the TensorFlow community tends to attract more active users relative to developers across releases. Over time, both communities exhibit a statistically significant decrease in the inflow rates of core developers, indicating potential challenges in engaging new contributors as projects progress. The higher dropout and inflow rates in the Tensorflow community suggest a less stable core development team for the TensorFlow project as compared to PyTorch. Major releases tend to hamper the inflow of practitioners into the core development team in the TensorFlow community.

RQ3. How do community characteristics affect the popularity of DL frameworks?

Previous studies investigated various factors that affect the popularity of OSS projects, including programming languages [9, 13], application domains [13], social media [32], and documentation updates [4]. Nonetheless, the causal effects between community characteristics and project popularity remain unexplored.

In RQ3, we applied a causal discovery method to explore the causal effects between community characteristics and the popularity of the two frameworks. PyTorch has accumulated a smaller total number of stars, which grows linearly at an average rate 2.2 times slower than TensorFlow. Two major releases of TensorFlow tend to exert divergent impacts on the popularity of the project. Project popularity has statistically significant and positive causal effects on the influx of new developers into core development teams and attracting active users in the PyTorch community, and on retaining core developers in the TensorFlow community. Conversely, the growth of peripheral developers and retention of core developers demonstrate statistically significant and positive causal effects on the popularity of PyTorch, while the expansion of active users exhibits statistically significant and negative causal effects on the popularity of TensorFlow.

Based on our findings, we discuss implications and provide practical lessons for sustaining OSS communities of DL frameworks and facilitating decision-making concerning the selection of DL frameworks. We also highlight several research avenues, such as the integration of diverse data

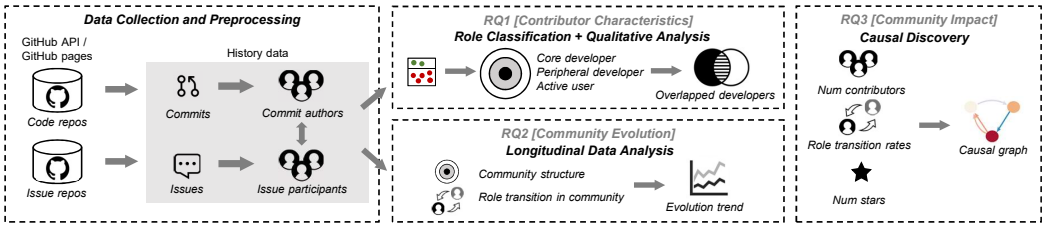


Figure 1. Overview of research methodology.

sources for developer classification in OSS communities and a systematic exploration into the impact of major releases on OSS communities. This paper makes the following contributions:

- We perform a comparative study to characterize the OSS communities behind two representative deep learning frameworks, TensorFlow and PyTorch.
- We provide a dataset that includes 12,618 and 23,038 contributors, as well as 81,892 and 55,286 issues from the two communities for future investigations by others[1].
- We evaluate the effectiveness of four operationalizations for core-peripheral developer classification in OSS communities, using the development data of deep learning frameworks.
- We provide a discussion of practical implications and outline future avenues of research.

The replication package is online at https://github.com/UniqueClouds/deep-learning-communities.

2 METHODOLOGY

We designed and carried out a mixed-methods empirical study, analyzing a dataset of the contributors in the two OSS communities of deep learning frameworks on GitHub, as depicted in Figure 1. In RQ1, we investigated the structural composition of the two communities by classifying contributors into various roles, and further analyzed the developers who contribute to both communities, which is crucial for understanding the characteristics of the contributors in these communities. In RQ2, we explored the temporal evolution of the two communities by analyzing the number of contributors and their role transitions over time, which provides insights into the growth and turnover dynamics of these communities. In RQ3, we applied a causal discovery method to evaluate the impact of community characteristics on the popularity of the two deep learning frameworks, which is essential for understanding the influence of communities on the success and adoption of these frameworks. Our research methodology is detailed in the following subsections.

2.1 Data Collection and Preprocessing

There are several well-known DL frameworks, such as PyTorch, TensorFlow, Keras, Google JAX, and Deeplearning4j. PyTorch and TensorFlow are the two representative frameworks for building and deploying DL models. Keras is a high-level API that runs on top of TensorFlow, offering an easy-to-use interface for building and training DL models. Google JAX is geared more towards research and experimentation for developing new algorithms and exploring different DL approaches. Deeplearning4j is a commercial-grade DL library suitable for developing DL applications on the JVM. Previous research in software engineering has focused on PyTorch and TensorFlow, conducting comparative studies between the two frameworks [22, 27, 43], as well as investigating various aspects of the two frameworks such as bugs [50, 57, 101], performance

[1]https://doi.org/10.5281/zenodo.10477123

Table 1. Descriptive statistics of dataset.

	PyTorch	TensorFlow
Start Date	August 13th, 2016	November 7th, 2015
End Release	1.12.0	2.9.0
# Commits	47,790	88,682
# Issues	81,892	55,286
# Issue Events	1,367,246	935,513
# Commit Authors	2,792	3,288
# Participants Involved in Issue Events	9,826	19,750

issues [19], API evolution [105], framework bindings [56], and supply chains and code clones [36, 64, 85]. Consequently, we chose PyTorch and TensorFlow as the subjects for our comparable case study on the OSS communities of deep learning frameworks.

We collected historical contributor data of the PyTorch and TensorFlow communities on GitHub until August 1st, 2022. For each community, we first collected the commits of the `master` branch of its code repository through GitHub REST API. Note that we excluded merge commits as code changes in merge commits are already reflected in the parent branches that we considered. Next, we collected the account information (i.e., GitHub login) of GitHub users who have authored commits as indicated by the commits, to enable further analysis of the development activities of commit authors on GitHub. We further collected issues (including pull requests) and the relevant issue events from the issue repository, and extracted the account information of the participants who opened issues or were involved in issue events. In addition, we excluded bot accounts by using `BoDeGHa` [40] for automatic detection and a follow-up validation of the detection results.

We notice that 7,355 out of 47,790 commits (15.4%) and 10,307 out of 88,682 (11.6%) do not include GitHub login in their authors for the PyTorch and TensorFlow projects, respectively. Thus, we followed three steps to identify GitHub login of the authors of these commits: (1) We first identified the aliases of commit authors by aggregating similar (*name*, *email*) tuples as inspired by previous studies [39, 53, 93], considering that the same person may use different aliases when committing code. (2) For each commit author we identified, we manually checked the commits they authored on GitHub pages, to identify their GitHub accounts. (3) For the rest of the commit authors whom we cannot identify their GitHub accounts, we searched their names and emails online to determine the possible GitHub accounts, and further validated the accounts by inspecting account activities. In this way, we identified the GitHub accounts of 234 and 188 additional commit authors for the PyTorch and TensorFlow projects, respectively.

As a result, we compiled a dataset with 2,792 and 3,288 commit authors, among whom 2,472 and 3,183 have GitHub accounts, as well as 9,826 and 19,750 GitHub users who have been involved in GitHub issue events, for PyTorch and TensorFlow, respectively, as shown in Table 1.

2.2 Role Classification

Based on our dataset, we classified the contributors of each community into three roles, i.e., core developers, peripheral developers, and active users. We considered a contributor who has previously made contributions to the `master` branch of the code repository as a *developer* (core or peripheral), and one who has been involved in issue events (e.g., opening issues, merging pull requests, and adding comments) but never made code contributions to the `master` branch as an *active user*. We further applied an automatic approach that implements four operationalizations to classify the developers into core and peripheral developers with respect to their contributions – developers

who have accomplished 80% of the contributions in a project are core developers, and the rest are peripheral developers – which aligns with the standard 80th percentile threshold in the core-peripheral classification [26]. The four operationalizations include three variations of count-based operationalizations and an operationalization from the network perspective, as proposed in previous work [51], to characterize the contributions of a contributor:

- **Commit count**: the number of commits that a contributor has authored and eventually has been merged to a repository in a certain time period. Core developers typically contribute to a code repository more frequently; their commit counts tend to be higher than those of peripheral developers.
- **Lines of code (LOC) count**: the number of added and deleted lines of code a contributor has authored and eventually has been merged to a repository in a certain time period. Since core developers are responsible for the majority of code changes, they are supposed to achieve a higher LOC count than peripheral developers.
- **Issue count**: the number of issues a contributor in a project on GitHub in a certain time period. Core developers usually have a higher level of technical skill as well as a better understanding of the open-source project than peripheral developers. Thus, they tend to be more intensively involved in discussions of issues and pull requests in a project than peripheral developers.
- **Degree centrality of issue network**: An issue network is a relational abstraction that represents the communication of contributors in the issue repository of a project on GitHub. In the issue network for a given project, the nodes represent contributors and the edges represent the communication relationship between contributors. To construct an issue network, we applied a variation of the standard approach for mailing lists [8], where edges are added between individuals who contribute to the discussion of a common issue as well as the issuer, within the same time period. Degree centrality aims at measuring local importance, representing the number of connections (edges) a contributor has to other contributors [18]. As important members of the leadership and coordination structure, core developers have connections with other core members and with peripheral developers to provide technical guidance, thus having a greater degree of centrality.

2.3 Characterizing Technical Background of Developers

To characterize the technical background of developers, we first analyzed the affiliations of the developers who contribute to the two communities. Specifically, we determined the affiliations of the developers based on the domains of their email addresses, following the approach of previous studies [102, 104] that involves the steps below:

- **Email Domain Extraction and Filtering.** We initially extracted 678 domains from the email addresses of 5,043 developers in the two communities. We then excluded domains such as *gmail.com* and *users.noreply.github.com* by referring to a list of free email providers [69]. As a result, we identified 608 domains representing non-free email providers.
- **Email Domain and Affiliation Matching.** We used two open-source lists [6, 45] that include email domains for organizations, companies, and universities. Based on the lists, we matched 283 organizations, companies, and universities with the identified 608 email domains.

Furthermore, we conducted a manual examination of the educational and work experience of a random sample of 154 and 261 core developers in the PyTorch and TensorFlow communities, with a 95% confidence level and a 5% margin of error. For each developer in the samples, we searched their GitHub and LinkedIn profiles, as well as their personal blogs. Wherever accessible, we

inspected these sources to gather information on the educational background (e.g., degree and major), work experience (e.g., employers and job responsibilities), and open-source contributions (e.g., activities on GitHub) of the sampled developers.

2.4 Longitudinal Analysis of Community Characteristics

To understand the evolution of characteristics of OSS communities, we initially considered major and minor releases of each project to segment our continuous dataset into distinct time windows like Jergensen et al. did [49]. PyTorch and TensorFlow both follow a semantic versioning convention to number the releases. In such repositories, versions are identified by three integers, in the format `x.y.z`: increments in `x` denote major releases, which can be incompatible with old versions; increments in `y` denote minor releases, which add functionality in a backward-compatible manner; and increments in `z` denote patches implementing bug fixes. The temporal segmentation approach resulted in some of the initial time windows spanning fewer than three months. Previous studies [51] indicated that role transitions may not occur within a short period of time. Thus, we further applied a three-month restriction on the duration of the time windows, as did Joblin et al. [51]. Specifically, if an initial time window spans less than three months, we merged it with its following time window. For instance, TensorFlow 0.11.0 was released on November 12th, 2016, while TensorFlow 0.12.0 was released on December 20th, 2016; the time interval between these two releases is less than three months, thus we merged the time windows of release 0.11.0 and 0.12.0 for TensorFlow. In this way, we obtained 14 and 20 time windows from the 14 and 20 releases of PyTorch and TensorFlow, respectively.

Next, we estimated the numbers of core and peripheral developers as well as active users of each community on a time window basis, and evaluated the evolution of the community structures with respect to various roles across releases with two measures:

$$Ratio_{pd/cd} = \frac{\#\ peripheral\ developers}{\#\ core\ developers} \tag{1}$$

$$Ratio_{au/d} = \frac{\#\ active\ users}{\#\ developers} \tag{2}$$

We also captured role transitions with respect to core developers in each community on a time window basis: (1) **Retention**: A core developer retained their core role in the current time window. (2) **Dropout**: A core developer has made no commits since the current time window. (3) **Core to peripheral** (*core2peripheral*): In cases where a core developer had previously contributed but no longer held the core role within the current time window, we characterized this as a role transition from core to peripheral. (4) **Inflow**: If a developer assumed a core role within the current time window without having held the core role before, we denoted this as an *inflow* of a core developer. Specifically, we evaluated the evolution of role transition rates with respect to core developers in each community across releases.

In addition, we used the Mann-Kendall test [42] in the `pyMannKendall` [47] Python package to determine whether the trends observed in the longitudinal analysis are increasing or decreasing over time to a statistically significant degree.

2.5 Causal Discovery between Community Characteristics and Project Popularity

We used a causal discovery method to test whether observational time series data support the hypothesis that the community characteristics causally affected the project popularity of PyTorch (September 2017 - July 2022) and TensorFlow (March 2016 - July 2022). Causal discovery [77], or causal structure learning, qualitatively reconstructs the links in either a complete causal graph or just the causes of a particular target variable. Specifically, we used the PCMCI method [78] in

the `Tigramite` Python package[2], which can reliably estimate causal graphs including time lags from time series data. The PCMCI method estimates a "time series graph" (causal graphs including time lags) with nodes representing variables at different time lags and links representing causal relations. If the time lag is denoted by τ, a causal link is notated $X_{t-\tau} \rightarrow Y_t$, which exists if $X_{t-\tau}$ is not conditionally independent of Y_t given the past of all variables. In the time series graph, the parents $\mathcal{P}(X)$ of a variable X are defined as the set of all nodes with links towards X.

PCMCI follows a two-stage procedure: (1) Condition selection via PC_1 algorithm: For each variable X, estimate a superset $\tilde{\mathcal{P}}(X_t)$ of its parents $\mathcal{P}(X_t)$ with PC_1 algorithm, which is an iterative Markov discovery algorithm [82]. The condition selection stage aims to avoid conditioning on irrelevant variables. (2) Momentary conditional independence (MCI) test: use these parents as conditions, and test all variable pairs $X_{t-\tau}$ and Y_t with time lags $\tau \in 0, ..., \tau_{max}$ to establish a link $X_{t-\tau} \rightarrow Y_t$. If $\tau = 0$, conditional independence is estimated for contemporaneous variables.

Both stages consist of conditional independence tests, which can be implemented with different test statistics. We used the linear partial correlation test `ParCorr` implemented in `Tigramite` for conditional independent tests given `ParCorr` have a higher detection power of causal relations than other test statistics [78]. We chose to let `Tigramite` determine the significance level α_{PC} for conditional independence tests in the condition selection stage based on the Akaike information criterion. In the MCI stage, we set the maximum time lag (τ_{max}) to be 12, which is 1 year. The resulting p values and correlation coefficients of conditional independence tests of MCI indicate the significance and strength of causal links in time series graphs.

To compile time series datasets in causal discovery, we considered a time window of one month for both communities. For each time window, we computed the variables that capture community characteristics, and collected the number of stars as a proxy of project popularity, which is widely used in previous studies (e.g., [12–14]). The resulting time series datasets contain 71 and 80 observations for PyTorch and TensorFlow, respectively.

3 RESULTS

We now present the results of applying our research methodology to our dataset. We succinctly answer each of our research questions.

3.1 Structures of Communities (RQ1)

In RQ1, we investigate the structures of the PyTorch and TensorFlow communities with respect to their contributors with different roles, as well as compare the contributions of the common developers across the two communities.

3.1.1 Roles of Contributors in Communities. Table 4 presents the contributor roles we have identified in the two DL communities, including core developers, peripheral developers, and active users. The *Commit Count*, *LOC Count*, *Issue Count* and *Issue Network* rows present the results of four operationalizations we considered for the classification of core-peripheral developers, respectively. The *Aggregate* row presents the results after we merged the resulting sets of core developers from the four operationalizations.

The PyTorch community has 2,792 developers, who have previously committed to the `master` branch, and 9,826 active users, who have been involved in issue events but never made code contributions. The number of active users is 3.5x greater than that of developers. Among the 2,792 developers in the PyTorch community, the percentage of core developers ranges from 3.8% to 7.3% based on the four operationalizations. Meanwhile, the TensorFlow community has 3,288 developers and 19,750 active users. The number of active users is up to 6.0x the number of developers.

[2]https://github.com/jakobrunge/tigramite

Table 2. Companies with top 5 most developers in deep learning communities.

(a) PyTorch

Company	# Developers	Percentage
Meta	595	64.4%
Intel	40	4.3%
Microsoft	31	3.4%
NVIDIA	21	2.3%
AMD	13	1.4%

(b) TensorFlow

Company	# Developers	Percentage
Google	608	56.1%
Intel	64	5.9%
NVIDIA	38	3.5%
IBM	34	3.1%
Arm	31	2.9%

Table 3. Technical background of core developers in deep learning communities.

	% Core Developers in PyTorch	% Core Developers in TensorFlow
Ph.D. Holder	25.9%	21.9%
Employee in the Leading Company	50.0%	23.4%
Full-Time Job as Contributor	44.9%	19.3%
Contributor of other ML/DL OSS project(s)	77.2%	77.7%

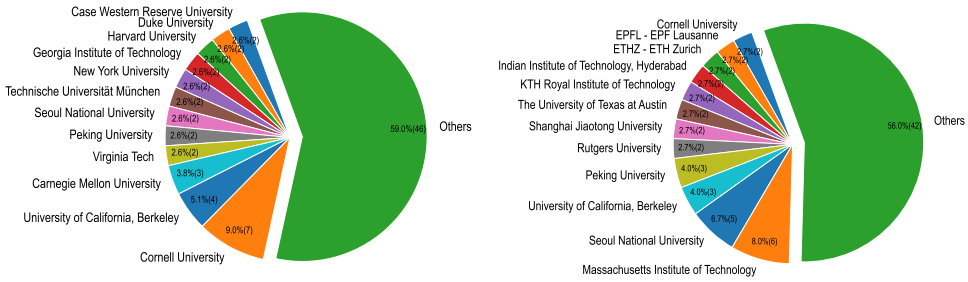
The percentage of core developers in the TensorFlow community ranges from 4.0% to 19.1% based on the four operationalizations.

We further compared the number of contributors with various roles between the two communities. The numbers of developers and active users in the TensorFlow community are 1.18x and 2.01x of those in the PyTorch community, indicating that the TensorFlow community involved a larger population of contributors compared to the PyTorch community, especially in terms of active users. The TensorFlow community has a larger ratio of active users relative to developers compared to the PyTorch community (6.0x vs. 3.5x). Core developers account for 9.1% of the developers in the PyTorch community, which is lower than the percentage of the TensorFlow community (24.5%).

Finding 1. The TensorFlow community exhibits a larger population of contributors, with a higher proportion of core developers in its development team and a more extensive cohort of active users relative to developers, as compared to the PyTorch community.

3.1.2 Technical Background of Developers in Communities. The contributors in the PyTorch and TensorFlow communities are affiliated with 144 and 190 organizations, companies, and universities, respectively, with 51 overlapped affiliations. Table 2(a) and (b) present the top 5 affiliations with the most developers in the PyTorch and TensorFlow communities, respectively. We observed that the employees in Meta and Google account for over half of the developers in the PyTorch and TensorFlow communities. Note that Meta and Google are the primary sponsors of the two deep learning frameworks. Additionally, the employees of leading hardware companies like Intel and NVIDIA show enthusiasm in contributing to the development of the two deep learning frameworks.

In addition, Figure 2(a) and (b) present the distributions of 78 and 75 developers who are affiliated with 58 and 54 universities in the PyTorch and TensorFlow communities, respectively. Among the affiliated universities, the universities with the top 3 most developers are Cornell (7), UC Berkeley (4), and CMU (3) for the PyTorch community, and MIT (5), Seoul National University (5), and UC Berkeley (3) for TensorFlow. Meanwhile, 59% and 56% affiliated universities have only one developer in the PyTorch and TensorFlow communities. The contributions from the developers affiliated with universities suggest the potential collaboration between academia and industry in the DL communities.



(a) PyTorch (b) TensorFlow

Figure 2. Developers affiliated with universities in deep learning communities.

Table 3 presents the technical background of the core developers in the two communities. We observed that over one-fifth of the core developers in the two communities hold Ph.D. degrees, majoring in computer science, mathematics, informatics, and physics. The employees in the leading company account for a larger amount of core developers in PyTorch as compared to TensorFlow (25.9% for META vs. 21.9% for Google). 44.9% and 19.3% of the core developers in PyTorch and TensorFlow contribute to the development of the two DL frameworks as a full-time job. In addition, 77.2% of the core developers in PyTorch have contributed to other ML/DL OSS projects, such as triton [2], Fuser [25], and pyro [7]. Meanwhile, 77.7% of the core developers in TensorFlow have contributed to other ML/DL OSS projects, such as xla [70], iree [87], and profiler [88].

Finding 2. In the PyTorch and TensorFlow communities, 64.4% and 56.1% of developers are employed by their leading companies, Meta and Google, respectively. 25.9% of the core developers in PyTorch and 21.9% in TensorFlow hold a Ph.D. degree. Furthermore, 77.2% of the core developers in PyTorch and 77.7% in TensorFlow have contributed to other ML/DL OSS projects.

3.1.3 Contributors in Common across Communities. We further investigated the software practitioners who contribute to both the PyTorch and TensorFlow communities. We observed that 233 developers and 1,164 active users made contributions to both communities. As shown in Table 5, among the 233 common developers across the two communities, 12 developers take core roles in both communities, while 97 developers (77 + 20) take core roles in either community. In total, 109 out of the 233 common developers take the core roles in the two communities, accounting for 46.8% of the common developers. The percentage is higher than that of either community, 9.1% for PyTorch and 24.5% for TensorFlow. The higher percentage of core developers among common developers indicates that core developers are more likely to contribute to multiple deep-learning projects, compared to peripheral developers.

Finding 3. Core developers demonstrate a higher propensity to contribute across the two projects, as opposed to peripheral developers.

Among the common developers across the two communities, we identified three groups of developers: (i) core developers in both communities (12), (ii) core developers in either community (97), and (iii) peripheral developers in both communities (124). We further investigated the *spatial* and *temporal* characteristics of the contributions across the two communities made by the three groups of developers.

Table 4. Contributors with various roles in deep learning communities identified by classification operationalizations.

(a) PyTorch

	# Core Developers	# Peripheral Developers
Commit Count	191 (6.8%)	2,601 (93.2%)
LOC Count	110 (3.9%)	2,682 (96.1%)
Issue Count	205 (7.3%)	2,587 (92.7%)
Issue Network	107 (3.8%)	2,685 (96.2%)
Aggregate	254 (9.1%)	2,538 (90.9%)

(b) TensorFlow

	# Core Developers	# Peripheral Developers
Commit Count	260 (7.9%)	3,028 (92.1%)
LOC Count	130 (4.0%)	3,158 (96.0%)
Issue Count	628 (19.1%)	2,660 (80.1%)
Issue Network	178 (5.4%)	3,110 (94.6%)
Aggregate	805 (24.5%)	2,483 (75.5%)

Table 5. Developers in common across PyTorch and TensorFlow communities.

		TensorFlow		
		Core Developers	Peripheral Developers	None
PyTorch	Core Developers	12	20	222
	Peripheral Developers	77	124	2,337
	None	716	2,339	/

In terms of spatial characteristics, we inspected the titles and discussions of pull requests to understand the contributions developers made. As for temporal characteristics, we compared the time periods during which developers submitted pull requests to a code repository.

Among the 12 core developers in both communities, eight developers (66.7%) tend to contribute more code in the PyTorch community as compared to TensorFlow. Four developers (33.3%) reveal similarities in terms of spatial characteristics of pull requests across the two communities. For instance, a core developer has fixed bugs related to JIT compilers in both projects (see Figure 9(a) and (b) in Appendix); a core developer has updated the `QNNPACK` submodule in PyTorch, and fixed a crash when using the `XNNPACK` library of TensorFlow, which is based on the `QNNPACK` library in PyTorch (see Figure 9(c) and (d) in Appendix). In addition, 4 of 12 of the developers have simultaneously submitted pull requests to both projects, indicating temporal similarity in pull requests of common core developers to some extent.

Some of the 97 core developers in either community exhibit temporal and spatial similarity to a certain extent in their contributions across the two communities. 32 out of 97 (33%) demonstrate similarity in the spatial characteristics of their pull requests across the two communities. Take a developer for example, who takes the core role in TensorFlow and the peripheral role in PyTorch, he/she has submitted pull requests relevant to `oneDNN` for `AArch64` in both projects (see Figure 10 in Appendix). On the other hand, 33 out of 97 (34%) have submitted pull requests to both projects during

overlapped periods, demonstrating similarity in the temporal characteristics of their contributions across the two communities.

Some of the 124 peripheral developers in both communities demonstrate temporal and spatial similarity in their contributions, particularly in terms of documentation. 16.9% (21 out of 124) of the peripheral developers have submitted pull requests to both projects during overlapping periods, indicating a certain degree of temporal similarity in their contributions. In terms of spatial similarity, 12.1% (15 out of 124) have submitted pull requests regarding documentation, including 'format', 'typo fix', and 'docstring'.

Finding 4. Developers contributing to both communities exhibit a certain level of spatial and temporal resemblance in their pull requests across the respective projects.

3.2 Evolution of Communities (RQ2)

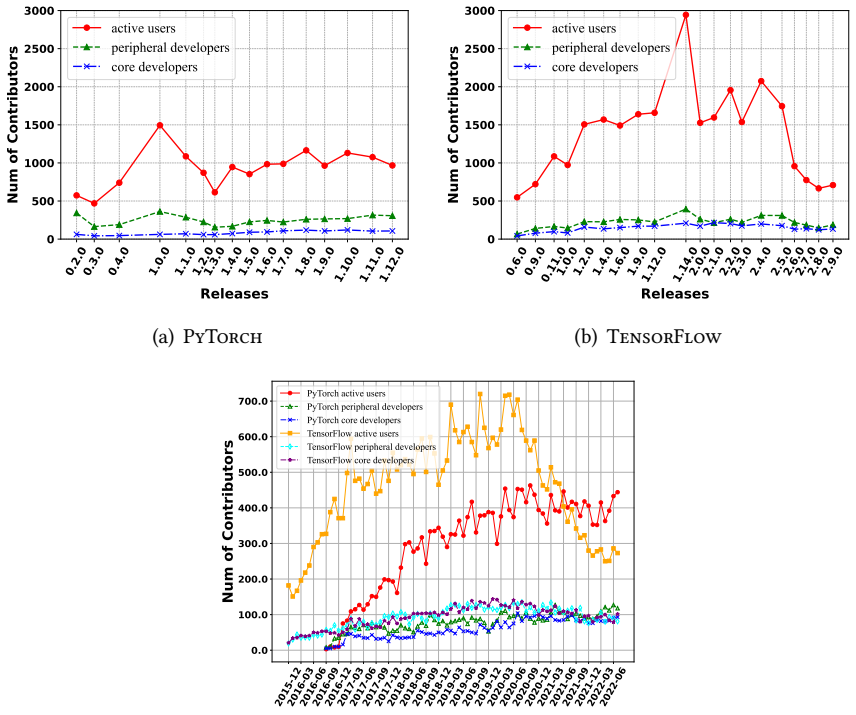
In RQ2, we investigate the evolution of two communities over time in terms of numbers of contributors with various roles, role ratios, and role transitions.

3.2.1 Evolution of Contributors with Various Roles. Figure 3 presents the numbers of core developers, peripheral developers, and active users of the PyTorch and TensorFlow communities across releases.

As depicted in Figure 3(a), the numbers of contributors associated with each of the three roles in the PyTorch community demonstrate an upward trend over time, accompanied by fluctuations in the growth of active users and peripheral developers. In particular, the number of active users has nearly doubled from release 0.4.0 to 1.0.0, followed by a substantial decrease until version 1.3.0. Subsequently, the number of active users has experienced a gradual increase, characterized by a relatively steady trend. Meanwhile, the growth of peripheral developers follows a similar pattern to active users, experiencing a peak at release 1.0.0 and declining numbers until version 1.3.0. Subsequent releases, particularly from 1.9.0 to 1.12.0, have experienced a discernible upward trend. In contrast, the number of core developers exhibits a consistent rise, initiating from release 0.2.0 and stabilizing after the release of version 1.7.0.

As shown in Figure 3(b), the evolution of the contributors with various roles in the TensorFlow community exhibits a distinct turning point, initially demonstrating an ascending trend followed by a subsequent decline. Specifically, the number of active users has experienced a steady increase from release 0.6.0 to 1.12.0, nearly doubling by release 1.14.0. Subsequently, after release 1.14.0, a notable decline in active users ensued until reaching its lowest point at release 2.8.0. The evolution of core and peripheral developers resembles that of active users, albeit with a more subdued level of fluctuation. The turning point in the evolution of contributors may be attributed to the substantial transition of TensorFlow from its 1.x.x releases to the major release of 2.0.0.

Comparatively, notable peaks in the numbers of peripheral developers and active users occurred in both communities around major releases, at the major release 1.0.0 of PyTorch and around the major release 2.0.0 of TensorFlow. Moreover, a stable trend in the number of core developers persisted in both communities, stabilizing around 100 after PyTorch release 1.7.0 and approximately 130 after TensorFlow release 2.6.0.



(a) PyTorch (b) TensorFlow

(c) PyTorch vs. TensorFlow (in months)

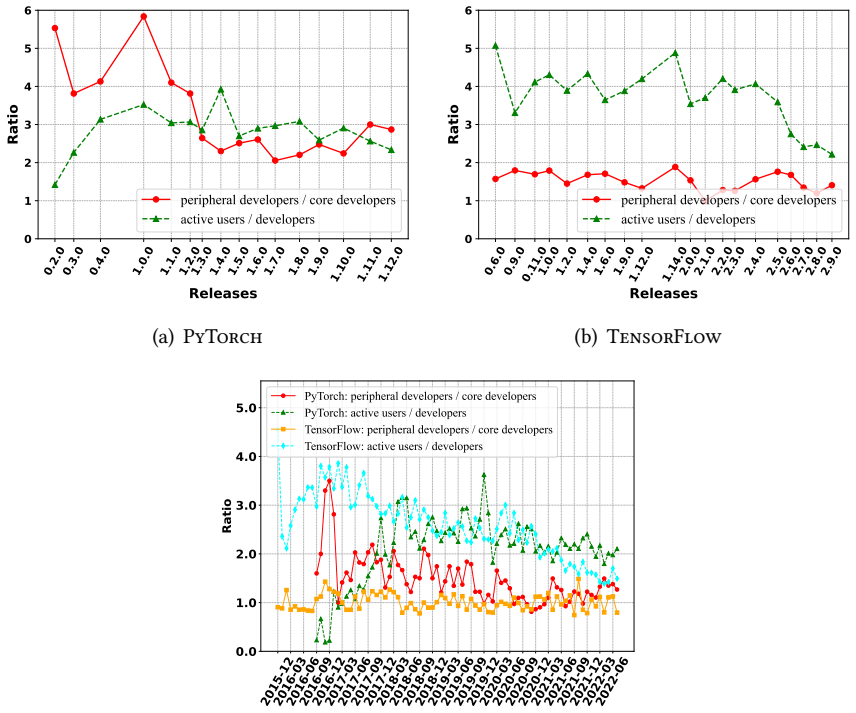
Figure 3. Numbers of contributors with various roles in deep learning communities across releases.

Finding 5. The evolution of contributors with various roles in the PyTorch community exhibits a rising trend over time, accompanied by fluctuations in the growth of active users and peripheral developers. The evolution of contributors with various roles in the TensorFlow community indicates a noticeable shift, starting with an upward trend followed by a subsequent decline. Substantial increases in peripheral developers and active users coincided with major releases in both communities, while a stable trend persisted in the numbers of core developers across both.

3.2.2 Role Ratios across Releases. Figure 4 presents the evolution of role ratios of the PyTorch and TensorFlow communities across releases. As shown in Figure 4 (a), $Ratio_{pd/cd}$ and $Ratio_{au/d}$ in the PyTorch community exhibit fluctuations prior to the major release 1.0.0, subsequently converging between 2.0 and 3.0 after release 1.5.0, implying a balanced contributor growth across roles in the PyTorch community. Specifically, $Ratio_{pd/cd}$ peaked at 5.8 during release 1.0.0, indicating a 5.8-fold difference between peripheral and core developers, followed by a sharp decline until release 1.4.0. In addition, the Mann-Kendall trend test indicates a statistically significant decreasing trend in $Ratio_{pd/cd}$ (p-values = 0.023). As shown in Figure 4(b), $Ratio_{pd/cd}$ and $Ratio_{au/d}$ in the TensorFlow community exhibit a significant declining trend, particularly after release 1.14.0, as supported by the Mann-Kendall trend test (*p-values* = 0.042 and 0.021). Specifically, both ratios reach their peaks

at release 1.14.0, with subsequent downward trends in $Ratio_{au/d}$, while $Ratio_{pd/cd}$ demonstrates fluctuations between 1 and 2, implying that the number of peripheral developers ranges from 1 to 2 times the number of core developers.

Comparatively, the PyTorch community demonstrates a higher ratio of peripheral to core developers ($Ratio_{pd/cd}$), while TensorFlow has a higher ratio of active users to developers ($Ratio_{au/d}$), which indicates a relatively larger proportion of peripheral developers and a smaller proportion of active users in the PyTorch community over time as compared to TensorFlow.



(a) PyTorch (b) TensorFlow

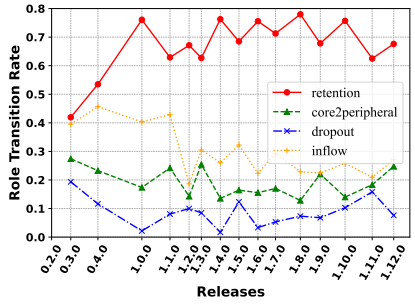
(c) PyTorch vs. TensorFlow (in months)

Figure 4. Role ratios in deep learning communities across releases.

Finding 6. The PyTorch community tends to attract relatively more peripheral developers than core developers as compared to TensorFlow. In the meantime, the TensorFlow community tends to attract relatively more active users than developers across releases as compared to PyTorch.

3.2.3 Role Transitions across Releases. Figure 5 presents the evolution of role transitions across releases for core developers in the PyTorch and TensorFlow communities.

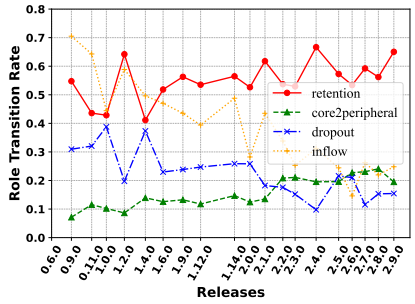
As shown in Figure 5(a), the role transition rates in the PyTorch community tend to fluctuate within a narrow range after the major release 1.0.0. In particular, the Mann-Kendall test indicates a statistically significant decreasing trend in the inflow rate of core developers (*p-values*=0.023). In the meantime, the retention rate of core developers ranges between 0.6 and 0.8 after the major



Rates	Min	Max	Mean	Std	Trend
Retention	0.419	0.780	0.672	0.097	-
Core2peripheral	0.128	0.274	0.191	0.049	-
Dropout	0.017	0.194	0.087	0.049	-
Inflow	0.186	0.457	0.298	0.086	↓ *

*p<0.05; **p<0.01; ***p<0.001

(a) PyTorch



Rates	Min	Max	Mean	Std	Trend
Retention	0.411	0.667	0.549	0.070	↑ *
Core2peripheral	0.071	0.241	0.158	0.052	↑ ***
Dropout	0.082	0.342	0.180	0.065	↓ **
Inflow	0.146	0.705	0.390	0.153	↓ ***

*p<0.05; **p<0.01; ***p<0.001

(b) TensorFlow

Figure 5. Role transitions of core developers in deep learning communities across releases.

release 1.0.0. The decreasing trend of inflow rates and the stabilization of retention rates suggest growing stability within the core development team of the PyTorch project over time.

As shown in Figure 5(b), various role transition rates in the TensorFlow community exhibit distinct upward or downward trends from the initial release, as evidenced by the Mann-Kendall test. Specifically, the retention and core2peripheral rates demonstrate statistically significant increasing trends (p-*values* = 0.036 and 2.7×10^{-5}), while the dropout and inflow rates show statistically decreasing trends (p-*values* = 0.001 and 3.9×10^{-6}). Such statistically significant trends suggest that the TensorFlow community tends to attract fewer newcomers and experience less turnover within its core development team over time. Additionally, sharp declines in the inflow rate are observed at major releases 1.0.0 and 2.0.0, indicating a negative impact from major releases on the influx of developers into the core development team.

In comparison, we observe statistically significant decreasing trends in inflow rates across both communities over time, indicating potentially increasing challenges in engaging new contributors in the core development team as projects evolve. In the meantime, the average core2peripheral rates across releases remain comparable between the two communities (0.191 for PyTorch versus 0.158 for TensorFlow). The TensorFlow community demonstrates higher dropout and inflow rates on average, approximately 2.1x and 1.3x respectively those of the PyTorch community, indicating a comparatively less stable core development team in the TensorFlow project. In particular, the TensorFlow community demonstrates substantially higher inflow rates during its initial releases

as compared to PyTorch, implying an earlier engagement of practitioners in the TensorFlow community.

Finding 7. Over time, a statistically significant decreasing trend is evident in the inflow rates of core developers across both communities, indicating potential challenges in engaging new contributors as projects progress. The higher dropout and inflow rates over time in the TensorFlow community suggest a less stable core development team for the TensorFlow project as compared to PyTorch. Major releases tend to negatively influence the inflow of developers into the core development team in the TensorFlow community.

3.3 Community Characteristics vs. Project Popularity (RQ3)

In RQ3, we explore how the popularity of projects evolves across releases, and investigate how community characteristics affect project popularity over time.

3.3.1 Evolution of Project Popularity. Figure 6 illustrates the evolution of project popularity, denoted by the number of stars over time, throughout the releases of both the PyTorch and TensorFlow projects. The PyTorch project has received a smaller number of stars in total as compared to TensorFlow (56.6k vs. 163.7k). Moreover, the number of stars for PyTorch grows 2.4 times slower than that for TensorFlow (0.8k vs. 1.9k), and exhibits distinct growth patterns. Specifically, the number of stars grows linearly in the PyTorch project with a growth rate of around 0.8k stars per month (R^2 = 0.995). Conversely, the number of stars in the TensorFlow project has experienced two turning points in growth rates: (1) an acceleration point in growth rate between release 0.11.0 and 1.0.0, and (2) a deceleration point in growth rate between release 1.14.0 and 2.0.0. The two turning points tend to align with the two major releases of TensorFlow, release 1.0.0 and 2.0.0, which exert divergent effects on the project popularity of TensorFlow.

Finding 8. PyTorch has accrued a smaller total number of stars over time, which grows linearly at an average rate 2.2 times slower than TensorFlow. The major releases 1.0.0 and 2.0.0 of TensorFlow tend to exert divergent impacts on the popularity of the project.

3.3.2 Causal Relations between Community Characteristics and Project Popularity. Figure 7 presents the time series graphs demonstrating causal relationships between community characteristics

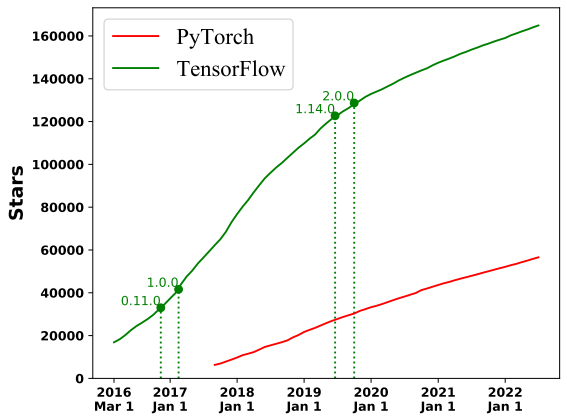


Figure 6. Popularity growth of deep learning projects across releases.

Table 6. Definition of variables in time series graphs.

Variable	Definition
S	Number of accumulated stars received by the project until month t.
C	Number of core developers in the community at month t.
P	Number of peripheral developers in the community at month t.
AU	Number of active users in the community at month t.
R	Retention rate of core developers in the community at month t.
I	Inflow rate of core developers in the community at month t.
D	Dropout rate of core developers in the community at month t.
$C2P$	Core to peripheral developers rate in the community at month t.

and project popularity along the time for both PyTorch (cf. Figure 7(a)) and TensorFlow (cf. Figure 7(b)), employing eight variables as illustrated in Table 6.

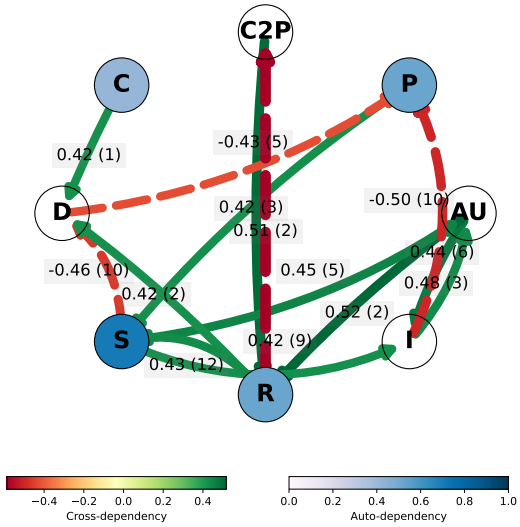
PyTorch. Initially, we investigate the causal links that connect with the node S, which represents project popularity. S is detected as the common cause of AU and I with positive effect sizes of 0.449 and 0.424, respectively, at time lags of 5 and 9 months. The positive causal links suggest that the upsurge in project popularity of month t leads to the increase in active users of month $t + 5$ and in the inflow rate of month $t + 9$. Meanwhile, S is identified as the cause of D with a negative effect size of -0.459 at a time lag of 10 months, indicating that a surge in project popularity of month t leads to a decline in the dropout rate for core developers of month $t + 10$. In addition, both P and R exhibit positive causal effects on S, with effect sizes of 0.425, 0.437 (and 0.432) at time lags of 3, 6 (and 12) months, respectively. The positive causal links indicate that an increase in the number of peripheral developers of month t results in an increase in project popularity of month $t + 3$, while a heightened retention rate of core developers of month t leads to increased project popularity of month $t + 6$ and $t + 12$.

Subsequently, we examine the causal relationships among the variables representing community characteristics. The contemporaneous causal link between $C2P$ (core2peripheral rate) and R (retention rate) suggests a negative coupling, with an effect size of 0.587. Moreover, the time-lagged causal loop between AU and I indicates the mutual positive causal effects between active users and the inflow rate with time lags. Additionally, the time-lagged causal chain $R \rightarrow D \rightarrow P$ implies that an increase in the retention rate (R) results in a reduction in the number of peripheral developers (P) in the long run, owing to a rise in the dropout rate (D).

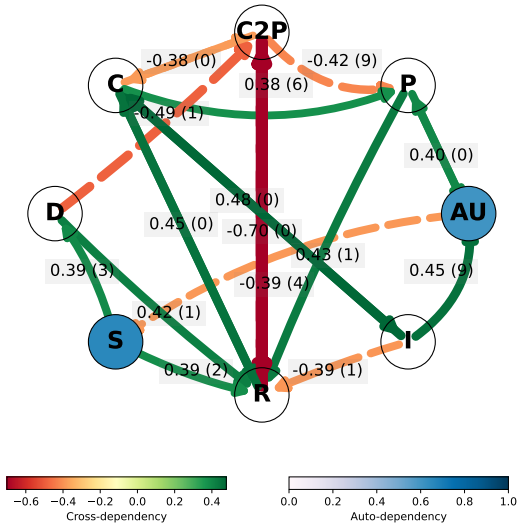
TensorFlow. We first look at the causal links associated with the node S, which denotes project popularity. S is detected as the cause of R with a positive effect size of 0.385 at a time lag of 2 months, indicating that the elevation in project popularity of month t leads to an increase in the retention rate of core developers of month $t + 2$.

We then turn to the causal links among the nodes representing the variables of community characteristics. C, R, I, and $C2P$ are fully coupled as suggested by the contemporaneous causal links. The contemporaneous causal link between P and AU suggests that the number of peripheral developers is positively coupled with the number of active users in the same month, with an effect size of 0.397. Moreover, the time-lagged causal chain $C, C2P \rightarrow P \rightarrow R$ indicates that the increase in the number of core developers (C) and the decrease in the core-to-peripheral rate ($C2P$) lead to increased retention rates in the long run, attributed to the expansion of peripheral developers.

PyTorch vs. TensorFlow. Comparatively, we observe two consistent causal links evident in the time series graphs of both PyTorch and TensorFlow. One is the negative contemporaneous causal



(a) PyTorch



(b) TensorFlow

Figure 7. Resulting time series graphs of community characteristics and project popularity for PyTorch and TensorFlow. Dashed and solid lines represent negative and positive causal links, respectively. The numbers on the edges, such as 0.42(9), indicate the strength of the relationship (0.42) and the time lag for causal links in months (9). Node colors depict the auto-dependency strength, and edge colors the cross-dependency strength at the lags with the maximum absolute MCI value. Nodes are labeled with the following abbreviations: **S**: number of accumulated stars; **C**: number of core developers; **P**: number of peripheral developers; **AU**: number of active users; **R**: retention rate of core developers; **I**: inflow rate of core developers; **D**: dropout rate of core developers; **C2P**: ratio of core to peripheral developers.

link between R and $C2P$, suggesting that retention and core2peripheral rates are negatively coupled in the same month. The other is the positive time-lagged causal link from I to AU, indicating that the influx of developers into the core development teams results in a subsequent increase of active users in the long run.

Finding 9. Project popularity has statistically significant and positive causal effects on the influx of new developers into core development teams and attracting active users in the PyTorch community, and on retaining core developers in the TensorFlow community. From the opposite perspective, the expansion of peripheral developers and the retention of core developers demonstrate statistically significant and positive causal effects on the popularity of PyTorch, while the expansion of active users exhibits statistically significant and negative causal effect on the popularity of TensorFlow.

4 DISCUSSION

We reflect on our findings of research questions and discuss implications for tool builders, researchers, and practitioners. We also highlight the avenues for future research.

4.1 Hypotheses

The count- and network-based operationalizations for classifying core and peripheral developers we used claim to be valid measures. If this is a matter of fact, we expect to reach consistent conclusions about whether a given developer is core or peripheral. Due to finite random sampling and sources of noise, we expect imperfect agreement between any two operationalizations even if they are consistent in capturing the same abstract concept. Nevertheless, if any two operationalizations give inconsistent results, the level of the agreement should be significantly greater than in the case of random assignment of contributor roles. Our null model for zero agreement is the amount of agreement that results from any two operationalizations that assign classes according to a Bernoulli process.

We use the Cohen's Kappa measure [24] to examine the agreement between two operationalizations for role classification, as a previous study did [51]:

$$\kappa = (p_o - p_e) / (1 - p_e)$$

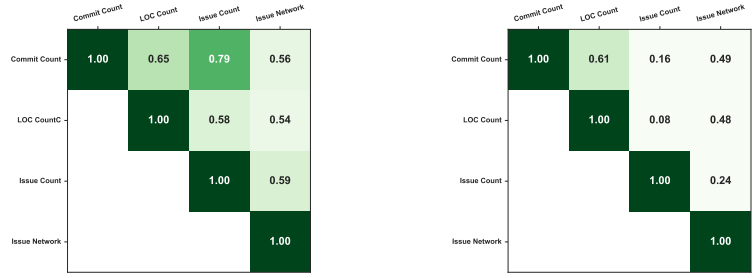
where p_o is the number of times the two operationalizations agree on the role of a developer, divided by the total number of developers, and where p_e is the expected probability of agreement when there is a random assignment of roles to developers. The interpretation of Kappa values is shown in Table 7.

Table 7. Interpretation of Kappa values.

Kappa value	Interpretation
<= 0	Poor agreement
[0.01, 0.20]	Slight agreement
[0.21, 0.40]	Fair agreement
[0.41, 0.60]	Moderate agreement
[0.61, 0.80]	Substantial agreement
[0.81, 1.00]	Almost perfect or perfect agreement

Figure 8 presents the resulting agreements between four operationalizations used to identify core and peripheral developers. For the operationalizations based on the version-control systems,

i.e., commit count and LOC count, we see substantial agreement (0.65 and 0.61) across the two communities.



(a) PyTorch (b) TensorFlow

Figure 8. Pairwise agreement of operationalizations for developer classification in deep learning communities.

In the PyTorch community, all comparisons show an agreement greater than "moderate" (i.e., with Kappa values greater than 0.4), which significantly exceeds the level of agreement expected by chance. The agreement level indicates that the four operationalizations do not lead to contradicted role classifications in the PyTorch community. On the contrary, in the TensorFlow community, the results of role classification based on issue count demonstrate merely slight to fair agreement with those based on commit count, LOC count, and issue network, with Kappa values ranging from 0.08 to 0.24. We further investigated the contributors who have frequently opened issues in the TensorFlow community. We observed a minimal overlap between those who raised issues and those who actively contributed code or participated in issue events, which indicates the potential existence of communication channels other than GitHub for issue tracking in the TensorFlow project (e.g., the official forum[3] for Tensorflow).

The inconsistency observed in the results implies that relying solely on a single operationalization might occasionally be insufficient for effective developer classification owing to incomplete data. Therefore, future work could explore approaches and techniques that systematically integrate various data sources, such as privileged events in GitHub issue discussions and pull requests [11], for enhancing the automatic classification of developer roles in OSS communities.

4.2 Implications

Structures of DL communities. The PyTorch and TensorFlow communities have 9.1% and 24.5% of their developers who take the roles as core developers (RQ1). The percentage of core developers in deep learning communities is considerably higher compared to that of other OSS communities studied in previous work. For instance, merely 3.9% (15 out of 388) developers were identified as core developers in the Apache community [66]; the percentage of core developers is even lower (0.02%, 3 out of 1,518) for some OSS projects hosted on SourceForge [26]. We consider this reasonable as GitHub, functioning as a social work environment, offers greater transparency compared to conventional open-source environments [91]. The incorporation of the pull request system in GitHub serves to standardize the contribution process, thereby lowering the barriers for contributions [74]. This heightened transparency and standardized process empower GitHub projects to attract a broader and more diverse spectrum of software practitioners. Besides, in

[3]https://discuss.tensorflow.org/

recent years, there has been a marked surge of interest in deep learning, alongside substantial corporate involvement in relevant OSS projects [41]. Such involvement may potentially instigate transformative shifts in the structures of OSS communities, as evidenced by prior studies [83]. Future research could systematically investigate the transformative shifts occurring in the structures of OSS communities, particularly in swiftly evolving domains, and evaluate how corporate involvement affects the shifts.

Contributions of common developers across DL communities. In RQ1, we identified 12 developers who hold core roles in both communities, wherein 33% exhibit spatial similarities in the pull requests they have contributed to the two projects. In addition, 33% of the 97 developers who hold core roles in one community and peripheral roles in the other demonstrate in their pull requests across the two projects. Prior work has also noted the tendency of software practitioners to engage in multiple open-source communities concurrently [80]. The resemblance in contributions might stem from the expertise of these practitioners. Therefore, coordinators in deep learning communities could consider historical code contributions from relevant projects while recruiting new developers. Future research could develop automatic approaches for identifying pertinent projects and potential candidates based on their code contributions, to facilitate the recruitment process.

Technical Background of DL Framework Developers. Over half of the core developers in PyTorch (64.4%) and TensorFlow (56.1%) are hired by the leading companies in the two communities, Meta and Google, in the development of the two deep learning frameworks (RQ1). Similar to other software systems like Linux and OpenStack, leading companies play a crucial role in the corresponding OSS communities [102]. Specifically, the leading companies not only serve as the primary sponsors of the corresponding deep learning frameworks, but also hire a large number of their core developers to contribute to these projects. Prior work indicates that the intensive participation of companies could benefit the sustainability of OSS projects, such as PyPI [92] and OpenStack [103]. Future studies could explore how the contributions of the employees in leading companies affect the development of deep learning frameworks.

Meanwhile, a notable percentage of core developers in the two communities hold Ph.D. degrees, with 25.9% in PyTorch and 21.9% in TensorFlow; some core developers are affiliated with top-tier universities such as Cornell, MIT, and UC Berkeley (RQ1). The educational background and research experience of core developers in the two communities suggest the participation of academia in the development of deep learning frameworks. Nonetheless, previous studies [44, 71, 79, 81] indicate that the collaboration between interdisciplinary groups (i.e., researchers and developers) in software development could be challenging due to the differences in their educational background, mindset, and end goals (research vs. software products). Future studies could investigate the unique challenges posed by the development of deep learning frameworks due to the collaboration between academia and industry.

Retention and dropout of core developers in DL communities. The TensorFlow community demonstrates a comparatively lower retention rate of core developers across its releases in contrast to PyTorch, approximately between 50% and 70% compared to between 60% and 80% observed in PyTorch (RQ2). Moreover, the relatively higher dropout rate of core developers in the TensorFlow community suggested that more core developers tend to leave the TensorFlow community in the long term, as compared to PyToch. Sustaining a high retention rate is pivotal in upholding the technical proficiency and productivity of OSS projects [108]. Considering the popularity of TensorFlow, future work could systematically investigate the impact of various factors, such as major releases, on the enduring engagement of core developers in the TensorFlow community.

Causal relationships between community characteristics and project popularity. Our study provides empirical evidence demonstrating the statistically significant and sizable causal

relationships between community characteristics and the popularity of deep learning projects (RQ3). In particular, the increase in the retention rate of core developers in the PyTorch community leads to an upsurge in project popularity within six months. Conversely, the increase in the popularity of the TensorFlow project causes a boost in the retention rate of core developers after two months. Prior study [13] identified statistically significant yet weak positive correlations between the number of stars and the quantify of contributors and commits in OSS projects. The positive causal relationships between retention rate and project popularity observed in our study emphasize the importance of incorporating effective strategies for OSS communities to retain core developers, including implementing transparent governance mechanisms like roadmaps [1] and establishing explicit reputation systems for contributors [63].

Generalization Implications beyond Case Study. Our study of PyTorch and TensorFlow revealed significant insights into how community structures and contributor roles evolve and influence project popularity. These findings, while derived from two prominent deep learning frameworks, may suggest broader implications applicable to other OSS projects. Firstly, the significant overlap of contributors between the two communities, with spatial and temporal similarities among the contributions (RQ1), indicates the existence of a common labor pool of contributors, which is also observed in prior work [31]. Understanding the labor pool across OSS projects offers a perspective for proposing effective recruitment and retention strategies. Future research could incorporate labor pool factor when analyzing the sustainability and success of OSS projects. In RQ2, we discovered a statistically significant decreasing trend in the inflow rates of core developers across both communities, indicating potential challenges in engaging new contributors as projects progress. Our findings regarding structured role transitions and the backgrounds of core contributors suggest patterns applicable to other OSS initiatives. OSS community managers and project leaders can use these principles to enhance engagement and foster project growth. Additionally, previous studies have extensively explored relations between the communities and popularity of OSS projects [10, 15, 59, 59, 75, 90]. We are the first to introduce PCMCI, a causal discovery technique, to evaluate the causal effects between community characteristics and project popularity. Our findings show that retention and core-to-peripheral rates are negatively coupled within the same month, and the influx of developers into core development teams results in a subsequent increase in active users over time (RQ3). OSS community managers and project leaders can leverage these causal relationships to improve community engagement and project popularity in their contexts. Nonetheless, it is crucial to consider the unique aspects of each project when generalizing these findings. Future research could validate or expand upon our findings across different OSS communities by employing causal discovery techniques and longitudinal analysis used in our study.

5 THREATS TO VALIDITY

Internall Validity. The cross-linking between the commit author and GitHub user may not always be correct, potentially threatening internal validity. We expect potential noise to be introduced due to the aliasing recognition heuristics. To mitigate such threat, we devised a series of heuristics for recognizing aliases of developers that are widely used in previous studies on OSS communities [39, 53, 93]. To evaluate the accuracy of cross-linking results, we randomly selected a sample of 100 pairs of matched commit authors and GitHub users from our dataset. We then manually examined their corresponding commit logs and GitHub pages. Out of the 100 pairs, 95 were evidently correct (95%), 4 were clearly incorrect (4%), and the remaining 1 (1%) could not be validated based on the information publicly available.

External Validity. We focus our comparable case study on two representative deep learning frameworks, PyTorch and TensorFlow, which have been the subjects of numerous empirical

studies in software engineering. Like other case studies, our findings on PyTorch and TensorFlow may not be directly generalizable to other deep learning frameworks, such as Keras, Google JAX, and Deeplearning4j, introducing potential threats to external validity. Nonetheless, the phenomenon we observed, along with the implications and insights we gained regarding PyTorch and TensorFlow, have the potential to guide the governance of other deep learning communities and a broader range of OSS communities.

Additionally, we collected historical data from the PyTorch and TensorFlow communities on GitHub up until August 1, 2022. This cutoff may exclude future activities in these communities, potentially posing threats to external validity. It is particularly noteworthy that the major release 2.0.0 of PyTorch, which occurred on March 15, 2023, may influence the evolution of the PyTorch community. To validate our findings regarding the evolution of deep learning communities, we extended the dataset through May 2024 and conducted experiments for RQ2 to examine the evolution of contributors with diverse roles, role ratios, and role transitions across multiple releases in both communities. Our extended analysis largely corroborates our initial findings, with only a few minor variations. In particular, the ratio of active users to developers in the PyTorch community generally exceeds that in the TensorFlow community ($\leq$ 2.0 vs. $\geq$ 2.4). The TensorFlow community experienced its lowest number of active users, 230, at the release of version 1.13.0. The trends in role transitions for both communities remain consistent, although the significance levels have increased, as indicated by the changes in p-values. Notably, the PyTorch 2.0.0 major release exhibits a comparable impact on the evolution of the PyTorch community as observed with the preceding major release 1.0.0. Specifically, the 2.0.0 release is associated with substantial growth in the numbers of contributors: active users increased from 955 to 1723, peripheral developers from 264 to 447, and core developers from 108 to 121. Furthermore, substantial increases were observed in multiple role ratios, including the peripheral-to-core developer ratio (from 2.4 to 3.7), the active-user-to-developer ratio (from 2.6 to 3.0), and the dropout rate (from 0.07 to 0.18). In contrast, the release also corresponded with decreases in the inflow rate (from 0.22 to 0.19), the retention rate (from 0.69 to 0.63), and the core-to-peripheral ratio (from 0.20 to 0.19). These findings indicate that our collected dataset is robust and sufficiently comprehensive to accurately characterize the evolution of deep learning communities.

6 RELATED WORK

6.1 Software Engineering for DL and ML

In software systems empowered by DL/ML capabilities, DL/ML contributes individual or multiple components to larger systems that also integrate non-DL/ML components, with their development facilitated by DL/ML frameworks and platforms. Considerable attention in practice focuses on building robust pipelines for training and deploying DL/ML models in a scalable fashion, frequently referred to by "AI engineering", "AIOps" and "MLOps" [29, 58, 60].

Some researchers have conducted empirical studies aimed at exploring a range of technical aspects in DL/ML frameworks, platforms, and software systems, such as bugs in TensorFlow [101] and ML software systems [89], faults in DL frameworks (TensorFlow, Keras and PyTorch) [46], program failures of jobs on a DL platform in Microsoft [99]. Other researchers have focused on investigating software engineering practices involved in the development of DL/ML systems, including shifts in software engineering practices [5, 96], testing methodology tailed for these systems [17, 73], misuse and updates of libraries [30, 94], refactoring practice [86], collaboration in building these systems [67], and engineering challenges gleaned from Stack Overflow posts [21, 37, 95, 100].

Nonetheless, little is known about the OSS communities that contribute to the development of DL frameworks, thus our study aims to address this gap.

6.2 Empirical Studies on OSS Communities

Prior work has characterized the joining process in open-source communities as one that follows the onion model [68]. Newcomers start as observers and passive users (outermost layer), eventually moving up the hierarchy to become contributors and then core developers (innermost layer). The inner layers are associated with more technical and higher reputation roles, and the movement across the layers is referred to as a migration [49]. The distinction between the different roles of developers is represented as a dichotomy consisting of core and peripheral developers. Core developers play a critical role in developing system architectures and shaping leadership structures. In contrast, peripheral developers are irregularly involved in bug fixes or small-scale improvements. Researchers have proposed diverse operationalizations for classifying contributors in OSS communities, including count-based metrics [51, 66], network-based metrics related to bug trackers [26], change logs [62], mailing lists [51], access to core project files [98], and participation in privileged events [11]. In our study, we adopt four operationalizations, including both count- and network-based metrics, to classify developers contributing to the development and evolvement of deep learning frameworks. Additionally, we conduct a comparative evaluation of the outcomes derived from the operationalizations.

Researchers have intensely investigated diverse factors that influence the sustainability of OSS communities, including enduring engagement of contributors [107], developer turnover [33], social capital [76], activities carried out by elite developers [97], onboarding processes [34], and the involvement and collaboration of corporate entities [104]. Moreover, previous studies have investigated the factors that affect the popularity of OSS projects, encompassing considerations like programming languages [9, 13], application domains [13], social media [32], and documentation updates [4]. Different from the studies above, we focus our research on characterizing OSS communities of deep learning frameworks in terms of their structures and evolution patterns. Additionally, we aim to explore the causal relationships between community characteristics and the popularity of deep learning frameworks.

7 SUMMARY AND FUTURE WORK

In this paper, we conducted a comparative study to empirically investigate the characteristics of open-source communities associated with two representative deep learning frameworks, PyTorch and TensorFlow. Specifically, we characterized various contributor roles in the communities as well as the evolution of community structures and role transitions across project releases, and explored the impact of community characteristics on project popularity. We find that the TensorFlow community hosts a larger contributor base, featuring a higher proportion of core developers in its development team and a more extensive cohort of active users compared to the PyTorch community. In the two communities, 64.4% and 56.1% of developers are employed by the leading companies behind the projects, while 25.9% and 21.9% of core developers hold Ph.D. degrees, and 77.2% and 77.7% contribute to other ML/DL open-source projects. Developers shared between the two communities exhibit spatial and temporal similarity to some extent in their pull requests across their respective projects. The evolution of contributors with various roles exhibits a consistent upward trend over time in the PyTorch community. Conversely, the TensorFlow community experiences a noticeable turning point in its evolution. Both communities demonstrate a statistically significant decreasing trend in the inflow rates of core developers. Furthermore, the study reveals that the expansion of peripheral developers and the retention of core developers demonstrate statistically significant and positive causal effects on the popularity of PyTorch, while the expansion of active users exhibits statistically significant and negative causal effects on the popularity of TensorFlow.

Future work could devise automatic approaches that integrate various data sources for effective developer classification in OSS communities, and consider the causal effects identified between community characteristics and project popularity when formulating strategies for sustaining deep learning communities.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation of China (No. 62472383 and No. 62102358), the Natural Science Foundation of Zhejiang Province (No. LQ21F020008), and the Ministry of Education, Singapore under its Academic Research Fund Tier 3 (Award ID: MOET32020-0004). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore. We would like to thank Chenyue Ma for his valuable contributions to the early-stage exploration of this research as part of his undergraduate thesis, which laid a foundation for the continued progress of this research.

REFERENCES

[1] 2023. TensorFlow Core. https://www.tensorflow.org/guide

[2] 2023. Triton Inference Server: An Optimized Cloud and Edge Inferencing Solution. https://github.com/triton-inference-server/server

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[4] Karan Aggarwal, Abram Hindle, and Eleni Stroulia. 2014. Co-Evolution of Project Documentation and Popularity within Github. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 360–363. https://doi.org/10.1145/2597073.2597120

[5] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[6] BigPicture. 2024. BigPicture Documentation: Free Datasets - Companies. https://docs.bigpicture.io/docs/free-datasets/companies/. Accessed: July 8, 2024.

[7] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *J. Mach. Learn. Res.* 20 (2019), 28:1–28:6. http://jmlr.org/papers/v20/18-403.html

[8] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*. 137–143.

[9] Tegawendé F. Bissyandé, Ferdian Thung, David Lo, Lingxiao Jiang, and Laurent Réveillère. 2013. Popularity, Interoperability, and Impact of Programming Languages in 100,000 Open Source Projects. In *2013 IEEE 37th Annual Computer Software and Applications Conference*. 303–312. https://doi.org/10.1109/COMPSAC.2013.55

[10] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. 2016. Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information and Software Technology* 70 (2016), 30–39.

[11] Thomas Bock, Nils Alznauer, Mitchell Joblin, and Sven Apel. 2023. Automatic Core-developer Identification on GitHub: A Validation Study. *ACM Transactions on Software Engineering and Methodology* (2023).

[12] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Predicting the popularity of github repositories. In *Proceedings of the The 12th international conference on predictive models and data analytics in software engineering*. 1–10.

[13] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 334–344.

[14] Hudson Borges and Marco Tulio Valente. 2018. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software* 146 (2018), 112–129.
[15] Hudson Silva Borges and Marco Tulio Valente. 2019. How do developers promote open source projects? *Computer* 52, 8 (2019), 27–33.
[16] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs*. http://github.com/google/jax
[17] Houssem Ben Braiek and Foutse Khomh. 2020. On testing machine learning programs. *Journal of Systems and Software* 164 (2020), 110542.
[18] Ulrik Brandes. 2005. *Network analysis: methodological foundations*. Vol. 3418. Springer Science & Business Media.
[19] Junming Cao, Bihuan Chen, Chao Sun, Longjie Hu, Shuaihong Wu, and Xin Peng. 2022. Understanding performance problems in deep learning systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 357–369.
[20] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 2722–2730. https://doi.org/10.1109/ICCV.2015.312
[21] Zhenpeng Chen, Yanbin Cao, Yuanqiang Liu, Haoyu Wang, Tao Xie, and Xuanzhe Liu. 2020. A Comprehensive Study on Challenges in Deploying Deep Learning Based Software. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 750–762. https://doi.org/10.1145/3368089.3409759
[22] Mihai Cristian Chirodea, Ovidiu Constantin Novac, Cornelia Mihaela Novac, Nicu Bizon, Mihai Oproescu, and Cornelia Emilia Gordan. 2021. Comparison of tensorflow and pytorch in convolutional neural network-based applications. In *2021 13th international conference on electronics, computers and artificial intelligence (ECAI)*. IEEE, 1–6.
[23] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.
[24] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.
[25] NVIDIA Corporation. 2024. *Fuser: A Fusion Code Generator for NVIDIA GPUs (commonly known as "nvFuser")*. https://github.com/NVIDIA/Fuser
[26] K. Crowston, Kangning Wei, Qing Li, and J. Howison. 2006. Core and Periphery in Free/Libre and Open Source Software Team Communications. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, Vol. 6. 118a–118a. https://doi.org/10.1109/HICSS.2006.101
[27] Hulin Dai, Xuan Peng, Xuanhua Shi, Ligang He, Qian Xiong, and Hai Jin. 2022. Reveal training performance mystery between TensorFlow and PyTorch in the single GPU environment. *Science China Information Sciences* 65 (2022), 1–17.
[28] Jurafsky Dan and James H. Martin. 2014. *Speech and Language Processing*. Vol. 3. Pearson London.
[29] Yingnong Dang, Qingwei Lin, and Peng Huang. 2019. AIOps: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 4–5.
[30] Malinda Dilhara, Ameya Ketkar, and Danny Dig. 2021. Understanding Software-2.0: A Study of Machine Learning Library Usage and Evolution. *ACM Trans. Softw. Eng. Methodol.* 30, 4, Article 55 (jul 2021), 42 pages. https://doi.org/10.1145/3453478
[31] Hongbo Fang, James Herbsleb, and Bogdan Vasilescu. 2023. Matching Skills, Past Collaboration, and Limited Competition: Modeling When Open-Source Projects Attract Contributors. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 42–54.
[32] Hongbo Fang, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. 2022. " This is damn slick!" estimating the impact of tweets on open source project popularity and new contributors. In *Proceedings of the 44th International Conference on Software Engineering*. 2116–2129.
[33] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. 2015. Impact of Developer Turnover on Quality in Open-Source Software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 829–841. https://doi.org/10.1145/2786805.2786870
[34] Armstrong Foundjem, Ellis Eghan, and Bram Adams. 2021. Onboarding vs. Diversity, Productivity and Quality — Empirical Study of the OpenStack Ecosystem. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1033–1045. https://doi.org/10.1109/ICSE43902.2021.00097
[35] Jingyue Gao, Xiting Wang, Yasha Wang, Zhao Yang, Junyi Gao, Jiangtao Wang, Wen Tang, and Xing Xie. 2019. CAMP: Co-Attention Memory Networks for Diagnosis Prediction in Healthcare. In *2019 IEEE International Conference on Data Mining (ICDM)*. 1036–1041. https://doi.org/10.1109/ICDM.2019.00120

[36] Kai Gao, Runzhi He, Bing Xie, and Minghui Zhou. 2024. Characterizing deep learning package supply chains in PyPI: Domains, clusters, and disengagement. *ACM Transactions on Software Engineering and Methodology* 33, 4 (2024), 1–27.

[37] Kai Gao, Zhixing Wang, Audris Mockus, and Minghui Zhou. 2022. On the variability of software engineering needs for deep learning: Stages, trends, and application types. *IEEE Transactions on Software Engineering* 49, 2 (2022), 760–776.

[38] Adam Gibson et al. 2014. Deeplearning4j. https://github.com/deeplearning4j/deeplearning4j.

[39] Mathieu Goeminne and Tom Mens. 2013. A comparison of identity merge algorithms for software repositories. *Science of Computer Programming* 78, 8 (2013), 971–986.

[40] Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. 2021. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *Journal of Systems and Software* 175 (may 2021). https://doi.org/10.1016/j.jss.2021.110911

[41] Mariam Guizani, Aileen Abril Castro-Guzman, Anita Sarma, and Igor Steinmacher. 2023. Rules of Engagement: Why and How Companies Participate in OSS. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 2617–2629. https://doi.org/10.1109/ICSE48619.2023.00218

[42] Khaled H Hamed and A Ramachandra Rao. 1998. A modified Mann-Kendall trend test for autocorrelated data. *Journal of hydrology* 204, 1-4 (1998), 182–196.

[43] Cristina Heghedus, Antorweep Chakravorty, and Chunming Rong. 2019. Neural network frameworks. comparison on public transportation prediction. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 842–849.

[44] Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P Pawlowski, Eric T Phipps, et al. 2005. An overview of the Trilinos project. *ACM Transactions on Mathematical Software (TOMS)* 31, 3 (2005), 397–423.

[45] Hipo. 2024. University Domains List. https://github.com/Hipo/university-domains-list. Accessed: July 8, 2024.

[46] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of Real Faults in Deep Learning Systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1110–1121. https://doi.org/10.1145/3377811.3380395

[47] Md. Hussain and Ishtiak Mahmud. 2019. pyMannKendall: a python package for non parametric Mann Kendall family of trend tests. *Journal of Open Source Software* 4, 39 (25 7 2019), 1556. https://doi.org/10.21105/joss.01556

[48] Daniel Izquierdo-Cortazar, Gregorio Robles, Felipe Ortega, and Jesus M Gonzalez-Barahona. 2009. Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *2009 42nd Hawaii International Conference on System Sciences*. IEEE, 1–10.

[49] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. 2011. The Onion Patch: Migration in Open Source Ecosystems. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 70–80. https://doi.org/10.1145/2025113.2025127

[50] Li Jia, Hao Zhong, Xiaoyin Wang, Linpeng Huang, and Xuansheng Lu. 2021. The symptoms, causes, and repairs of bugs inside a deep learning library. *Journal of Systems and Software* 177 (2021), 110935.

[51] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. 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, 164–174.

[52] Mitchell Joblin, Barbara Eckl, Thomas Bock, Angelika Schmid, Janet Siegmund, and Sven Apel. 2023. Hierarchical and Hybrid Organizational Structures in Open-Source Software Projects: A Longitudinal Study. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–29.

[53] Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, and Mark GJ Van Den Brand. 2012. Who's who in Gnome: Using LSA to merge software repository identities. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 592–595.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (may 2017), 84–90. https://doi.org/10.1145/3065386

[55] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[56] Hao Li, Gopi Krishnan Rajbahadur, and Cor-Paul Bezemer. 2024. Studying the Impact of TensorFlow and PyTorch Bindings on Machine Learning Software Quality. *ACM Transactions on Software Engineering and Methodology* (2024).

[57] Chen Liu, Jie Lu, Guangwei Li, Ting Yuan, Lian Li, Feng Tan, Jun Yang, Liang You, and Jingling Xue. 2021. Detecting TensorFlow program bugs in real-world industrial environment. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 55–66.

[58] Sasu Mäkinen, Henrik Skogström, Eero Laaksonen, and Tommi Mikkonen. 2021. Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software*

Engineering for AI (WAIN). IEEE, 109–112.
[59] Danaja Maldeniya, Ceren Budak, Lionel P Robert Jr, and Daniel M Romero. 2020. Herding a deluge of good samaritans: How github projects respond to increased attention. In *Proceedings of The Web Conference 2020*. 2055–2065.
[60] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2022. Software engineering for AI-based systems: a survey. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–59.
[61] Brian McMahan and Delip Rao. 2018. Listening to the World Improves Speech Command Recognition. In *AAAI*.
[62] Andrew Meneely and Laurie Williams. 2011. Socio-technical developer networks: should we trust our measurements?. In *2011 33rd International Conference on Software Engineering (ICSE)*. 281–290. https://doi.org/10.1145/1985793.1985832
[63] Courtney Miller, Christian Kästner, and Bogdan Vasilescu. 2023. "We Feel Like We're Winging It:" A Study on Navigating Open-Source Dependency Abandonment. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1281–1293.
[64] Ran Mo, Yao Zhang, Yushuo Wang, Siyuan Zhang, Pu Xiong, Zengyang Li, and Yang Zhao. 2023. Exploring the Impact of Code Clones on Deep Learning Software. *ACM Transactions on Software Engineering and Methodology* 32, 6 (2023), 1–34.
[65] Audris Mockus. 2009. Organizational volatility and developer productivity. In *ICSE Workshop on Socio-Technical Congruence*.
[66] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.* 11, 3 (jul 2002), 309–346. https://doi.org/10.1145/567793.567795
[67] Nadia Nahar, Shurui Zhou, Grace Lewis, and Christian Kästner. 2022. Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process. In *Proceedings of the 44th International Conference on Software Engineering*. 413–425.
[68] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. 2002. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*. 76–85.
[69] okutbay. 2024. List of Free Email Provider Domains - GitHub Gist. https://gist.github.com/okutbay/5b4974b70673dfdcc21c517632c1f984. Accessed: July 8, 2024.
[70] OpenXLA Contributors. 2023. XLA: Accelerated Linear Algebra. https://github.com/openxla/xla. Accessed on [Date Accessed].
[71] Drew Paine and Charlotte P Lee. 2017. " Who Has Plots?" Contextualizing Scientific Software, Practice, and Visualizations. *Proceedings of the ACM on human-computer interaction* 1, CSCW (2017), 1–21.
[72] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).
[73] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.
[74] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. 2013. Creating a shared understanding of testing culture on a social coding site. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 112–121.
[75] Huilian Sophie Qiu, Yucen Lily Li, Susmita Padala, Anita Sarma, and Bogdan Vasilescu. 2019. The signals that potential contributors look for when choosing open-source projects. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–29.
[76] Huilian Sophie Qiu, Alexander Nolte, Anita Brown, Alexander Serebrenik, and Bogdan Vasilescu. 2019. Going Farther Together: The Impact of Social Capital on Sustained Participation in Open Source. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 688–699. https://doi.org/10.1109/ICSE.2019.00078
[77] Jakob Runge, Andreas Gerhardus, Gherardo Varando, Veronika Eyring, and Gustau Camps-Valls. 2023. Causal inference for time series. *Nature Reviews Earth & Environment* (2023), 1–19.
[78] Jakob Runge, Peer Nowack, Marlene Kretschmer, Seth Flaxman, and Dino Sejdinovic. 2019. Detecting and quantifying causal associations in large nonlinear time series datasets. *Science advances* 5, 11 (2019), eaau4996.
[79] Judith Segal. 2008. Scientists and software engineers: A tale of two cultures. (2008).
[80] Xiaotao Song, Jiafei Yan, Yuexin Huang, Hailong Sun, and Hongyu Zhang. 2022. A Collaboration-Aware Approach to Profiling Developer Expertise with Cross-Community Data. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 344–355.
[81] Tim Storer. 2017. Bridging the chasm: A survey of software engineering practice in scientific programming. *ACM Computing Surveys (CSUR)* 50, 4 (2017), 1–32.
[82] GJ Székely, ML Rizzo, and NK Bakirov. 2007. Measuring and testing dependence by correlation of distances. *Annals of Statistics* 35, 6 (2007), 2769–2794.

[83] Damian A Tamburri, Rick Kazman, and Hamed Fahimi. 2022. On the Relationship Between Organizational Structure Patterns and Architecture in Agile Teams. *IEEE Transactions on Software Engineering* 49, 1 (2022), 325–347.

[84] Damian A Tamburri, Patricia Lago, and Hans van Vliet. 2013. Organizational social structures for software engineering. *ACM Computing Surveys (CSUR)* 46, 1 (2013), 1–35.

[85] Xin Tan, Kai Gao, Minghui Zhou, and Li Zhang. 2022. An exploratory study of deep learning supply chain. In *Proceedings of the 44th International Conference on Software Engineering*. 86–98.

[86] Yiming Tang, Raffi Khatchadourian, Mehdi Bagherzadeh, Rhia Singh, Ajani Stewart, and Anita Raja. 2021. An Empirical Study of Refactorings and Technical Debt in Machine Learning Systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 238–250. https://doi.org/10.1109/ICSE43902.2021.00033

[87] IREE Team. 2023. IREE: A modular and portable ML IR and runtime. https://github.com/iree-org/iree.

[88] TensorFlow Team. 2023. TensorFlow Profiler. https://github.com/tensorflow/profiler.

[89] Ferdian Thung, Shaowei Wang, David Lo, and Lingxiao Jiang. 2012. An Empirical Study of Bugs in Machine Learning Systems. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*. 271–280. https://doi.org/10.1109/ISSRE.2012.22

[90] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. 2018. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th international conference on software engineering*. 511–522.

[91] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th international conference on Software engineering*. 356–366.

[92] Marat Valiev, Bogdan Vasilescu, and James Herbsleb. 2018. Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 644–655.

[93] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. 2015. Gender and tenure diversity in GitHub teams. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 3789–3798.

[94] Chengcheng Wan, Shicheng Liu, Henry Hoffmann, Michael Maire, and Shan Lu. 2021. Are Machine Learning Cloud APIs Used Correctly?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 125–137. https://doi.org/10.1109/ICSE43902.2021.00024

[95] Zhiyuan Wan, Jiaheng Tao, Jiakun Liang, Zhengong Cai, Cheng Chang, Lin Qiao, and Qiaoni Zhou. 2019. Large-scale empirical study on machine learning related questions on Stack Overflow. *J. ZheJiang University (Eng. Sci.),* 53, 5 (2019), 819–828. https://doi.org/10.3785/j.issn.1008-973X.2019.05.001

[96] Zhiyuan Wan, Xin Xia, David Lo, and Gail C. Murphy. 2021. How does Machine Learning Change Software Development Practices? *IEEE Transactions on Software Engineering* 47, 9 (2021), 1857–1871. https://doi.org/10.1109/TSE.2019.2937083

[97] Zhendong Wang, Yang Feng, Yi Wang, James A. Jones, and David Redmiles. 2020. Unveiling Elite Developers' Activities in Open Source Projects. *ACM Trans. Softw. Eng. Methodol.* 29, 3, Article 16 (jun 2020), 35 pages. https://doi.org/10.1145/3387111

[98] Zhefu Wu, Tiantong Zhu, Qi Xuan, and Yue Yu. 2018. Evaluation of Core Developers in Open Source Software by Contribution Allocation. *Journal of Software* 29, 8 (2018), 2272–2282. https://doi.org/10.13328/j.cnki.jos.005521

[99] Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. 2020. An Empirical Study on Program Failures of Deep Learning Jobs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1159–1170. https://doi.org/10.1145/3377811.3380362

[100] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. 2019. An Empirical Study of Common Challenges in Developing Deep Learning Applications. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. 104–115. https://doi.org/10.1109/ISSRE.2019.00020

[101] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An empirical study on tensorflow program bugs. In *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis*. 129–140.

[102] Yuxia Zhang, Hui Liu, Xin Tan, Minghui Zhou, Zhi Jin, and Jiaxin Zhu. 2022. Turnover of companies in OpenStack: Prevalence and rationale. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 4 (2022), 1–24.

[103] Yuxia Zhang, Klaas-Jan Stol, Hui Liu, and Minghui Zhou. 2022. Corporate dominance in open source ecosystems: a case study of OpenStack. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1048–1060.

[104] Yuxia Zhang, Minghui Zhou, Klaas-Jan Stol, Jianyu Wu, and Zhi Jin. 2020. How do companies collaborate in open source ecosystems? an empirical study of openstack. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1196–1208.

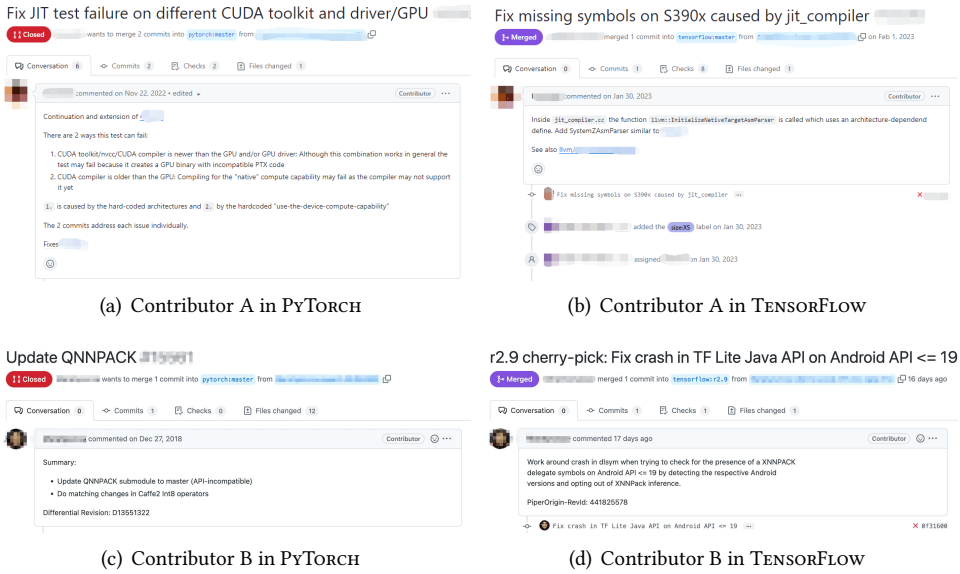
[105] Zejun Zhang, Yanming Yang, Xin Xia, David Lo, Xiaoxue Ren, and John Grundy. 2021. Unveiling the mystery of API evolution in Deep Learning frameworks: a case study of TensorFlow 2. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 238–247.

[106] Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. 2017. On the scalability of linux kernel maintainers' work. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 27–37.

[107] Minghui Zhou and Audris Mockus. 2012. What make long term contributors: Willingness and opportunity in OSS community. In *2012 34th International Conference on Software Engineering (ICSE)*. 518–528. https://doi.org/10.1109/ICSE.2012.6227164

[108] Minghui Zhou, Audris Mockus, Xiujuan Ma, Lu Zhang, and Hong Mei. 2016. Inflow and retention in oss communities with commercial involvement: A case study of three hybrid projects. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 2 (2016), 1–29.

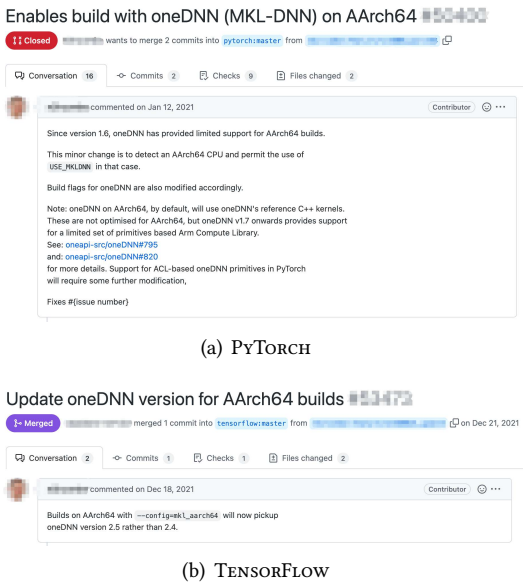
A APPENDIX



(a) PyTorch (b) TensorFlow

(c) Contributor B in PyTorch (d) Contributor B in TensorFlow

Figure 9. Examples pull requests of two common core developers across communities (Contributor A and B).



(a) PyTorch

(b) TensorFlow

Figure 10. Example pull requests of a common developer across communities, who takes the peripheral role in the PyTorch community and the core role in the TensorFlow community.